The Black-Box Simplex Architecture for Runtime Assurance of Multi-Agent CPS

Sanaz Sheikhi[1], Usama Mehmood[2], Stanley Bak[1*], Scott A. Smolka[1*] and Scott D. Stoller[1*]

[1]Department of Computer Science, Stony Brook University.
[2]Department of Computer Science, Information Technology University.

*Corresponding author(s). E-mail(s): sbak@cs.stonybrook.edu; sas@cs.stonybrook.edu; stoller@cs.stonybrook.edu;

Abstract

The Simplex Architecture is a runtime assurance framework where control authority may switch from an unverified and potentially unsafe *advanced controller* to a backup *baseline controller* in order to maintain the safety of an autonomous cyber-physical system. In this work, we show that runtime checks can replace the requirement to statically verify safety of the baseline controller. This is important as there are many powerful control techniques, such as model-predictive control and neural network controllers, that work well in practice but are difficult to statically verify. Since the method does not use internal information about the advanced or baseline controller, we call the approach the *Black-Box Simplex Architecture*. We prove the architecture is safe and present two case studies where (i) model-predictive control provides safe multi-robot coordination, and (ii) neural networks provably prevent collisions in groups of F-16 aircraft, despite the controllers occasionally outputting unsafe commands. We further show how to safely blend commands from the advanced and baseline controllers in multi-agent systems, reducing the performance impact when switching is necessary to preserve safety.

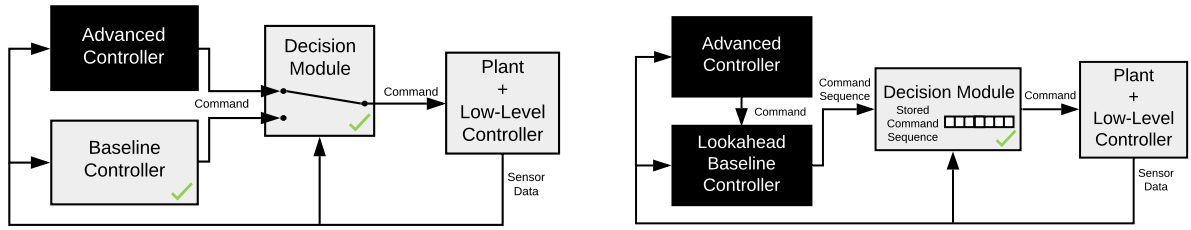
Keywords: Black-Box Simplex, Runtime Assurance, Autonomous CPS, Multi-agent

1 Introduction

Autonomous cyber-physical systems (CPS) have the potential to transform vital domains such as transportation, health-care, and energy management. As these systems perform complex functions, they often require complex designs. Moreover, since autonomous CPS interact with the physical world, they are typically safety-critical. Formal analysis, however, can be difficult for complex systems.

In the development of such CPS, powerful control techniques such as model-predictive control and deep reinforcement learning are increasingly being used instead of traditional controller design techniques. Such trends exacerbate the safety verification problem. Additionally, there is increasing interest in systems that can *learn in the field*, changing their behaviors based on observations. Classical verification strategies are poorly suited for such designs.

One approach for dynamically providing safety for systems with complex and unverified components is *runtime assurance* [1], where the state of the plant is monitored at runtime to mitigate possible imminent violations of formal properties.



(a) Traditional Simplex Architecture

(b) Black-Box Simplex Architecture

Fig. 1: The Black-Box Simplex Architecture guarantees safety despite a black-box advanced controller and a black-box baseline controller.

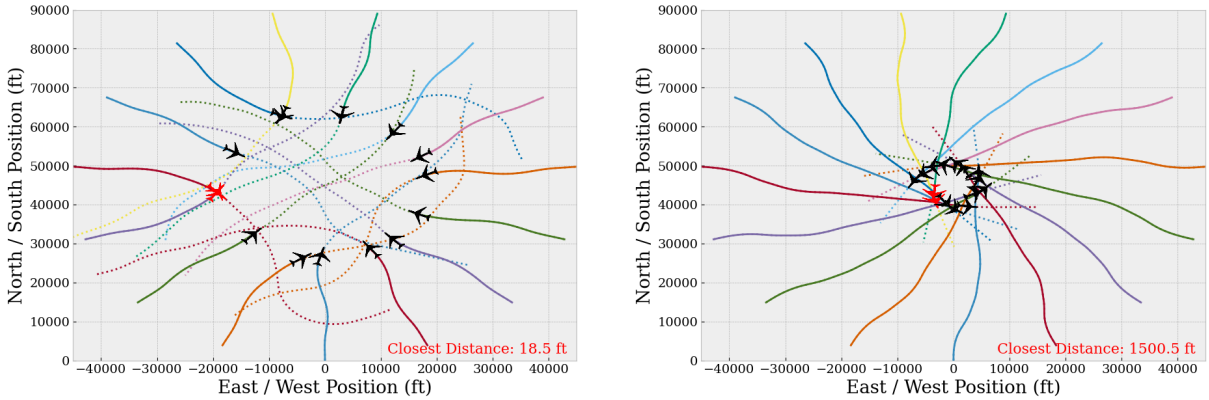
A well-known runtime assurance technique is the Simplex Control Architecture [2, 3], which has been applied to a wide range of systems [4–6]. In the original Simplex Architecture, shown in Figure 1(a), the *baseline controller* (BC) and the *decision module* (DM) are part of the trusted computing base. The DM monitors the state of the system and switches control from the *advanced controller* (AC) to the BC if using the former could result in a safety violation in the near future. The original Simplex Architecture requires creating a provably safe BC, which can be difficult. In this work, we eliminate this requirement through a greater reliance on runtime verification.

In the proposed *Black-Box Simplex Architecture* (BSA), shown in Figure 1(b), the BC (now referred to as the *Lookahead Baseline Controller* (LBC), no longer needs to be statically verified, and can even be incorrect. The tradeoff is that the DM performs more extensive runtime checking and stores backup command sequences from previous computation steps. The DM performs simulation or reachability analysis based on a known system model. If the DM's computation time is too large, BSA keeps the system safe by switching control to a stored command sequence generated at an earlier step by the LBC and checked for safety by the DM. We note that in case the DM's computation timeouts are too frequent, BSA is not well suited for the system, as it causes performance degradation.

For our method, an initial permanently safe command sequence is required that keeps the system safe from the start state. This should be much easier to provide than a safety controller typically used in Simplex which must guarantee safety from any state the advanced controller is allowed to reach. The specifics of the approach will be discussed in Section 2.

We prove two theorems about this architecture: (i) safety is always guaranteed, and (ii) when the baseline and advanced controllers perform well (to be formally defined in Section 2), the architecture is transparent: the advanced controller appears to have full control of the system. The practicality of these assumptions and the utility of the BSA architecture itself is demonstrated through two significant case studies. In the first, a multi-robot coordination system uses a BC based on a model-predictive control algorithm with a potential-field approach for collision avoidance. Such a setup is difficult to statically verify as it depends on the online solution of a nonlinear optimization problem. In the second, a mid-air collision avoidance system for groups of F-16 aircraft is created from imperfect logic encoded in neural networks. A preview of the second case study is shown in Figure 2, where directly using the neural networks causes a collision (left), but the Black-Box Simplex approach safely navigates the scenario, resulting in an emergent maneuver similar to a roundabout (right).

Here, we extend an earlier version of these works [7, 8] in two ways. First, we adapt the developed theory of Black-Box Simplex to multi-agent systems, and present *command blending* as a technique to safely combine commands from the advanced and baseline controllers (Section 3). Second, we implement the command blending decision module in the multi-robot case study and demonstrate the improvement to the system's performance (Section 4.2).



(a) Original System (unsafe, the two red aircraft collide)

(b) Black-Box Simplex (safe, snapshot shown at closest distance)

Fig. 2: Black-Box Simplex safely navigates complex scenarios. In the 15-aircraft case, all aircraft cross the circle while maintaining a 1500 ft separation distance.

2 Black-Box Simplex

The traditional Simplex Architecture, shown in Figure 1(a), preserves the safety of the system while permitting the use of an unverified AC. It does this by using the AC in conjunction with a verified BC and a verified DM. The DM cannot simply check if the next state is safe, as cyber-physical systems have inertia and it may be too late to take corrective action. Rather, the verified design of a Simplex system usually requires offline reasoning with respect to a trusted BC and the system dynamics.

If the system dynamics are linear and the admissible states are defined with linear constraints, a state-feedback BC and a DM can be synthesized by solving a linear matrix inequality [2]. If the system dynamics or constraints are nonlinear, however, there is no direct approach to create a trusted BC and DM. This prevents more widespread use of the traditional Simplex Architecture.

The proposed Black-Box Simplex Architecture removes the requirement that the BC is statically verified, allowing provable safety with both an unverified AC and an unverified BC. Its architecture is shown in Figure 1(b). Apart from eliminating the need to establish safety of the BC, BSA differs from the traditional Simplex Architecture in other important ways. First, the AC shares its command with the LBC instead of passing it directly to the DM. Second, the LBC uses this command as the starting point of a *candidate safe command sequence*.

Candidate command sequences may be generated using state-of-the-art controller designs, including neural networks trained with reinforcement learning or MPC. Note that a candidate command sequence is not guaranteed to be safe until it is verified by the DM through a runtime check. Specifically, the DM checks safety of the LBC's candidate command sequence, rejecting it if safety is not ensured. The DM checks safety by running simulations (rollouts) for deterministic systems; for systems with uncertainty, it performs online reachability computation [9–11]. BSA is able to maintain the safety of the system in case the DM is unable to complete the online verification of the proposed command sequence in time. If the computation takes too long, the DM aborts and switches to a backup command sequence that continues to ensure system safety. It can subsequently switch back to the AC when the runtime checks finish in time.

As long as the AC drives the system through states from which the LBC can recover, it continues to actuate the system. However, if the LBC fails to compute a candidate command sequence that maintains safety—due to a fault of the unverified AC or the unverified BC, or due to excessive computation time for any of the components—the DM can still recover the system using the safe command sequence from the previous step.

Note that the DM does not generate any command sequences. It only performs runtime checks and stores command sequences to maintain a safe backup plan at all times.

The applicability of BSA depends on the feasibility of two system-specific steps: (i) constructing candidate command sequences and (ii) proving their safety at runtime. For some systems, a safe command sequence can simply bring the system to a stop. An autonomous car, for example, could have a safe command sequence that steers the car to the side of the road and then stops. A safe sequence for a drone might direct it to the closest emergency landing location. For a rapidly-moving autonomous fixed-wing aircraft swarm, a safe sequence could fly all aircraft in non-intersecting circles to allow time for human intervention. Proving safety of a given command sequence can also be challenging and depends on the system dynamics. For nondeterministic systems, this could involve performing reachability computations at runtime [9–11]. Such techniques assume an accurate system model is available in order to compute reachable sets. Notice that traditional offline control theory also requires this assumption, so we do not view it as overly burdensome.

In BSA, although both controllers are unverified, we do not combine them into a single unverified controller, even if the AC is MPC-based and produces command sequences. This allows for a logical separation of concerns, where the AC focuses on making progress on the mission, and the BC focuses on generating safe backup plans.

2.1 Formal Definition of Black-Box Simplex

We formalize the behavior and requirements for the components of the Black-Box Simplex Architecture in order to prove properties about the system's behavior.

Plant Model. We consider discrete-time plant dynamics, modeled as a function

$$f(\underbrace{x_i}_{\text{state}}, \underbrace{u_i}_{\text{input}}, \underbrace{w_i}_{\text{disturbance}}) = \underbrace{x_{i+1}}_{\text{next state}} \tag{1}$$

where $i \in \mathbb{Z}^+$ is the time step, $x_i \in \mathcal{X}$ is the system state, $u_i \in \mathcal{U}$ is a control input command, and $w_i \in \mathcal{W}$ is an environmental disturbance. We sometimes also consider *deterministic* systems without disturbances.

Admissible States. The system is characterized by a set of operational constraints which include physical limits and safety properties. States that satisfy all the operational constraints are called *admissible states*.

Candidate Command Sequences. A single-input command is some $u \in \mathcal{U}$, and a k-length sequence of commands is written as $\overline{u} \in \mathcal{U}^k$. The length of a sequence can be written as $\overline{u}_{\text{len}} = k$, where we also can take the length of a single command, $u_{\text{len}} = 1$. We use Python-like notation for subsequences, where the first element in a sequence is $\overline{u}[0]$, and the rest of the sequence is $\overline{u}[1{:}]$.

Decision Module. The decision module in Black-Box Simplex stores a command sequence $\overline{s}$, which we sometimes call the decision module's state. The behavior of the DM is defined through two functions, dm_{update} and dm_{step}. The dm_{update} function attempts to modify the DM's stored command sequence:

$$dm_{\mathsf{update}}(\underbrace{x}_{\text{state}}, \underbrace{\overline{s}}_{\text{cur seq}}, \underbrace{\overline{t}}_{\text{proposed seq}}) = \underbrace{\overline{s'}}_{\text{new seq}} \tag{2}$$

where if $\overline{s'} = \overline{t}$ then we say that the proposed command sequence is *accepted*; otherwise $\overline{s'} = \overline{s}$ and we say that it is *rejected*. Correctness conditions on dm_{update} are given in Section 2.2. For the Multi-robot coordination problem in Section 4.1, the dm_{update} function accepts the safe command sequence $\overline{t}$, if for all states in the state trajectory obtained by executing $\overline{t}$ from the current state, all pairs of agents are a safe distance apart and the velocities point in non-conflicting directions, excluding the chance of future collisions.

Note that the DM will accept a safe command sequence from the LBC even if the previous command sequence from the LBC was rejected because it was unsafe. As in [12], we refer to this as *reverse switching*, since it switches control back to the AC.

The dm_{step} function produces the next command u to apply to the plant, as well as the next step's command sequence $\overline{s'}$ for the DM:

$$dm_{\mathsf{step}}(\underbrace{\overline{s}}_{\text{cur seq}}) = (\underbrace{u}_{\text{next input}}, \underbrace{\overline{s'}}_{\text{next seq}}) \tag{3}$$

where $u = \overline{s}[0]$ and $\overline{s'}$ is constructed from $\overline{s}$ by removing the first command (if the current sequence $\overline{s}$ has only one command then it is repeated):

$$\overline{s'} = \begin{cases} \overline{s} & \text{if } \overline{s}_{\text{len}} = 1 \\ \overline{s}[1:] & \text{otherwise} \end{cases}$$

Controllers. The AC and LBC are defined using functions of the system state. In particular, the AC is defined by a function $ac(x) = u$, where $u \in \mathcal{U}$ is a single command. BSA's *look-ahead baseline controller* is defined by $lbc(x) = \overline{u}$, where $\overline{u} \in \mathcal{U}^k$ is a k-length command sequence. The LBC outputs candidate command sequences that start with a given command, specifically, the command proposed by the AC. These can be defined with a function $lbc_{ac}(x) = \overline{u}$, with $\overline{u}[0] = ac(x)$. We note that the choice of the controller for the remaining sequence $\overline{u}[1+]$ is system-dependent. We generally drop the subscript on lbc, as it is clear from context.

Execution Semantics. At step i, given system state x_i and DM state $\overline{s_i}$, the next system state x_{i+1} and next DM state $\overline{s_{i+1}}$ are computed with the following sequence of steps: (1) $z_i = ac(x_i)$; (2) $\overline{t_i} = lbc(x_i)$, with $\overline{t_i}[0] = z_i$; (3) $\overline{s'_i} = dm_{\mathsf{update}}(x_i, \overline{s_i}, \overline{t_i})$; (4) $(u_i, \overline{s_{i+1}}) = dm_{\mathsf{step}}(\overline{s'_i})$; (5) $x_{i+1} = f(x_i, u_i, w_i)$, for some disturbance $w_i \in \mathcal{W}$.

We note that the functions lbc and dm_{update} are system-specific. For example, in the multi-robot coordination case study in Section 4.1, the agents start on the circumference of a circle and are required to safely reach their target locations on the opposite side of the circle. The lbc combines accelerations from MPC based AC and BC to produce a command sequence. The cost function of the AC, given in Eq. 6, is designed to attract the agents toward their goal locations. The cost function of the BC, given in Eq 7, prioritizes safety and pushes the agents to move out of the circle. Similarly, the dm_{update} checks for the safety of the command sequence. First, for all states in the state trajectory obtained by executing the LBC's command sequence, the Euclidean distance between every pair of distinct agents is greater than a threshold. Second, in the final state, for all pairs of distinct agents, the rays extending from their positions and in the directions of their velocities do not intersect.

2.2 Safety and Transparency Theorems

We define several relevant concepts and then state and prove safety and transparency theorems for Black-Box Simplex.

Definition 1 (Safe System Execution) A system execution is called *safe* if and only if the system state is admissible at every step.

Safety can be ensured by following a permanently safe command sequence from a given system state.

Definition 2 (Permanently Safe Command Sequence) Given state x_i, a k-length *permanently safe command sequence* $\overline{s_i} \in \mathcal{U}^k$ is one where the state x_j is admissible at every step $j \geq i$, where $(u_i, \overline{s_{i+1}}) = dm_{\mathsf{step}}(\overline{s_i})$, and $x_{i+1} = f(x_i, u_i, w_i)$, for every choice of disturbance $w_i \in \mathcal{W}$.

That is, the system state will remain admissible when applying each command in the sequence $\overline{s_i}$, and then repeatedly using the last command forever, according to the semantics of dm_{step}. More general definitions of permanently safe command sequences could be considered, such as repeating a suffix rather than just the last command. For simplicity we do not explore this here.

We define recoverable commands to be commands that result in states that have permanently safe command sequences.

Definition 3 (Recoverable Command) Given state x_i, a *recoverable command* u is one where there exists a permanently safe command sequence from x_{i+1}, where $x_{i+1} = f(x_i, u, w_i)$, for every choice of disturbance $w_i \in \mathcal{W}$.

Optimal decision modules are defined by requiring the dm_{update} function accept all sequences that can guarantee future safety, starting from a safe AC command.

Definition 4 (Optimal Decision Module) An *optimal decision module* has a dm_{update} function that accepts $\overline{t}$ at state x if and only if $\overline{t}$ is a permanently safe command sequence starting from x.

A correct DM is one which only accepts sequences that can guarantee future safety. A correct DM, by this definition, could reject every command sequence.

Definition 5 (Correct Decision Module) A *correct decision module* has a dm_{update} function that accepts $\overline{t}$ at state x only if $\overline{t}$ is a permanently safe command sequence starting from x.

The role of the BC is to try to keep the system safe. An optimal look-ahead BC can be defined as one that always produces a permanently safe command sequence when it exists. This is optimal in the sense that during system execution, it allows the DM to override the AC as infrequently as possible while still guaranteeing safety. This notion of optimality can be defined with respect to a specific advanced controller ac.

Definition 6 (Optimal Look-Ahead Baseline Controller) Given state x with $u = ac(x)$, if there exists a permanently safe command sequence $\overline{s}$ from x with $\overline{s}[0] = u$, then an *optimal look-ahead baseline controller* will always produce a permanently safe command sequence $\overline{t}$, with $\overline{t}[0] = u$.

Note that $\overline{t}$ may differ from $\overline{s}$, as there can be multiple permanently safe command sequences from the same state.

Theorem 1 (**Safety**) *Given initial state x_0 along with an initial permanently safe command sequence $\overline{s_0}$, if the decision module is correct, then the system's execution is safe regardless of the outputs of the advanced controller ac and look-ahead baseline controller lbc.*

Proof The command executed at each step comes from the state of the decision module $\overline{s_i}$, which maintains the invariant that $\overline{s_i}$ is always a permanently safe command sequence from the current system state $\overline{x_i}$. The dm_{update} function can only replace a permanently safe command sequence with another permanently safe command sequence. Since initially, $\overline{s_0}$ is permanently safe, then by induction on the step number, the decision module's command sequence at every step is permanently safe, and so the system's execution is safe. □

Although safety is important, achieving only safety is trivial, as a decision module can simply reject all new command sequences. A runtime assurance system must also have a transparency property, where the advanced controller retains control in sufficiently well-designed systems.

Theorem 2 (**Transparency**) *If (i) from every state x_i encountered, the output of the advanced controller $ac(x_i) = z_i$ is a recoverable command, (ii) the look-ahead baseline controller is optimal, and (iii) the decision module is optimal, then the input command used to actuate the system at every step is the advanced controller's command, z_i.*

Proof At the initial step, assume that a permanently safe command sequence exists. The proof proceeds by stepping through an arbitrary step i of the execution semantics defined in Section 2.1. Since the output of the advanced controller $ac(x_i) = z_i$ is assumed to be recoverable, there exists a permanently safe command sequence from x_i that starts with z_i. By the definition of an optimal look-ahead baseline controller, since there exists a permanently safe command sequence, the output $lbc(x_i) = \overline{t}$ must also be a permanently safe command sequence, with $\overline{t}[0] = z_i$ as required by the definition of a look-ahead baseline controller. In step (3) of the execution semantics, $dm_{\mathsf{update}}(x_i, \overline{s_i}, \overline{t_i}) = \overline{s_i'}$. Since $\overline{t}$ is a permanently safe command sequence and the decision module is optimal, the command sequence will be accepted by the decision module, and so $\overline{s_i'} = \overline{t}$. Step (4) of the execution semantics produces u_i, which is the first command in the sequence $\overline{t}$. As shown before, this command is equal to z_i, which is used in step (5) of the execution semantics to actuate the system. This reasoning applies at every step, and so the advanced controller's command is always used. □

Discussion. There are several practical considerations with the described approach. For example, the black-box controllers may not only generate unsafe commands, but a controller implementation may fail to generate a command at all, for example, entering an infinite loop. To account for such behaviors, a runtime cap can be used with a default command sequence assumed if the DM

receives no input. For increased protection, the black-box controllers can be isolated on dedicated hardware [13] so that they do not, for example, crash a shared operating system. Also, the DM's analysis of the command sequence is nontrivial and could involve a runtime reachability computation. If this may take too long, we again could use a runtime cap. This means that the practicality of the architecture depends on the efficiency of runtime reachability methods, an active area of research orthogonal to this work.

Another consideration is the feasibility of coming up with permanently safe command sequences. For systems where landing or coming to a stop is considered safe, remaining there forever will be permanently safe. Other approaches, which we use the case studies in the next section, rely on geometric arguments to show permanent safety. Methods from control theory could also be used for this, such as computing forward invariant sets [14] or using a locally stable controller. For example, using the indirect method of Lyapnuov, a closed-loop system's equilibrium point x^* can be proven to be stable using linearization, along with conservative bounds on its basin of attraction [15]. The BC would then strive to get the system into the basin of attraction of x^*, and then use the locally stable controller to ensure indefinite future safety. Directly using the locally stable controller as the BC, however, would be overly conservative, as it would not allow the system to leave the (potentially small) basin of attraction.

3 Commands Blending for Multi-Agent Systems

The BSA theory presented in the previous section applies to any general autonomous cyber-physical systems. In case of multi-agent systems, we can sometimes further improve the DM to allow the system to use the advanced controller commands more frequently. In the theory presented above, the DM would either make all the agents use control commands from the advanced controller or all agents use the previous permanently safe control command sequence. The drawback of this strategy is that if only a small subset of the agents are at risk of a safety violation, all the agents will be forced to revert their command to the LBC. To overcome this drawback, we introduce *command blending*, in which some agents use the command from the AC while others use the LBC. In this section, we adapt the previous formalism to provide conditions that ensure command blending does not violate system safety.

Command blending is applicable for multi-agent systems. We assume there are n agents and extend the notation to use a superscript to denote a specific agent's state, command and disturbance. The main change for command blending is in the decision module's dm_{update} function.

Command Blending Decision Module. The decision module dm_{update} function attempts to modify the DM's stored command sequence, as defined before in Equation 2. Recall that decision module's state (the stored command sequence) is denoted by $\overline{s}$, the new proposed command sequence is $\overline{t}$ and the command sequence after the safety check is $\overline{s'}$, which previously was equal to either $\overline{s}$ or $\overline{t}$. With command blending, we construct a new command sequence $\overline{r}$ that can alternately be accepted by the decision module.

The proposed command sequence $\overline{t}$ in the multi-agent case consists of commands $\overline{t}[i]^j$ for agent j at time step i. As before, if the state is admissible at every step when executing the proposed command sequence $\overline{t}$, then $\overline{s'} = \overline{t}$ and we say that the proposed command sequence is *accepted*. Otherwise, rather than immediately falling back to the stored command sequence $\overline{s'} = \overline{s}$ as before, the decision module attempts to construct a mixed command sequence. A mixed command sequence between the stored command sequence $\overline{s}$ and the proposed command sequence $\overline{t}$ is constructed by starting with $\overline{t}$ and iteratively replacing the command sequence for a chosen agent j (for all time steps) with the commands from the safe backup sequence $\overline{s}$. If agent j is chosen in the first iteration, the new command sequence $\overline{r}$ has $\overline{r}[i]^j = \overline{s}[i]^j$, and for all other agents $k \neq j$, $\overline{r}[i]^k = \overline{t}[i]^k$. The modified command sequence is then checked again using dm_{update}, and if the command sequence is not accepted, a second agent is selected to roll back to $\overline{s}$. This process repeats until the blended command sequence is accepted by the DM.

The agent which is chosen should be different at each iteration, to ensure that eventually all agents get rolled back if needed. This ensures the approach eventually terminates, as in the worst case, after n iterations all the agents will be rolled

back and $\overline{r}$ will be same as $\overline{s}$, which was shown as admissible at the previous step and will therefore be accepted by the DM.

Discussion. The safety of this approach follows directly from the safety theorem in Section 2.2 for the general BSA. The choice of agent to rollback can be done arbitrarily without affecting safety, but in many cases heuristics associated with the safety check performed by dm_{update} can be used to select the agent, in order to reduce the number of rolled-back agents. For example, in our case study we use collision safety properties. When two agents are detected to be colliding in the future, we select one of them to rollback, in order to resolve the conflict. However, at the next iteration of the process to create the blended command sequence, dm_{update} may detect a new set of conflicting agents, which will lead to another agent being selected.

The trade off of the proposed process is that dm_{update} may need to run multiple times, which increases the computational cost. Note that, as before, if the computation budget is exceeded it is always safe to simply use the full backup command sequence $\overline{s}$.

Blended commands may also not be desirable for all multi-agent systems, for example if there are synchronization concerns where we want either all agents to complete a task or none of them. However, as we will show in the evaluation, for motion properties, using blended commands can help multi-agent systems complete tasks that cannot be completed using the general BSA.

4 Case Studies

In this section, we apply the approach to two case studies: a multi-robot coordination system, and a mid-air collision avoidance system for groups of F-16 aircraft.

4.1 Multi-Robot Coordination

We consider a multi-agent system (MAS), indexed by $\mathcal{M} = \{1, ..., n\}$, of planar robots modeled with discrete-time dynamics of the form:

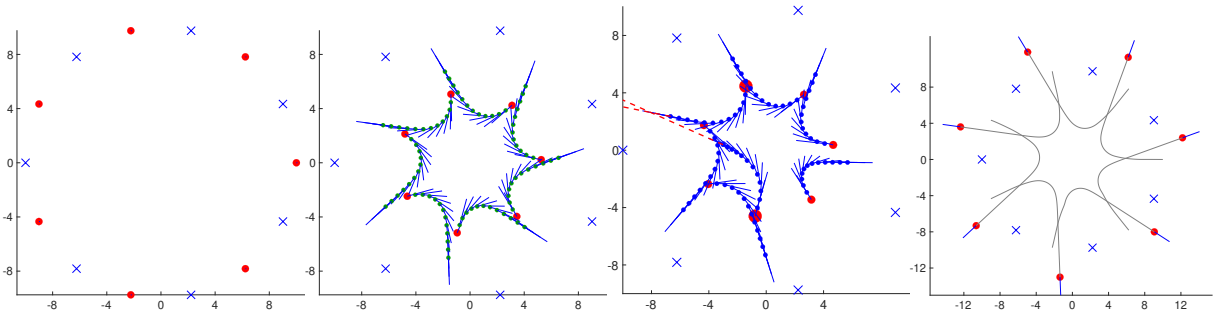
$$\begin{aligned} p_i(k+1) &= p_i(k) + dt \cdot v_i(k),\ |v_i(k)| < v_{max} \\ v_i(k+1) &= v_i(k) + dt \cdot a_i(k),\ |a_i(k)| < a_{max} \end{aligned} \tag{4}$$

where p_i, v_i, $a_i \in \mathbb{R}^2$ are the position, velocity and acceleration of agent i, respectively, at time step k, and $dt \in \mathbb{R}^+$ is the time step. The magnitudes of velocities and accelerations are bounded by v_{max} and a_{max}, respectively. The acceleration a_i is the control input for agent i. The combined state of all agents is denoted as $x = [p_1^T, v_1^T, ..., p_n^T, v_n^T]^T$, and their accelerations are $a = [a_1^T, ..., a_n^T]^T$.

In the initial configuration, the agents are equally spaced on the boundary of a circle and are at rest. Agent i's goal is to reach a target location r_i, located on the opposite side of the circle. The initial configuration of the MAS is shown in Figure 3(a), where the agents and their target locations are represented as red dots and blue crosses, respectively. The safety property is absence of inter-agent collisions. A pair of agents is considered to collide if the Euclidean distance between them is less than a non-negative threshold d_{min}. Thus, the safety property is that $\|p_i - p_j\| > d_{min}$ for all pairs of agents $i, j \in \mathcal{M}$ with $i \neq j$.

Both the AC and the BC are designed using centralized Model Predictive Control (MPC), which produces command sequences as part of the solution of a nonlinear optimization problem. For collision avoidance, we use a potential field formulation [16] in both the AC and BC. While the AC tries to reach the target positions on the opposite side of the circle, the BC has a simpler goal of having each agent leave the circle. Note that numerical methods for global nonlinear optimization, such as MATLAB's `fmincon` used in our implementation, do not provide a guaranteed optimal solution. To create unsafe variants of the controllers, we simply limit the number of iterations used for optimization.

The AC only outputs the first command of the command sequence, whereas the BC produces the full command sequence. Both the AC and the BC are high-level controllers that produce accelerations. In our simulations, we do not model the low-level controller; the plant dynamics work directly with the accelerations. When implementing our approach on physical robots, a trusted low-level controller will map the desired acceleration commands to actuator inputs. A centralized MPC controller produces a command sequence $\overline{s}$ of length T, where T is the prediction horizon, and



(a) Initial configuration, $k = 1$ (b) $k = 10$ (c) BC fails, $k = 11$ (d) Final configuration, $k = 32$

Fig. 3: Simulation of the MAS with 7 robots. The DM performs system recovery after the BC produces an unsafe command sequence. The BC's proposed path is shown in part (c) at $k = 11$, where the two dotted red lines intersect, indicating the future paths of the agents cross. We represent current positions as red dots, future positions corresponding to the safe/unsafe command sequences as green/blue dots, velocities as blue lines, and agent trajectories as grey curves.

each command $\overline{s}[i]$ contains the accelerations for all agents to use at step i.

The centralized MPC controller solves the following optimization problem at each time step k:

$$\underset{a(k|k),\ldots,a(k+T-1|k)}{\arg\min} \sum_{t=0}^{T-1} J(k+t \mid k) + \lambda \cdot \sum_{t=0}^{T-1} \|a(k+t \mid k)\|^2 \quad (5)$$

where $a(k+t \mid k)$ and $J(k+t \mid k)$ are the predictions made at time step k for the values at time step $k+t$ of the accelerations and the centralized (global) cost function J, respectively. The first term is the sum of the centralized cost function, evaluated for T time steps, starting at time step k. It encodes the control objective. The second term, scaled by a weight $\lambda > 0$, penalizes large control inputs.

Advanced controller. The centralized cost function J_{ac} for the AC contains two terms: (1) a *separation* term based on the inverse of the squared distance between each pair of agents (potential field term for collision avoidance); and (2) a *target seeking* term based on the distance between the agent and its target location.

$$J_{ac} = \omega_s \sum_{i>j} \frac{1}{\|p_i - p_j\|^2} + \omega_t \sum_i \|p_i - r_i\|^2 \quad (6)$$

where $\omega_s, \omega_t \in \mathbb{R}$ are the weights of the separation term and target seeking terms. The separation term promotes inter-agent spacing but does not guarantee collision avoidance. The AC generates a command sequence by solving the optimization problem in Eq. 5, with J replaced by J_{ac}. The first command in that sequence is the AC's command; it is passed to the LBC.

Baseline controller. The centralized cost function J_{bc} for the BC contains two terms. As in Eq. 6, the first term is the separation term (collision avoidance based on potential fields). The second term is a *divergence* term which forces the agents to move out of the circle by aligning their velocities with rays radially pointing out of the center of the circle.

$$J_{bc} = \omega_s \sum_{i>j} \frac{1}{\|p_i - p_j\|^2} + \omega_d \sum_i \left(1 - \frac{(p_i - c) \cdot v_i}{\|p_i - c\| \, \|v_i\|}\right) \quad (7)$$

where $\omega_s, \omega_d \in \mathbb{R}$ are the weights of the separation term and the divergence term, and c is the center of the circle containing the initial configuration of the robots and their target locations. The control law for the BC is Eq. 5, with J replaced by J_{bc}. A zero acceleration is appended to the end of the BC's command sequence to help establish collision freedom for all future time steps.

Decision module. The LBC combines accelerations from the AC and the BC, producing the command sequence $\overline{t} = [ac(x), bc(x'), \vec{0}]$, where x' is the next state after executing $ac(x)$ in state x.

The function $dm_{\mathsf{update}}(x, \overline{s}, \overline{t})$ accepts the proposed command sequence $\overline{t}$ if and only if $\overline{t}$ is a permanently safe command sequence. For this system, a command sequence $\overline{t}$ is considered permanently safe in a state x if it satisfies the following two conditions. First, for all states in the state trajectory obtained by executing $\overline{t}$ from x, the Euclidean distance between every pair of distinct agents is at least d_{min}. Second, in the final state, for all pairs of distinct agents, the rays extending from their positions and in the directions of their velocities do not intersect. Any pair of agents that satisfies the second condition will not collide in the future, since the last command in the sequence $\overline{t}$ has zero acceleration. The initial permanently safe command sequence is a zero acceleration for all agents, as the agents start at rest.

MPC Parameters. In our case study, we use the following MPC parameters: $dt = 0.3$ *sec*, $d_{min} = 1.7$, $a_{max} = 1.5$, and $v_{max} = 2$. The length of the prediction horizon for MPC is $T_{ac} = T_{bc} = 10$.

Successful Recovery After Failure. We first consider seven robotic agents initialized on a circle centered at the origin, with a radius of 10. The initial state of the system is shown in Figure 3(a). At $k = 11$, the BC produces an unsafe command sequence. The state trajectory corresponding to the unsafe sequence is shown in blue. As shown in Figure 3(c), the final paths of the two agents corresponding to the larger red dots cross after simulating the current state forward with the unsafe sequence. Hence, at $k = 11$, the DM rejects the proposed command sequence and shifts control to the previous safe command sequence, which safely recovers the system. Here, we purposefully did not return control to the AC to demonstrate how the stored command sequence keeps the agents safe [1].

Reverse Switching Scenario. We stress-tested the multi-robot system by initializing 12 agents on a circle of radius 10. The path of the agents is shown in Figure 4. There are 10 instances where the DM rejects the AC's proposed command sequence and instead uses the stored command sequence. Nonetheless, all agents reach their target locations without colliding, maintaining a minimum separation of 1.724 between any pair of agents[2].

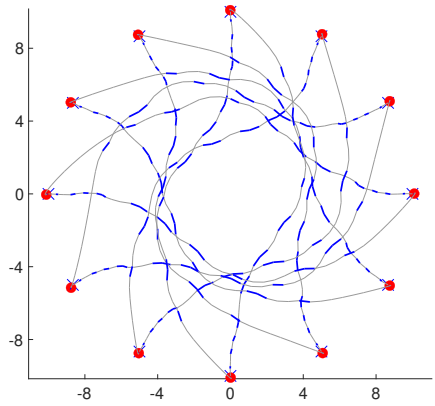


Fig. 4: Stress test of robotic MAS with 12 robots reaching their targets. Trajectory segments where stored command sequences are used are shown in blue.

Handling Uncertainty. We next investigate the DM's runtime overhead when there is uncertainty in the robot's state or the dynamics. The former case arises when the sensors used to determine the positions and velocities are subject to sensor noise. The latter case could be used to account for modeling errors, through disturbances on the positions and velocities at each step.

We continue to use the same MPC strategy as before; thus, the controllers ignore the uncertainty when generating proposed command sequences. Only the logic used by the DM to accept or reject command sequences is modified to account for uncertainty. We examine the scenario shown before in Figure 3(b). To account for the uncertainty, we perform an online reachability computation. To do this, we use efficient methods for reachability for linear systems based on zonotopes [17], which we implement in Python. Briefly, a zonotope is a set of states represented as an affine transformation of a unit box. The unit box is associated with a number of *generator vectors*, where each generator vector corresponds to one dimension of the box. The computational efficiency of propagating sets over time using zonotopes relates to the number of generators. Each agent has four

[1] A video of the simulation is available at https://youtu.be/bcVJBkGgnxA.

[2] A video of the simulation is available at https://youtu.be/qmk31jS6B2Y.

state variables, two for position and two for velocity. The composed system with seven agents has 28 state variables.

In the situation shown in Figure 5(a), the current state is assumed to have uncertainty independently in both position and velocity with an L^2 norm of 0.1. We use a 16-sided polygon to bound this uncertainty. In the plot, the deterministic simulation is given, along with black polygons for each agent that show the states that might be reachable at each step due to the sensor uncertainty. The uncertainty in the velocity causes the set to expand over time, since the open-loop command sequence does not attempt to compensate for the uncertainty. The zonotope representation of the composed system needs 112 generator vectors to represent the initial states, which remains constant at every time step.

In the situation shown in Figure 5(b), the initial state has very little error, but the dynamics is modified to have disturbances at each step. For each component of each agent's position and velocity, we allow an external disturbance value to be added in the range $[-0.02, 0.02]$. Since each agent has four independent disturbances, the zonotope representation of the composition will have 28 new generators added at each step. After 12 steps, the final zonotope will have a total of 364 generators.

Runtime. To measure runtime, we used a standard laptop with a 2.70 GHz Intel Xeon E-2176M CPU and 32 GB RAM. The method is fast. For the case of sensor uncertainty, computing the box bounds of the reachable set at all the steps takes about 1.5 milliseconds. With uncertainty, even though the number of generators grows over time, it is not large enough to significantly affect the runtime. The computation with disturbances requires about 2 milliseconds to complete. We believe such execution times are sufficiently fast for use in the decision module.

4.2 Multi-Robot Coordination with Command Blending

To evaluate the command blending strategy, we consider scenarios with large numbers of agents. In such difficult scenarios, the AC is not able to easily generate recoverable commands for all robots. This creates an opportunity for command blending to improve performance. The code is available online [3].

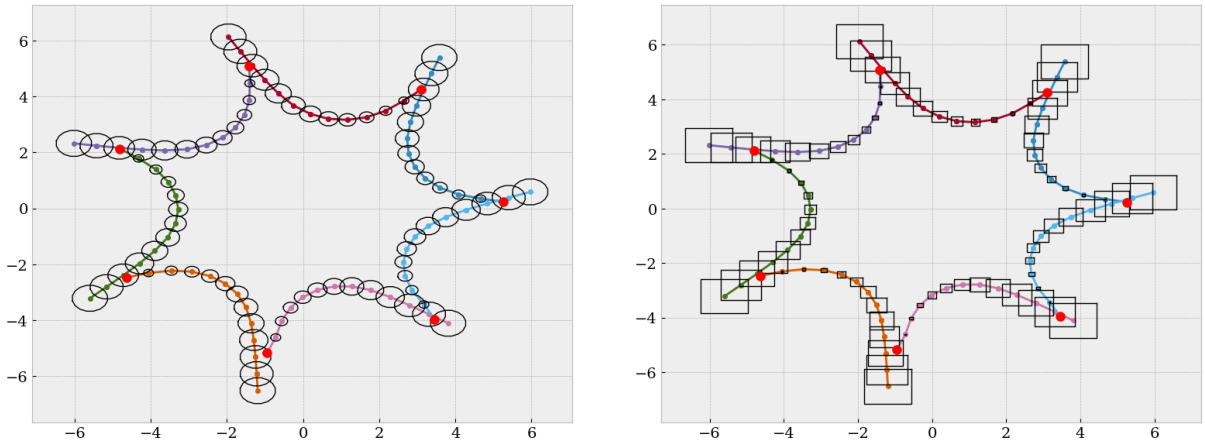
Table 1 shows the system performance for different numbers of agents. Our experiments show that when the number of agents is small both strategies perform well with high success rate and low distance. On the other hand, when the number of agents is large, the success rate decreases and the average distance increases. However, the command blending strategy alleviates this performance degradation. For all scenarios the mean of distances is lower and the success rate is much higher when applying the command blending strategy. In fact for more than 12 agents the success rate is zero with the original switching strategy. Without command blending, the final distances of the robots from the target location are high as the safety controller pushes the agents out of the circle.

Figure 6 shows a simulation of the MAS with 13 robots that illustrates why command blending performs better. The gray or red line segments represents the trajectory of an agent. Without command blending there are more rollbacks and hence more red segments, while with command blending agents use a mixture of proposed and safe backup command sequences and as a result there are combinations of gray and red segments.

In Fig. 6(a), the DM performed a complete switch for all agents to the backup sequence and redirected them to move out of the circle to avoid collisions. At a certain point, reverse switching can switch back to the advanced controller's command, for a time, but then after a few steps the system again switches to the backup command sequence and this process repeats over and over, similar to a livelock. We noticed this often happens in more difficult scenarios when there are many agents.

Fig. 6(b) shows the system with the command blending strategy where the DM is able to prevent collisions while some agents continue to make progress toward their destinations.

[3] https://github.com/sanazsheikhi/BlackBox-Simplex-Extension



(a) Reachable States with Sensor Error (b) Reachable States with Disturbances

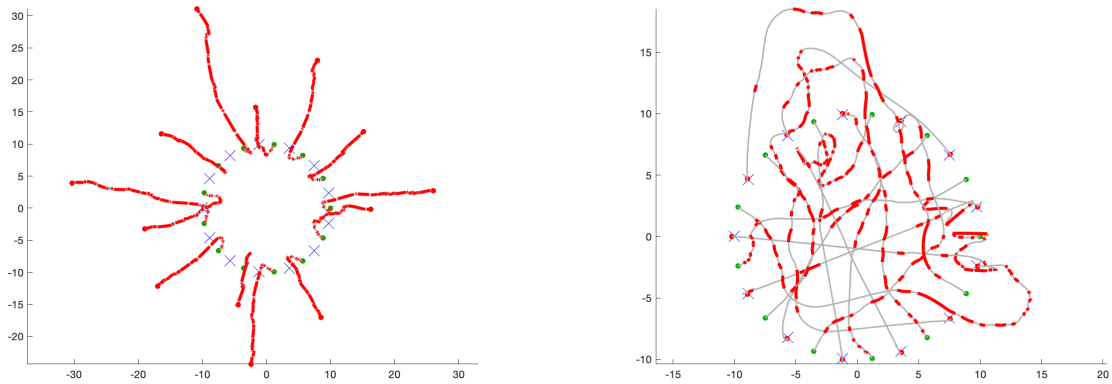
Fig. 5: Zonotope reachability computes future states with uncertainty.

Table 1: Command blending performance evaluation. The *Distance* columns represent the average final distance of all agents from their destinations. The *Success* columns indicate the percentage of agents who either reached to their destinations or to the vicinity of their destinations (e.g. less than 1.25% of the initial distance) by the end of the simulation. Each simulation is 200 time steps. Each experiment was repeated 20 times to average over the the effects of the randomness in MPC; the means and standard deviations over the 20 executions are reported.

	Without Command Blending		With Command Blending	
Agents	Distance	Success(%)	Distance	Success(%)
11	19.77 ± 20.83	41.67 ± 51.49	0.10 ± 0.01	100
12	22.86 ± 8.21	8.33 ± 28.87	0.12 ± 0.07	99.17 ± 3.73
13	31.78 ± 3.66	00.00	1.79 ± 6.12	93.85 ± 16.10
14	29.00 ± 4.69	00.00	2.58 ± 3.68	87.50 ± 14.82
15	25.15 ± 3.04	00.00	4.12 ± 5.49	80.00 ± 22.06
16	24.29 ± 2.47	00.00	9.96 ± 8.68	59.06 ± 19.39

4.3 Multi-Aircraft Collision Avoidance

Our second evaluation system guarantees collision avoidance for groups of aircraft. We use a full six-degrees-of-freedom F-16 simulation model [18], based on dynamics taken from an Aerospace Engineering textbook [19]. Each aircraft is modeled with 16 state variables, including positional states, positional velocities, rotational states, rotational velocities, an engine thrust lag term, and integrator states for the low-level controllers. These controllers actuate the system using the typical aircraft control surfaces—the ailerons, elevators, and rudder—as well as by setting the engine thrust. The system evolves continuously with piece-wise nonlinear differential equations, where the function that computes the derivative given



(a) Without command blending strategy

(b) With command blending strategy

Fig. 6: Simulation of the MAS with 13 robots with and without command blending. Green points are start position. Red points are final positions at the end of 200 time steps. Gray and red segments indicate when advanced commands and backup commands, respectively, are used.

the state is provided as Python code. In order to match the discrete-time plant model in Definition 1, we periodically select a control strategy with a frequency of once every two seconds. The model further includes high-level autopilot logic for waypoint following, which we reuse in the advanced controller.

For the collision-avoidance baseline controller, our controller is based on the ACAS Xu system designed for collision avoidance in unmanned aircraft [20]. While the original system was designed using a partially observable Markov decision process (POMDP), the resultant controller was encoded in a large look-up table that used hundreds of gigabytes of storage [21]. To make the system more practical, one early approach considered a downsampling process followed by a lossy compression using neural networks [21, 22]. We use these downsampled neural networks as the BC and refer to this as the original system.

The system issues horizontal turn advisories based on the relative positions of two aircraft, an *ownship* and an *intruder*. The system is similar to Simplex, where the output can be either *clear-of-conflict*, where any command is allowed, or an override command that is one of *weak-left*, *weak-right*, *strong-left* or *strong-right*. We adapt this system to the multi-aircraft case by having each aircraft run an instance of the system against every other aircraft, using the closest turn advisory as the output.

To create command sequences, the BC repeatedly advances the plant model and re-runs the collision avoidance system in a closed-loop fashion until the generated command sequence is permanently safe. To check whether a generated command sequence is permanently safe, the DM checks that (i) each aircraft's state stays within the model limits (e.g., no aircraft enters a stall), (ii) all aircraft obey the safety distance constraint at all times, and (iii) the execution ends in a state where the roll angle of each aircraft has been small (less than 15 degrees) and the distances between all pairs of aircraft has been increasing consecutively for several seconds. If all aircraft continue to fly straight and level from such a configuration, their distance would increase and no collisions would occur in the future.

As with the multi-robot scenario, we examine cases where the initial aircraft state x_0 has all aircraft starting evenly-spaced, facing towards the center of a circle with a given initial diameter. Each aircraft has an initial velocity of 807 ft/sec and an initial altitude of 1000 ft, both of which are maintained throughout the maneuver by the lower-level controllers. The AC commands each aircraft to fly towards a waypoint past the opposite side of the circle, which would cause a collision at the center. The safety property requires maintaining horizontal separation. The *near mid-air collision cylinder* (NMAC) uses a safe horizontal separation of 500 ft [23], although we will vary this

in our evaluation. For the initial permanently safe command sequence s_0, we have each aircraft fly in clockwise circles forever, which avoids collisions.

In addition to the AC being unsafe, the baseline controller should not be fully trusted for many reasons. The original POMDP formulation was not proven formally correct, not to mention the downsampling and lossy neural network compression. While some research has examined proving open-loop properties for the neural network compression [22, 24, 25], these do not imply *closed-loop* collision avoidance. Further, we use a multi-aircraft adaptation of the system, which could also lead to problems. Although aspirationally, the system should handle up to 30 intruders [21], in practice most analysis has been performed on two aircraft scenarios. Finally, the intended physical system response to the collision-avoidance commands is that *weak-left* and *weak-right* should cause turning at 1.5 degrees per second, whereas *strong-left* and *strong-right* turn at 3.0 degrees per second [21]. However, turning an aircraft in the F-16 model (as well as in the real world) is not an instantaneous process, and requires first performing a roll maneuver before the heading angle begins to change. For these reasons, the BC in this scenario is also an unverified component, and we will show scenarios where it misbehaves. Nonetheless, we will compose the incorrect AC with the incorrect BC to create a safe collision-avoidance system by using BSA.

We now elaborate on four scenarios: (i) a three aircraft case, which shows the safety of the system despite unsafe outputs, (ii) a four aircraft case, which shows the increased transparency of BSA, (iii) a seven aircraft case, which shows the safety condition can be easily customized and (iv) a 15 aircraft case, which shows safe navigation of a complex scenario.

In all the plots in this section, we show snapshots at the time when the distance between the two closest aircraft is smallest. The two red aircraft are the closest pair, and their distance is printed in the bottom right of each figure. The solid line shows the historic path of each aircraft, and the dotted line is the future trajectory.

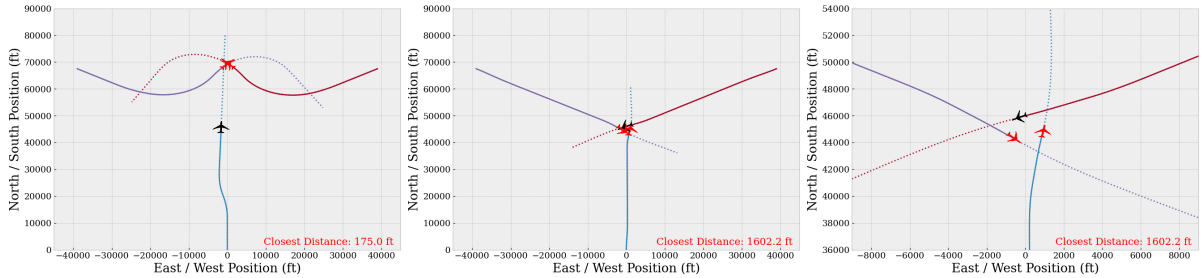
Three Aircraft Scenario. The original collision avoidance system was designed with two aircraft in mind, an ownship and an intruder. We adapted it to the multi-aircraft case, but this mismatch between the system design assumptions and usage scenario can lead to problems. In Figure 7, we show such a scenario, where the initial circle diameter is 90,000 ft. In Figure 7(a), the minimum distance between the top two aircraft is 175 ft, violating the near mid-air collision safety distance. The other two subplots show the system using BSA with a safety distance of 1500 ft; the minimum separation is 1602 ft, which satisfies the constraint.

Four Aircraft Scenario. Figure 9 shows a four-aircraft scenario using an initial circle diameter of 70,000 ft. In this case, both designs have safe executions. Using the original system leads to a minimum separation of 5342 ft, whereas the minimum separation with Black-Box Simplex is 1600 ft, much closer to the 1500 ft safety-distance constraint used in the DM. Although both systems are safe, from the plots it is clear that the Black-Box Simplex version is more transparent, in the sense that it produces smaller modifications to the direct-line trajectories commanded by the AC.

Seven Aircraft Scenario. We investigated a seven aircraft scenario with an initial circle diameter of 70,000 ft. Here, the original system violates the horizontal separation constraint, and the minimum separation distance is 277 ft. We run Black-Box Simplex on this system using three different safety distances, 1500 ft, 1000 ft, and 500 ft. All avoid collisions, and as the safety distance is decreased, the observed minimum distance also decreases. This shows that Black-Box Simplex can be easily customized to a change in the safety requirement. Doing this for the original system would require significant effort in recomputing the POMDPs and retraining the neural networks to perform a compression of the action tables. Plots of the seven aircraft trajectories are provided in Figure 8. Video of the 1000 ft case is available online [4].

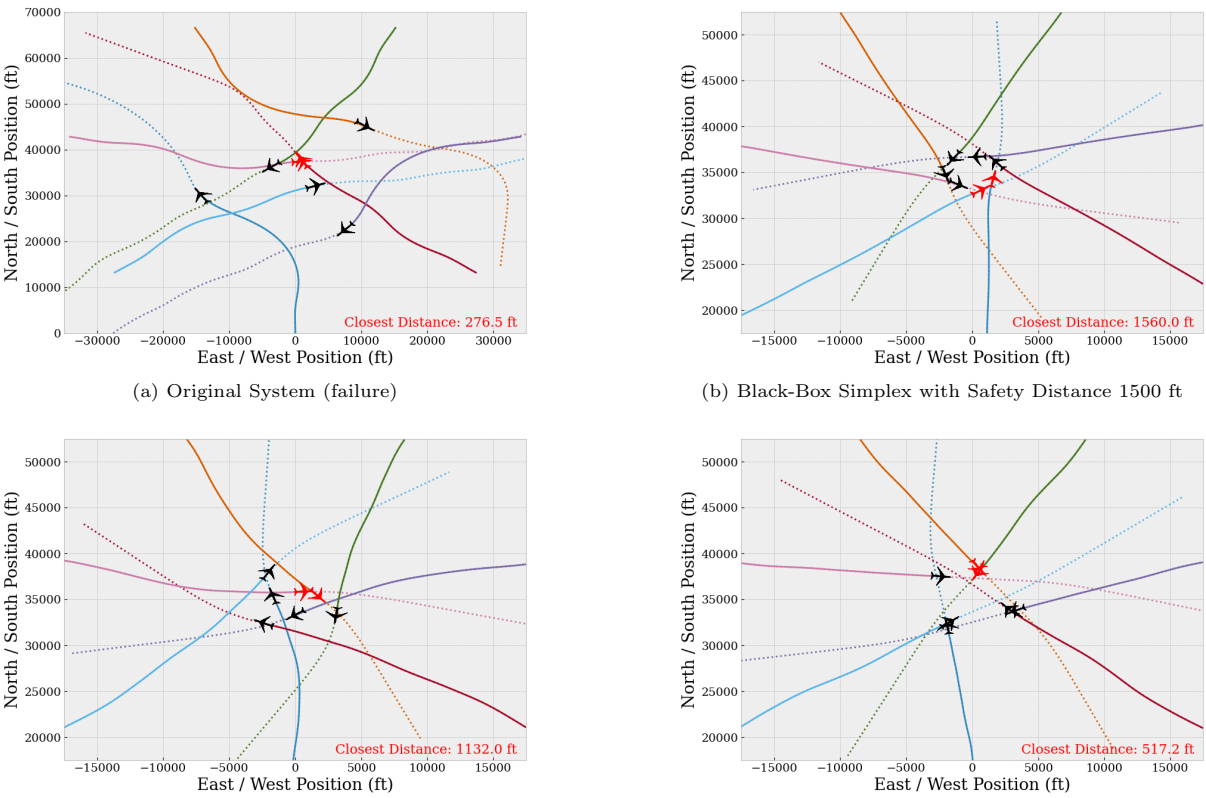
Fifteen Aircraft Scenario. Finally, we demonstrate the system's ability to safely navigate complex scenarios. For this, we use a 15 aircraft scenario, with an initial circle diameter of 90,000 ft. With 15 aircraft, the composed system has 240 real-valued state variables, each of which evolves according to piece-wise nonlinear differential equations. The plot for this system was shown in the introduction in Figure 2. While the original

[4] https://youtu.be/6ZXjk8k-Xqs



(a) Original System (b) Black-Box Simplex (c) Black-Box Simplex (Zoomed In)

Fig. 7: Black-Box Simplex is safe. In the three-aircraft case, the original system fails, whereas BSA maintains the 1500 ft separation.



(a) Original System (failure)

(b) Black-Box Simplex with Safety Distance 1500 ft

(c) Black-Box Simplex with Safety Distance 1000 ft

(d) Black-Box Simplex with Safety Distance 500 ft

Fig. 8: Black-Box Simplex is easily customizable. In the seven aircraft case, adjusting the safety distance in the decision module results in different system behaviors. In each case, the advanced controller command is overridden only enough to guarantee the corresponding safety constraint.

system is unsafe, Black-Box Simplex has a minimum separation of 1500.5 ft, just above the 1500 ft safety constraint used in the DM. Another surprising observation is that in some of the cases, such as this 15-aircraft case and the seven-aircraft case, the aircraft perform something similar to a roundabout maneuver. This is an emergent behavior,

not something explicitly hardcoded or anticipated. A video of this case is also available online[5].

Runtime. The existing implementation uses numerical integration for the dynamics with an adaptive-step explicit Runge-Kutta scheme of order 5(4) from Python's `scipy` package. On our laptop platform with default accuracy parameters, this runs at about 55 times faster than real-time per aircraft.

5 Related Work

Reachability-based verification methods for black-box systems for waypoint following with uncertainty have been recently investigated in the ReachFlow framework [9]. ReachFlow builds upon the Flow* reachability tool [26], which is unlikely to scale to systems like the 240-variable 15-aircraft scenario.

A framework for safe trajectory planning using MILP for piecewise-linear vehicle models is presented in [27, 28]. The method relies on the ability of an MPC controller to produce command sequences where the terminal state in the prediction horizon is constrained to lie within a safe invariant set. This provides a safe back-up command sequence for the next step in case the system fails to find a safe sequence. The scope of this work is limited to MPC, and it is not clear how to extend it to other types of controllers. Moreover, the conditions for switching back from the stored return trajectory are not formalized.

In the Contingency Model Predictive Control framework [29], an MPC controller maintains a contingency plan in addition to the nominal or desired plan to ensure safety during an identified potential emergency. Like BSA, the initial command is common to both plans. In this framework, both plans must be generated using their custom version of MPC, whereas Black-Box Simplex works with independent baseline and advanced controllers of any design.

Similar frameworks have been considered for autonomous vehicles, using fail-safe backup plans and reachability analysis [30]. In this case, the target was planning for autonomous vehicles where most likely trajectories are used for other vehicles but safety can still be provided if emergency maneuvers are performed instead. Other ideas such as Safety Net Control [31] extend the approach to use backreachability and underapproximations of nonlinear reachable sets while taking computation time into account.

Designing safe switching logic for a given baseline controller is related to the concept of computing viability kernels [32] (closed controlled-invariant subsets) in control theory. This often requires set operations which can be inefficient in high-dimensional spaces with nonlinear dynamics, although there has been some progress on this [33, 34].

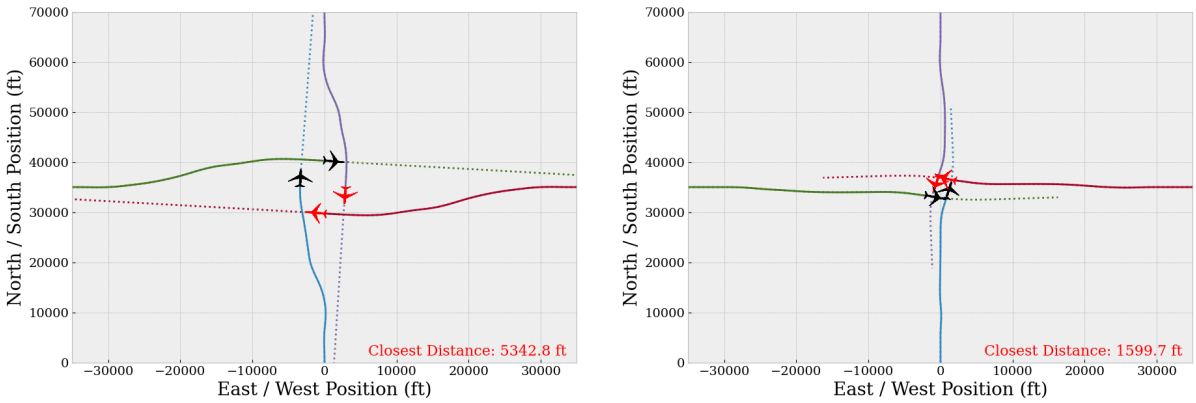
Simplex designs have also been considered that use a combination of offline analysis with online reachability [10]. Again, though, reachability computation is currently intractable for large nonlinear systems, and requires symbolic differential equations. Other work has used Simplex to provide safety guarantees for neural network controllers with online retraining [35]. In these approaches, the baseline controller must be verified ahead of time.

Online simulation-based methods have also been investigated to secure power grids from insider attacks [36]. As with this work, fast online simulation is critical, although the goal there is system security not safe high-level control design.

The design of the MPC controllers for our multi-robot case study is similar to Control Barrier Function (CBF) methods [37, 38] and Implicit Active Set Invariance Filtering [39]. There, a runtime assurance system was used to provide minimally perturbed advanced controller commands, computed using a constrained-optimization problem. However, the optimization problem might become infeasible or global nonlinear optimization could perform poorly at one of the steps at runtime, causing this method to be unsafe. With Black-Box Simplex, failure of the baseline controller does not compromise safety. Also, in general CBFs are not easy to construct. More recently sum-of-squares programming [40] and data-driven techniques [41] have been used for the synthesis of CBFs. Other methods of synthesizing CBFs are surveyed in [42].

Distributed simplex architecture [43] is runtime assurance technique for multi-agent systems where each agent has two controllers and the ability to independently shift the control between them. The AC is mission critical and the BC is safety critical. The design of the BC and the

[5] https://youtu.be/Bhn0uqKCj7Q



(a) Original System (b) Black-Box Simplex

Fig. 9: Black-Box Simplex is more transparent. For the four aircraft case, the original system is significantly more intrusive than Black-Box Simplex, which overrides commands just enough to guarantee the 1500 ft separation requirement.

decision module's logic relies on the existence of a CBF. In contrast, our method is generally applicable and does not rely on CBFs.

A decentralized shield-based technique for runtime verification of multi-agent systems is presented in [44]. In this approach, every agent has a shield consisting of two components: a pathfinder that corrects the agent's behavior and an ordering approach that dynamically modifies the agent's priority. In contrast to BSA, the work assumes that the agents know the intended behavior of other agents.

RV4JaCa [45] is proposed to integrate runtime verification in the multi-agent System domain. It brings a layer of security to the multi-agent system, capable of controlling events during the execution of the system without needing a specific implementation in the behavior of each agent to recognize the events by monitoring Agent Interaction Protocols.

Runtime assurance is close to the runtime enforcement. A fundamental work [46], proposes a security automaton that interposes itself between the program and the host machine and examines the sequence of security-relevant program actions one at a time. If the automaton recognizes an action that will violate its policy, it terminates the program. In an extension of this work, an automata with more powerful transformational abilities, including the ability to insert a sequence of actions into the event stream and to suppress actions in the event stream without terminating the program [47] is proposed. In another extension [48], a generic notion of enforcement monitors based on a memory device, finite sets of control states, and enforcement operations are introduced. Also, a systematic technique to produce a monitor from the automaton recognizing a given safety, guarantee, obligation, or response property is proposed [49]. In all these works, the monitor is usually modeled as an automaton (finite state machine), which dictates its behavior according to the input action and current state. Care must be taken to ensure that this finite state machine correctly enforces the policy and complies with the limitations imposed on the monitor's capabilities.

A technique based on a new model of enforcement monitors, allowing the comparison between multiple alternative corrective enforcement actions and the selection of the optimal one concerning an objective user-defined gradation separate from the security policy is proposed [50].

6 Conclusions

We have presented the Black-Box Simplex Architecture, a methodology for constructing safe CPS from unverified black-box high-level controllers. Unlike the classical Simplex design, the baseline controller does not need to be statically verified and can even be incorrect. The tradeoff is that the decision module performs more extensive runtime checking and stores backup command sequences

produced by the black-box baseline controller at previous time steps. The complexity of runtime checking depends on the nature of the system model. For deterministic models, simulation suffices. However, if the model has uncertainty then we need to perform online reachability analysis. In the case of multi-agent systems, we further showed how to use command blending to safely reduce how often backup command sequences need to be used.

BSA reduces the difficult problem of proving high-level safety to a simpler problem of *performance optimization*: ensuring that the runtime checking completes before a decision is needed. The practicality of the approach was demonstrated through two significant case studies, including a mid-air collision avoidance system for groups of F-16 aircraft created from imperfect logic encoded in neural networks. This case study involves a highly complex nonlinear system with over a hundred dimensional variables and a neural-network-based controller. Black-Box Simplex provides a feasible path for runtime verification of systems that are otherwise unverifiable in practice.

Acknowledgement. This material is based upon work supported by National Science Foundation (NSF) under grant numbers ITE-2134840, OIA-2040599, CCF-1918225, CCF-1954837 and CPS-1446832, the Office of Naval Research (ONR) under grants N000142112719 and N000142212156, and the Air Force Office of Scientific Research (AFOSR) under award numbers FA9550-19-1-0288, FA9550-21-1-0121, FA9550-22-1-0450. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF, United States Air Force or the United States Navy.

Statements and Declarations

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Clark, M., Koutsoukos, X., Porter, J., Kumar, R., Pappas, G., Sokolsky, O., Lee, I., Pike, L.: A study on run time assurance for complex cyber physical systems. Technical report, Air Force Research Laboratory, Aerospace Systems Directorate (2013)

[2] Seto, D., Krogh, B., Sha, L., Chutinan, A.: The simplex architecture for safe online control system upgrades. In: Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No. 98CH36207), vol. 6 (1998). IEEE

[3] Sha, L.: Using simplicity to control complexity. IEEE Software **18**(4), 20–28 (2001). https://doi.org/10.1109/MS.2001.936213

[4] Desai, A., Ghosh, S., Seshia, S.A., Shankar, N., Tiwari, A.: SOTER: A runtime assurance framework for programming safe robotics systems. In: 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019 (2019)

[5] Phan, D., Yang, J., Grosu, R., Smolka, S.A., Stoller, S.D.: Collision avoidance for mobile robots with limited sensing and limited information about moving obstacles. Formal Methods in System Design **51**(1), 62–86 (2017)

[6] Schierman, J., DeVore, M.D., Richards, N., Gandhi, N., Cooper, J., Horneman, K.R., Stoller, S., Smolka, S.: Runtime assurance framework development for highly adaptive flight control systems. Report AD1010277, Defense Technical Information Center (2015)

[7] Mehmood, U., Bak, S., Smolka, S.A., Stoller, S.D.: Safe cps from unsafe controllers. In: Proceedings of the Workshop on Computation-Aware Algorithmic Design for Cyber-Physical Systems, pp. 26–28 (2021)

[8] Mehmood, U., Sheikhi, S., Bak, S., Smolka, S., Stoller, S.: The black-box simplex architecture for runtime assurance of autonomous cps. In: NASA Formal Methods Symposium (2022)

[9] Lin, Q., Chen, X., Khurana, A., Dolan, J.:

Reachflow: An online safety assurance framework for waypoint-following of self-driving cars. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2020)

[10] Bak, S., Johnson, T.T., Caccamo, M., Sha, L.: Real-time reachability for verified simplex design. In: 35th IEEE Real-Time Systems Symposium (RTSS 2014). IEEE Computer Society, Rome, Italy (2014)

[11] Althoff, M., Dolan, J.M.: Online verification of automated road vehicles using reachability analysis. IEEE Transactions on Robotics **30**(4) (2014)

[12] Phan, D., Grosu, R., Jansen, N., Paoletti, N., Smolka, S.A., Stoller, S.D.: Neural simplex architecture. In: NASA Formal Methods Symposium (NFM 2020) (2020)

[13] Bak, S., Chivukula, D.K., Adekunle, O., Sun, M., Caccamo, M., Sha, L.: The system-level simplex architecture for improved real-time embedded system safety. In: 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 99–107 (2009). IEEE

[14] Kapinski, J., Deshmukh, J.: Discovering forward invariant sets for nonlinear dynamical systems. In: Interdisciplinary Topics in Applied Mathematics, Modeling and Computational Science, pp. 259–264 (2015)

[15] Murray, R.M., Li, Z., Sastry, S.S., Sastry, S.S.: A Mathematical Introduction to Robotic Manipulation, (1994)

[16] Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. In: Autonomous Robot Vehicles, pp. 396–404 (1986)

[17] Girard, A.: Reachability of uncertain linear systems using zonotopes. In: International Workshop on Hybrid Systems: Computation and Control (2005). Springer

[18] Heidlauf, P., Collins, A., Bolender, M., Bak, S.: Verification challenges in f-16 ground collision avoidance and other automated maneuvers. In: 5th International Workshop on Applied Verification of Continuous and Hybrid Systems. EPiC Series in Computing, vol. 54 (2018)

[19] Stevens, B.L., Lewis, F.L., Johnson, E.N.: Aircraft Control and Simulation, (2015)

[20] Kochenderfer, M.J., Chryssanthacopoulos, J.: Robust airborne collision avoidance through dynamic programming. Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371 **130** (2011)

[21] Julian, K.D., Kochenderfer, M.J., Owen, M.P.: Deep neural network compression for aircraft collision avoidance systems. Journal of Guidance, Control, and Dynamics **42**(3), 598–608 (2019)

[22] Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: International Conference on Computer Aided Verification, pp. 97–117 (2017). Springer

[23] Marston, M., Baca, G.: ACAS-Xu initial self-separation flight tests. Technical report, NASA (2015)

[24] Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (vnn-comp 2021): Summary and results. arXiv preprint arXiv:2109.00498 (2021)

[25] Bak, S., Tran, H.-D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: Proceedings of the 32nd International Conference on Computer Aided Verification (2020)

[26] Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: International Conference on Computer Aided Verification, pp. 258–263 (2013). Springer

[27] Schouwenaars, T., Valenti, M., Feron, E.,

How, J.: Implementation and flight test results of MILP-based UAV guidance. 2005 IEEE Aerospace Conference, 1–13 (2005)

[28] Schouwenaars, T.: Safe trajectory planning of autonomous vehicles. PhD thesis, Massachusetts Institute of Technology (2006)

[29] Alsterda, J.P., Brown, M., Gerdes, J.C.: Contingency model predictive control for automated vehicles. In: 2019 American Control Conference (ACC), pp. 717–722 (2019). https://doi.org/10.23919/ACC.2019.8815260

[30] Magdici, S., Althoff, M.: Fail-safe motion planning of autonomous vehicles. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pp. 452–458 (2016). IEEE

[31] Schurmann, B., Klischat, M., Kochdumper, N., Althoff, M.: Formal safety net control using backward reachability analysis. IEEE Transactions on Automatic Control (2021)

[32] Saint-Pierre, P.: Approximation of the viability kernel. Applied Mathematics and Optimization **29**(2), 187–209 (1994)

[33] Kaynama, S., Maidens, J., Oishi, M., Mitchell, I.M., Dumont, G.A.: Computing the viability kernel using maximal reachable sets. In: Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control, pp. 55–64 (2012)

[34] Maidens, J.N., Kaynama, S., Mitchell, I.M., Oishi, M.M., Dumont, G.A.: Lagrangian methods for approximating the viability kernel in high-dimensional systems. Automatica **49**(7), 2017–2029 (2013)

[35] Phan, D., Grosu, R., Jansen, N., Paoletti, N., Smolka, S.A., Stoller, S.D.: Neural simplex architecture. In: NASA Formal Methods Symposium (NFM 2020), pp. 97–114 (2020). Springer

[36] Mashima, D., Chen, B., Zhou, T., Rajendran, R., Sikdar, B.: Securing substations through command authentication using on-the-fly simulation of power system dynamics. In: IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (2018)

[37] Borrmann, U., Wang, L., Ames, A.D., Egerstedt, M.: Control barrier certificates for safe swarm behavior. In: Egerstedt, M., Wardi, Y. (eds.) ADHS. IFAC-PapersOnLine, vol. 48, pp. 68–73 (2015)

[38] Gurriet, T., Mote, M., Ames, A.D., Feron, E.: An online approach to active set invariance. In: Conference on Decision and Control (2018). IEEE

[39] Gurriet, T., Mote, M., Singletary, A., Feron, E., Ames, A.D.: A scalable controlled set invariance framework with practical safety guarantees. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 2046–2053 (2019). IEEE

[40] Wang, L., Han, D., Egerstedt, M.: Permissive barrier certificates for safe stabilization using sum-of-squares. In: 2018 Annual American Control Conference, ACC 2018, pp. 585–590. IEEE, ??? (2018)

[41] Zhao, H., Zeng, X., Chen, T., Liu, Z.: Synthesizing barrier certificates using neural networks. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control. HSCC '20. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3365365.3382222

[42] Ames, A.D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., Tabuada, P.: Control barrier functions: Theory and applications. In: 18th European Control Conference, ECC 2019, Naples, Italy, pp. 3420–3431. IEEE, ??? (2019)

[43] Mehmood, U., Roy, S., Damare, A., Grosu, R., Smolka, S.A., Stoller, S.D.: A distributed simplex architecture for multi-agent systems. Journal of Systems Architecture **134**, 102784 (2023). https://doi.org/10.1016/j.sysarc.2022.102784

[44] Raju, D., Bharadwaj, S., Djeumou, F., Topcu, U.: Online synthesis for runtime enforcement of safety in multiagent systems. IEEE Transactions on Control of Network Systems **8**(2), 621–632 (2021). https://doi.org/10.1109/TCNS.2021.3061900

[45] Engelmann, D.C., Ferrando, A., Panisson, A.R., Ancona, D., Bordini, R.H., Mascardi, V.: RV4jaca – runtime verification for multi-agent systems. Electronic Proceedings in Theoretical Computer Science **362**, 23–36 (2022). https://doi.org/10.4204/eptcs.362.5

[46] Schneider, F.B.: Enforceable security policies **3**(1), 30–50 (2000). https://doi.org/10.1145/353323.353382

[47] Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies (2002)

[48] Falcone, Y., Mounier, L., Fernandez, J.-C., Richier, J.-L.: Runtime enforcement monitors: composition, synthesis, and enforcement abilities. Formal Methods in System Design **38** (2011). https://doi.org/10.1007/s10703-011-0114-4

[49] Pinisetty, S., Preoteasa, V., Tripakis, S., Jéron, T., Falcone, Y., Marchand, H.: Predictive runtime enforcement, pp. 1628–1633 (2016). https://doi.org/10.1145/2851613.2851827

[50] Rania Taleb, R.K. Sylvain Hallé: A modular runtime enforcement model using multitraces. Foundations and Practice of Security Lecture Notes in Computer Science, 283–302 (2022)