2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Spontaneous state constraint insertions to operating symbolic controllers combined with runtime assurance for task allocation in UAV missions

Marcus Kreuzer, Alexander Weber and Alexander Knoll

Abstract— **The computation of symbolic controllers for nonlinear plants is typically computationally expensive due to the well-known curse-of-dimensionality. In fact, those controllers must be computed before operating the closed loop. This note presents a method to modify symbolic controllers while they are operating the closed loop to avoid spontaneously inserted state obstacles. In addition, we utilize methods of plan recognition in combination with our new algorithm for providing a technique of decentralized runtime assurance for efficient task allocation and mission guidance in a multi-UAV setting. Promising results and the applicability of the found method is demonstrated by simulation and experiments with real physical systems.**

I. Introduction

Symbolic control is a method to compute feedback controllers in a fully automated manner given the plant dynamics and control objective [1]. An outstanding property of this method is that the controller, once successfully computed, provably enforces the specification. Tuning or verification steps are obsolete. Although symbolic control allows to take into account uncertainties in the plant dynamics, measurement errors, state and input obstacles, all of the mentioned constraints have to be known at the time of synthesis. Once the controller is computed changes in those constraints, thus in the assumed control problem, lead to loosing all guarantees about the behavior of the closed loop. As the synthesis technique suffers from the curse-of-dimensionality, the computation is typically time demanding making re-computations useless for maintaining the correctness of the controller. On the other hand, estimating the constraints overly conservative to cover all possibilities typically results in an unsolvable control problem.

In practice, there are plenty of control applications where constraints are quantitatively unknown beforehand, e.g. collision avoidance for navigation systems or more generally, cyber-physical systems operating in dynamically changing environments. To give a concrete example, consider a team of firefighting drones, each one steered by a (symbolic) controller in a decentralized architecture, e.g. [2]. Each drone is delivering water to the fires from a base station. During this mission, a new obstacle appears in the area, e.g. caused by a rescue helicopter crossing the area unexpectedly. See Fig. 1. In that moment, the symbolic controller steering the drone looses all its guarantees so that the drone may hit the new obstacle area. A complete re-computation of the symbolic controller is not an option as the runtime is typically in the order of hours [2, Sect. IV.A] and therefore severely exceeds the time constraints of a collision avoidance maneuver.

In this note we present a technique to correct the given controller quickly such that in reasonable situations the drone in the example above remains in a safe state. More generally, we provide a method to adjust an already computed symbolic controller when a new state constraint appears during the operation of the closed loop. Moreover, we show that the plan recognition methodology for symbolic optimal control, firstly introduced in [2], is functioning for the extension introduced in this note. Furthermore, we demonstrate the integration of the plan recognition monitor in a runtime assurance framework for efficient and communicationless mission task allocation in a multi-UAV setting. The runtime assurance methodology was firstly introduced by [3]. Decentralized approaches are rare and presented in latest works, e.g. [4]. The present note is thematically part of techniques for synthesis of symbolic controllers such as [5]–[7]. On the other hand, the algorithm to present is also related to the theory of hyper-graphs since it basically takes a hyper-graph as input and operates on it. In fact, coming from that direction the work closest to ours is [8]. In the latter work, algorithms are presented for maintaining shortest paths in large hyper-graphs based on the famous Dijkstra algorithm [9]. Our algorithm also aims at “maintaining” optimal paths in a changing hyper-graph, yet it is based on a version of the Bellman-Ford algorithm as presented in [6]. (For the advantages of the Bellman-Ford algorithm in the context of symbolic control, see also [6].)

The remaining sections of this note are organized as

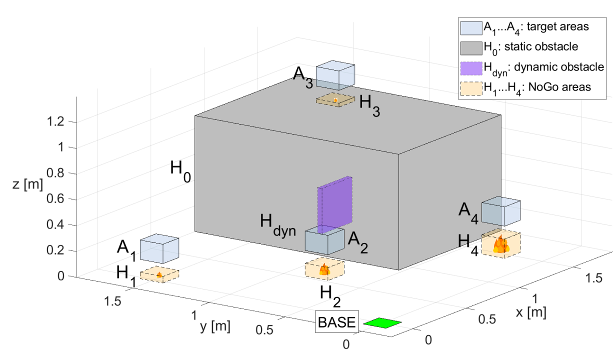


Fig. 1: Firefighting scenario. The orange- and gray-coloured cuboids represent the fires and a hill, respectively. The drone shall reach all of $A_1, \ldots, A_4$ and stay in each for some time without colliding with the obstacle H_{dyn}, which is spontaneously injected during flight.

The authors are with the Munich University of Applied Sciences, Dept. of Mechanical, Automotive and Aeronautical Eng., 80335 München, Germany.
This work has been supported by the German Federal Ministry of Education and Research (Project ARCUS; No. 13FH734IX6) and the Government of Upper Bavaria (Project EILT; No. HAM-2109-0030).

follows. The problem, which the novel algorithmic technique solves, is formally defined in Section II. The algorithm itself is presented and discussed in Section III. Two simulation examples are presented in Sections IV and V, respectively, where the latter includes a comprehensive demonstration for decentralized runtime assurance for task allocation in a multi-UAV setting. Finally, conclusions are given in Section VI.

II. Preliminaries and problem formulation

This paper builds upon symbolic (optimal) control as formulated in [1], [10] and takes up algorithmic techniques established in [6] for synthesizing symbolic controllers. Below, we give a short summary of the used concepts.

A. Notation

The set of integers and the set of non-negative integers is denoted by $\mathbb{Z}$ and $\mathbb{Z}_+$, respectively. For $a, b \in \mathbb{Z}$, $a \leq b$, the notation $[a; b]$ means the set $\{a, a+1, \dots, b\}$. The symbol $\emptyset$ stands for the empty set. For sets A and B the notation $f\colon A \rightrightarrows B$ means that f is a set-valued map. f is *strict* if $f(a) \neq \emptyset$ for all $a \in A$. The notation A^B stands for the set of all maps $B \to A$. In particular, $A^{\mathbb{Z}_+}$ is the set of all time-discrete signals with values in A. A signal $v \in \{0,1\}^{\mathbb{Z}_+}$ is *non-zero* if $v(t) = 1$ for some $t \in \mathbb{Z}_+$.

B. Plant and closed loop

This work uses the methodology of *transition systems* in discrete time to model plant dynamics. Specifically, the dynamics is assumed to be given by the difference inclusion

$$x(t+1) \in F(x(t), u(t)). \tag{1}$$

Here, x and u are the state and input signal taking values in nonempty sets X and U, respectively, and

$$F\colon X \times U \rightrightarrows X \tag{2}$$

is a strict set-valued map, so that the dynamics (1) is non-blocking and allows to take into account uncertainties. We refer to the triple

$$(X, U, F) \tag{3}$$

as *transition system*. A *controller*, which can be interconnected with (3), is then a strict set-valued map

$$\mu\colon X \rightrightarrows U \times \{0,1\}, \tag{4}$$

where the second component of the image is technically required to indicate if the controller is in operation ('0') or off ('1'). Consequently, the closed loop dynamics is characterized by (1) such that control signal u satisfies

$$u(t) \in \mu(x(t))_1 \tag{5}$$

for all $t \in \mathbb{Z}_+$. See Fig. 2.

The set $\mathcal{B}_p(\mu \times S)$ for the plant S in (3) and initial state $p \in X$ is of paramount importance in symbolic control, which is the set of all signals $(u, v, x) \in (U \times \{0,1\} \times X)^{\mathbb{Z}_+}$ that fulfil (1), (5) and $v(t) \in \mu(x(t))_2$ for all $t \in \mathbb{Z}_+$ with $x(0) = p$. In words, $\mathcal{B}_p(\mu \times S)$ contains all possible input-output pairs produced by the closed loop.

The performance of controllers is measured using the said set as summarized subsequently.

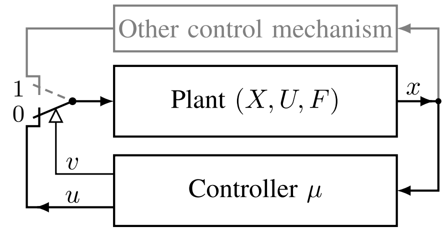


Fig. 2: Closed loop architecture [11] as considered in this work.

C. Closed loop performance

In order to quantify the performance of a controller, symbolic optimal control [10] uses in the simplest formulation the cost functional

$$J(u, v, x) = G(x(T)) + \sum_{t=0}^{T-1} g(x(t), x(t+1), u(t)) \tag{6}$$

with $T := \inf v^{-1}(1)$, where v is a non-zero signal, and with the following involved components. The *terminal cost*

$$G\colon X \to \mathbb{R}_+ \cup \{\infty\} \tag{7}$$

rates the state at stopping time while the *running cost*

$$g\colon X \times X \times U \to \mathbb{R}_+ \cup \{\infty\} \tag{8}$$

accumulatively rates state and control signal. Finally, the performance of a controller μ as in (4) is evaluated by the *closed-loop performance* $L_\mu\colon X \to \mathbb{R}_+ \cup \{\infty\}$ defined by

$$L_\mu(p) := \sup_{(u,v,x) \in \mathcal{B}_p(\mu \times S)} J(u, v, x). \tag{9}$$

Thus, performance is measured with respect to the *worst-case* trajectory of the closed loop.

D. Controller synthesis

To begin with, an *optimal control problem* is defined as the 5-tuple

$$(X, U, F, G, g) \tag{10}$$

with components as in (3), (7) and (8). The synthesis methods of symbolic optimal control aim at synthesizing controllers that minimize the closed-loop performance (9) as defined for (10) for every initial state.

The key idea of symbolic controller synthesis is to transfer the original problem with possibly continuous state space to a problem with *discrete* state and input space [1]. The details of the said transformation are not relevant for this note so that we may assume in the remaining part of this section that a *finite* transition system is given, which is a system (3) such that X and U are finite sets. Together with the transitions function F we can interpret (3) as a hyper-graph. (Edges from states may point to several successor states.)

Having briefly reviewed the concepts used in symbolic control, we can define the problem to be addressed in this note.

E. Problem formulation

First of all, we recall that classical reach-avoid problems can be formulated using the formalism above [10, Sect. III]. In particular, state obstacles are modelled using the running cost function (8) as follows. A non-empty set $H \subseteq X$ is a *(state) obstacle* if the following condition is true:

$$y \in H \Rightarrow \forall_{(x,u) \in X \times U} : g(x, y, u) = \infty. \tag{11}$$

The terminal cost in (7) for a reach-avoid problem is defined as 0 on target states and as ∞ on any other state.

Let an optimal control problem (10) be given, a controller μ as in (4) and a non-empty set $A \subseteq X$ such that $L_\mu(p)$ is finite for every $p \in A$. In words, the controller μ solves the problem for all initial states in A in the sense that the target is finally reached. (Optimality of the controller is not relevant for our purposes.) In this case, for every $p \in A$, there is a control symbol available by which we mean that $\mu(p)_1$ contains exactly one element of U. (Subsequently, we assume that $L_\mu(p) = \infty$ implies $\mu(p)_1 = U$.)

Now, assume that the set of state obstacles is enlarged at some point of time while the closed loop is operating and thus, some $H^* \supseteq H$ becomes valid. I.e. the function g in (10) becomes a function g^* satisfying (11) with H^* in place of H. The obvious approach to recompute a controller for the new control problem (X, U, F, G, g^*) requires typically as much runtime as the computation of μ.

The main contribution of this note is an algorithm that computes a stopgap quickly with the following properties: Consider the control problem

$$(X, U, F, 0, g^*), \tag{12}$$

where the 0 means that the terminal cost function is identically zero on its domain. Consider the corresponding cost functional J^* for this modified problem. The key property of the algorithm is that J^* remains *finite* whenever a control symbol is (still) available for current state of the plant, so the newly inserted obstacle (or any other obstacle) is not hit at any time.

III. Algorithm

To begin with, we note that the notion of *predecessors* of states is meaningful in our context, which is the set

$$\operatorname{pred}(x, u) = \{y \in X \mid x \in F(y, u)\}. \tag{13}$$

The idea of the algorithm is to trace back predecessors of states that are new obstacle states. The control symbol is "corrected" in a certain sense if a transition is possible to unaffected states.

More specifically, the controller to return is initialized as the original controller μ firstly (**line 1**) and the new obstacle states are thrown into the set $\mathcal{F}$ (**line 2**). Iteratively, every predecessor y of a state in $\mathcal{F}$ shall change the control symbol (**line 8**) if the original controller steers y into a state in $\mathcal{F}$ (**lines 6–7**). Otherwise, the state y is also declared a new obstacle state (**lines 23–24**).

The search for an alternative control symbol (**lines 17–24**) tests all remaining elements of U by testing successors (**line 18**): If all successors have a valid control symbol then y can be assigned an alternative control symbol (**line 19**).

Algorithm 1 Spontaneous state constraint insertion

Input: X, U, F, μ, H_{new},
Assumption: $L_\mu(p) = \infty \Rightarrow \mu(p)_1 = U$ // see (9)
1: $\mu_{\text{new}} \leftarrow \mu$
2: $\mathcal{F} \leftarrow H_{\text{new}}$
3: **while** $\mathcal{F} \neq \emptyset$ **do**
4: **for all** $x \in \mathcal{F}$ **do**
5: **for all** $u \in U$ **do**
6: **for all** $y \in \operatorname{pred}(x, u)$ **do**
7: **if** $u \in \mu(y)_1$ **and** $x \in F(y, u)$ **then**
8: TRYFINDALTERNATIVE(y)
9: **end if**
10: **end for**
11: **end for**
12: **end for**
13: $\mathcal{F} \leftarrow \mathcal{F}'$
14: $\mathcal{F}' \leftarrow \emptyset$
15: **end while**
Output: μ_{new}

16: **procedure** TRYFINDALTERNATIVE(y)
17: **for all** $u' \in U \setminus \{u\}$ **do**
18: **if** $\forall_{z \in F(y,u')} : \mu_{\text{new}}(z) \neq U$ **then**
19: $\mu_{\text{new}}(y)_1 \leftarrow \{u'\}$ // alternative found
20: **return**
21: **end if**
22: **end for**
23: $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{y\}$ // no alternative
24: $\mu_{\text{new}}(y)_1 \leftarrow U$
25: **end procedure**

The following theorem describes technically the fact that the state signal of the closed loop will never reach H_{new} if it did not for the original controller and if the algorithm succeeded in finding an alternative control symbol for the initial state.

III.1 Theorem. *Consider* (3), (4) *and* $H_{\text{new}} \subseteq X$ *as inputs to Alg. 1. Further, let* $p \in X$, *let* v *be a non-zero stopping signal, let* J^* *be the cost functional for the control problem in* (12) *and let* S *be the transition system in* (3). *The controller* μ_{new} *returned by Alg. 1 is such that*

$$J^*(u, v, x) < \infty$$

for all $(u, x) \in \mathcal{B}_p(\mu_{\text{new}} \times S)$ *if* $L_\mu(p) < \infty$ *and if* $\mu_{\text{new}}(p) \neq U$.

Proof. Let $p \in X$ be such that the condition in the statement of the theorem holds. Since $\mu_{\text{new}}(p) \neq U$ either $\mu_{\text{new}}(p)_1 = \mu(p)_1$ or $\mu_{\text{new}}(p)$ had been assigned in line 19 of Alg. 1. In the first case the state trajectory is as for the original controller since p was never an element of $\mathcal{F}$. In the second case, the test in line 18 ensures that $g(p, x_1, u) < \infty$ for all $x_1 \in F(p, u)$ with $u \in \mu(p)_1$. By induction, the proof is complete. □

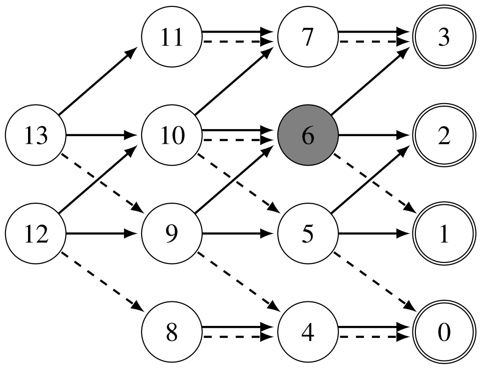


Fig. 3: Transition system with 14 states and two input elements, where the states $0, 1, 2, 3$ are target states. The original controller μ is defined as to enforce (only) the transitions drawn with a solid black line for each state.

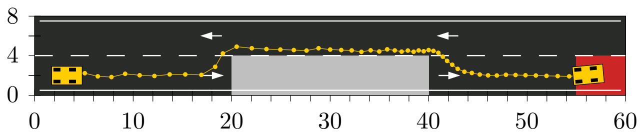


Fig. 4: Spontaneous spatial obstacle avoidance scenario of Section IV.

We would like to emphasize that there is no guarantee that the controller returned by Alg. 1 will solve the original optimal control problem in (10). The only given guarantee is that the new obstacle will be avoided. Therefore, optimality is abandoned in favour of execution time (runtime) so that a "safety" controller is available as fast as possible.

We briefly illustrate the algorithm on a small example.

III.1 Example. The transition system to consider is depicted in Fig. 3. In fact, the system has 14 states and the control symbols 0 and 1, whose transitions are the solid and dashed arrows, respectively. E.g., for the transition function F of this system it holds $F(10, 0) = \{6, 7\}$, $F(10, 1) = \{5, 6\}$ and $\mathrm{pred}(6, 0) = \{9, 10\}$, $\mathrm{pred}(6, 1) = \{10\}$. The initial controller is such that every state ends in one of the states $\{0, 1, 2, 3\}$. Now, the state 6 is declared a new obstacle, i.e. $H_{\text{new}} = \{6\}$, which implies that a trajectory starting in state 13 may result in the obstacle state 6. In particular, μ may steer state 13 to state 10 and subsequently to state 6. In order to avoid state trajectories to reach that unwanted state Alg. 1 is applied and proceeds as follows. Initially, $\mathcal{F} = H_{\text{new}} = \{6\}$. The function TRYFINDALTERNATIVE fails for state $10 \in \mathrm{pred}(6, 0)$ since for the alternative control symbol 1 it holds $6 \in F(10, 1)$. In contrast, $\mu_1(9) = \{1\}$ in line 19 since $F(9, 1) = \{4\}$. Hence, $\mathcal{F} = \{10\}$ in line 23. In the last iteration of the algorithm, $\mu_{\text{new}}(x)_1$ becomes $\{1\}$ for $x \in \{13, 14\}$ since $\mathrm{pred}(10, 0) = \{13, 14\}$. Consequently, μ_{new} steers state 13 to state 9 and finally to state 0, at which $\mu_{\text{new}}(0)_2 = \mu(0)_2 = 1$ indicates that a target state has been reached. □

IV. Simulation Example – Vehicle on a Road

We consider a vehicle on a road, which abruptly has to avoid an obstacle on the roadway and to reduce its velocity, respectively. The states of the vehicle model are the planar position (x_1, x_2), the orientation x_3 and the velocity x_4 of

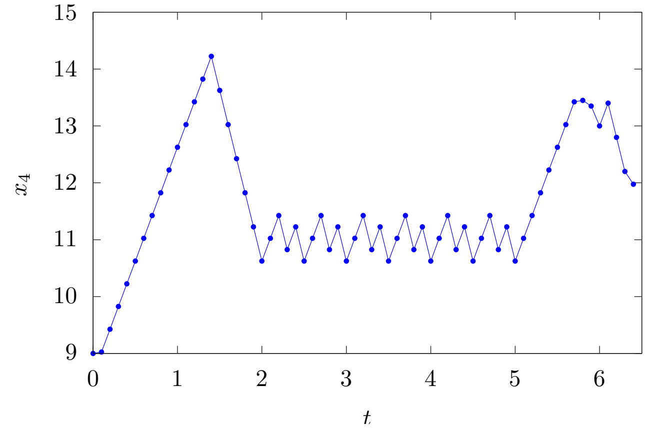


Fig. 5: Reduction of the velocity of the vehicle due to the spontaneously inserted obstacle H_1 in (15).

the vehicle while the inputs are the acceleration u_1 and the steering angle u_2. Specifically, we use dynamics [12], [13]

$$\dot{x} \in f(x, u) + W$$

with

$$f(x, u) = \\ (x_4 \cos(\alpha + x_3) \cdot \beta, x_4 \sin(\alpha + x_3) \cdot \beta, x_4 \tan(u_2), u_1),$$

where $U = [-6, 4] \times [-0.5, 0.5]$, $\alpha = \arctan(\tan(u_2)/2)$, $\beta = \cos(\alpha)^{-1}$, $W = \{(0, 0)\} \times [-0.01, 0.01] \times [-0.1, 0.1]$. The initial controller is synthesized to work on the states in

$$[0, 60] \times [0, 8] \times \mathbb{R} \times [0, 18]$$

and is specified to steer the vehicle into the set $[55, 60] \times [0, 4] \times \mathbb{R} \times [0, 18]$. The running cost function enforces a proper driving style. It is defined through

$$g(x, y, u) = \tau^2 + u_2^2 + \min_{m \in M} \|x - m\|_2,$$

where $\tau = 0.1$ and M is the union of the two centerlines of the roadways, i.e. the union of $[0, 60] \times \{2\} \times [-7\pi/16, 7\pi/16] \times [0, 18]$ and $[0, 60] \times \{6\} \times [9\pi/16, 23\pi/16] \times [0, 18]$. Next, we apply Alg. 1 to this plant and the following two obstacles (separately):

$$H_0 = [20, 40] \times [0, 4] \times \mathbb{R} \times [0, 18] \quad (14)$$
$$H_1 = [20, 40] \times [0, 8] \times \mathbb{R} \times [12, 18] \quad (15)$$

For H_0 the vehicle must avoid the spatial obstacle while for H_1 a reduction of the vehicle speed is enough. The computation of the initial controller takes about 15 minutes while Alg. 1 needs 25 and 11 seconds for H_0 and H_1, respectively. The software implementation of Alg. 1 is based on [14] and for this example run on 24 threads on Intel Xeon E5-2697 v3 (2.6 GHz). Some interesting parts of the trajectories are depicted in Fig. 4 and Fig. 5, respectively.

V. Runtime Assurance for Task Allocation in a Multi-UAV Scenario

In addition to the single UAV mission control framework and the plan recognition monitor (PRM) we presented in [2], in this work we (i) integrated the collision avoidance

(CA) algorithm described above and (ii) we developed the framework further to be used in a communicationless, cooperative multi-UAV setting: In this case-study, we extended the PRM to substitute communication requirements for multi-agent cooperation by state estimations of teammates. Additionally, we implemented a decentralized Runtime Assurance (RTA) mechanism to assure efficient mission progress and completion in the presence of environmental disturbances and spatio-temporal uncertainty of cyber-physical systems. In the following we will give a short introduction to the touched subjects and methods. We then introduce the scenario and describe the progress of a mission alongside its timeline.

A. Plan Recognition for state estimation

Referenced as *plan recognition by planning*, [15], [16] introduced a methodology, which allows an observer to infer the current behavior of a system by comparing its actions to a set of virtual plans generated by a planning algorithm. Inspired by approaches for the navigational domain [17]–[21], we contributed the integration of the methodology in the symbolic optimal control framework in [2]:

In short: We assume, that there is a strict set-valued map $H \colon X \rightrightarrows Y$ available to control the plant (3), which maps states to subsets of a given (non-empty) output set Y. Further, a potentially incomplete sequence $(o(0), \ldots, o(T))$ of observations in Y of the plant (3) is available, which has the property that there exists for all $t \in [0;T]$ an initial state $p \in X$, a controller μ of the form (4) and $(u, x) \in \mathcal{B}_p(\mu \times S)$ such that

$$o(t) \in H(x(t)). \tag{16}$$

The PR problem can now be defined: Given a set of $K > 1$ controllers $\{\mu_1, \ldots, \mu_K\}$ for plant S and an observation sequence $(o(0), \ldots, o(T))$, $T > 0$, identify $\mu_* \in \{\mu_1, \ldots, \mu_K\}$ such that there exists $p \in X$ and $(u, x) \in \mathcal{B}_p(\mu \times S)$ satisfying (16) for all $t \in [0;T]$.

By experience (see [2]), we have to cover uncertainty and real-world inaccuracy: We assume $Y = \mathbb{R}^3$ and H is given by $H(x) = x + [-\kappa/2, \kappa/2]^3$, where we set the acceptable offset threshold $\kappa = 0.15$[m], which is the diagonal dimension of the drone.

By observing an UAV's position over time and solving this problem online while mission execution, the PRM identifies μ_*, and thus is able to infer the future trajectory and the target state(s) of an observed, cooperative system without direct communication. For the RTA mechanism, this information is fundamental to make efficient task allocations in multi-UAV missions settled in dynamic environments.

B. Decentralized Runtime Assurance Mechanism

Traditional RTA mechanisms survey parameters of a single system to keep it in a defined set of safe states [22]–[26], using variations of the *simplex architecture*, introduced by the foundational work of [3], [27]. In newer approaches, (de-)centralized multi-system RTA architectures at the level of control and/or guidance are considered. These approaches rely on the (implicit) assumption of available and correct information of system states between agents or subsystems

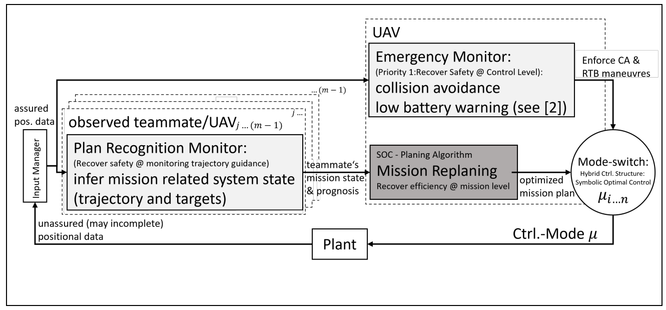


Fig. 6: Architecture of the Runtime Assurance Mechanism

[4], [28], [29]. Additionally in our work, we aim at assuring not only safe, but (i)cost efficient task execution at mission planning level. Furthermore, we (ii) assume, that there is only a minimum, partial set of state information obtainable (incomplete positional data of UAVs/agents). The RTA mechanism to be introduced is composed of the following components, depicted in Fig.6: An *Input manager* is processing incomplete positional data by filtering and interpolation. As described before, *Plan Recognition Monitors* provide the teammates mission states assured by observations. A *Mission replanning algorithm* (MP), which is based on [2], [30] and extended in this work for comparison of estimated mission costs of all UAVs to reallocate own tasks for minimum sum of costs. The *Emergency monitor* is enforcing collision avoidance, respectively return-to-base maneuvers triggered by thresholds at any time. The *Mode switch* is commanded by the monitors, respectively by MP. A built-in *waypoint manager* is steering the plant by processing the sequence of targets planned by the MP.

The RTA mechanism is executed by each of m individual UAVs and not influenced by any central decisive or controlling entity. Thus, we denote it as decentralized. For each of $(m-1)$ observed teammates, an UAV has to execute one PRM module instance, which calculates up to K virtual plans based on control modes $\{\mu_1, ..., \mu_K\}$. Thus, scalability can be defined as: $\mathcal{O}((m-1) \times K)$. We now introduce the firefighting scenario and describe the above mentioned implementations by a test case resulting in a successful mission completion. See a timeline for mission execution in Fig.7.

C. Multi-UAV firefighting scenario

The firefighting scenario is settled in a map with targets $A_1, ..., A_4$, the No-Go Areas $H_0, ..., H_4$ and a dynamic obstacle H_{dyn}, which appears during mission to trigger a CA maneuver of UAV1(red). The start- and end-state of the mission is at the base A_{base}. The task is to visit targets A_i and place retardant to fires without colliding with $H_0, ..., H_4$, H_{dyn} or a teammate. See Tab.I and Fig.8.

Two homogeneous UAVs are initiated with a pre-computed mission plan, which is defined by assuming a single UAV mission, forcing one single UAV to visit all targets, depicted by the green trajectory, Fig.8a. This represents a worst-case scenario for efficient firefighting: A single UAV will hardly be able to carry enough retardant and reach all fires in time to prohibit them from growing. We now describe mission

Symbol & Meaning	Value (north-east-up coordinate system)
X_0 (mission area)	$[-0.2, 1.8] \times [-0.2, 1.8] \times [-0.2, 1.8]$
A_{base} (base)	$[-0.1, 0.1] \times [-0.1, 0.1] \times \{0\}$
A_{hover} (generic hover area)	$[-0.1, 0.1] \times [-0.1, 0.1] \times [0, 0.15]$
A_1 (water release area)	$(0.05, 1.5, 0.15) + A_{\text{hover}}$
A_2 (water release area)	$(0.6, 0.8, 0.2) + A_{\text{hover}}$
A_3 (water release area)	$(1.36, 1.26, 1.13) + A_{\text{hover}}$
A_4 (water release area)	$(1.4, 0.2, 0.3) + A_{\text{hover}}$
H_0 (obstacle/hill)	$[0.91, 1.8] \times \ldots$ $[1.53, 1.8] \times [-0.2, 0.9]$
$H_1, \ldots, H_4$ (fires)	$-(0, 0, 0.25) + A_i,\ i \in \{1, \ldots, 4\}$
H_{dyn} (dynamic obstacle)	$[0.22, 0.5] \times \ldots$ $[0.55, 0.58] \times [0.50, 0.85]$

TABLE I: Data of spacial objects of the mission. Negative heights in X_0 are due to the fact that the positioning system used for real world experiments generates a slightly tilted coordinate system in space, see [2]. The presented collision avoidance algorithm works with arbitrary coordinates of H_{dyn}, hence they are fixed here for documentation reasons. Units are meters.

progress and actions taken by the RTA mechanism and its components:

After take-off at t_{45}, UAV1 is heading to A_4. Still on ground, UAV2 (blue) observes UAV1 violating the threshold κ and initializes a PR process at t_{55} (black asterisk in Fig.8a), which is generating virtual trajectories to all targets, see red dotted lines in Fig.8a. The PR component always considers all targets, even if already visited. For safety reasons, it's important to identify the current target of an UAV, no matter if it's a faulty choice in relation to mission efficiency.

In the following UAV2's PRM infers, that UAV1 is pursuing the planned trajectory slightly offset, but still heading to A_4. To minimize κ, this offset is adapted by UAV2's monitor through replanning the estimated trajectory of UAV1, starting from its current observed position, visiting remaining targets. The resulting trajectory estimation $A_4 \rightarrow A_3 \rightarrow A_1 \rightarrow A_2 \rightarrow A_{base}$ is shown by the thin red line in Fig.8b, starting at the correction marked $c_1@t_{59}$. It will be monitored from now, representing the *safe set* of UAV1's mission trajectory (considering κ, too).

Since UAV2's PRM inferred, that UAV1 is flying to A_4, the MP component of the RTA mechanism is skipping A_4 at t_{63} for cost reduction: A new plan, including only the remaining targets, is calculated and commanded to the *Mode-switch* component. This results in UAV2 heading directly for $A_2 \rightarrow A_1 \rightarrow A_3 \rightarrow A_{base}$, right after take off at t_{128}, Fig.8b.

Following the experiment's constraint of not communicating any mission replanning, respectively task re-allocation, the PRM of UAV1 gets not informed and is still expecting UAV2 to follow the initial mission trajectory leading to A_4 (in fact, UAV2 is now heading to A_2). At t_{148}, the PRM of UAV1, supervising the actual flown trajectory of UAV2, is identifying the violation of κ by UAV2. After starting a PR process (see blue dotted lines in Fig.8b), UAV1 successfully identifies A_2 as UAV2's current target at t_{152}, marked $c_2@t_{152}$ in Fig.8c.

Thus, the MP component of UAV1 skips A_2 at t_{152} and replans its mission to $A_3 \rightarrow A_1 \rightarrow A_{base}$. In addition, the PRM of UAV1 predicts the ongoing mission of UAV2 to follow $A_2 \rightarrow A_1 \rightarrow A_3 \rightarrow A_{base}$ depicted by the thin blue line. Since there is no communication, the UAVs have no knowledge about the planned target sequence of teammates. Hence, for safety reasons and complete mission execution the MP mechanism always considers all remaining targets until they are marked as surely visited by observation of the PR component. This is why A_1 is considered in both UAV's plans, see sequences above.

$t_{148} - t_{192}$: Both UAVs follow their predicted trajectories, see thin red, respectively blue lines and compare Fig.8b and Fig.8c. Since skipped by both UAVs at this time and thus not valid any more, the thin green line representing the initial worst-case single UAV mission plan is removed.

While UAV1 is leaving target A_3 and heading for A_1, the PRM of UAV1 observes UAV2 flying to A_1, too. UAV1's MP monitor compares estimated costs from current positions of both UAVs to reach A_1 and finds, that UAV2 will reach A_1 accumulating less cost. This triggers a MP process for UAV1 at t_{186}: Since there is no remaining target (A_2 was supplied by UAV2), it is heading to base A_{base}.

On its way to and while supplying target A_1, UAV2 observed UAV1 visiting target A_3 (see Fig.8c and Fig.8d). Since now (Fig.8e), there is no remaining target, the MP component of UAV2's RTA mechanism is adapting its mission plan by skipping A_3 at t_{163} and planning to return directly to A_{base}.

At t_{192} the PRM of UAV2 recognizes this target change of UAV1 by identifying an offset exceeding κ to the latest (recall c_1 at t_{59}, Fig.8b) estimated trajectory and initializes a PR process at t_{192}. Starting from the current position of UAV1 at t_{192}, virtual trajectories to A_2 and A_{base} lay close together (see Fig.8c). This causes the PRM to process a high number of observations to get a clear inference for the current target A_{base} of UAV1. The PRM accepts the new target at t_{216}, see marker $c_3@t_{216}$ in Fig.8d.

We assume, that e.g. a rescue helicopter is intruding the airspace and blocking the volume H_{dyn}, which is injected at t_{208}. The calculation time for the collision avoidance maneuver, respectively a trajectory starting at the current position of UAV1 at t_{208} (see black triangle marker, Fig. 8d) to its target A_{base} considering H_{dyn}, is $0.276[s]$. This forces UAV1 to conduct a collision avoidance maneuver at t_{215}, which is causing a strong trajectory offset $d_1 > \kappa$ and thus a PR process of UAV2 monitoring UAV1 at t_{218}, Fig.8d. At t_{221} UAV1 finishes to circumfly obstacle H_{dyn} and heads on pursuing its plan to target A_{base}. This is inferred by UAV2's PRM at t_{222}, see the adaption of trajectory estimation $c_4@t_{222}$ in Fig.8e.

The adjustment c_4 does not exactly fit to the flown trajectory, see distance $d_2 < \kappa$, but the PR is able to identify the correct target, since the flown trajectory lies within the maximum accepted offset threshold κ. This shows the PRM's robustness against unknown, severe disturbances, since by implementation it is not yet able to identify the collision avoidance mode in particular.

Leaving A_1, UAV2's MP process at t_{163} is now provoking a trajectory offset $> \kappa$, thus UAV1 starts a PR process at t_{230}, see Fig.8e. The inference mechanism identifies the correct target A_{base} at t_{237}, skips the former trajectory esti-

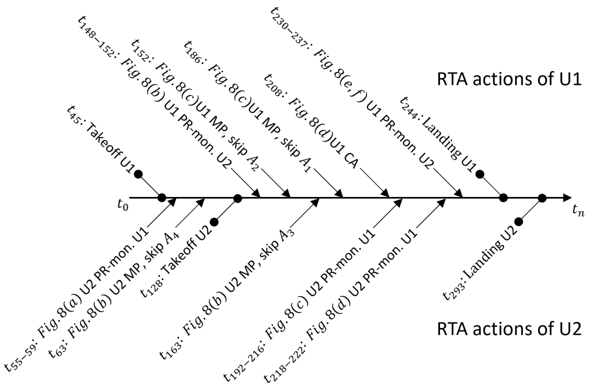


Fig. 7: Timeline of the Runtime Assurance Mechanism's interventions: Plan Recognition Monitoring (PR-mon.), Mission Planning (MP) and Collision Avoidance (CA) in a timely ordered sequence for UAV1(U1) and UAV2(U2), qualitative illustration of intervals.

mation (from t_{152}) and adopts the current one, see $c_5@t_{237}$ in Fig.8f. The end-state of the mission is depicted in Fig.8f.

VI. Conclusion

A drawback of symbolic control is the curse-of-dimensionality. Extensive computing times prevent the method from being applied to online calculations, e.g. in the case of sudden changes in environmental constraints. We developed a method to adapt an existing symbolic controller to new state constraints at an arbitrary point of time while executing the closed loop. We demonstrated the successful implementation of the presented algorithm by two examples settled in CA scenarios. In addition, we showed the successful integration of the method into a RTA mechanism. A detailed description of an example shows the robustness of the developed architecture and its components. Future work investigates an extension of the algorithm towards inclusion of optimality, which was given up in this work in order to succesfully minimize calculation time. Additionaly, the integration of particular CA modes to the PRM - solving the trade-off of short observation times vs. precise inference, has to be examined. Furthermore, the presented methods are to be applied and tested with cyber-physical systems.

References

[1] G. Reissig, A. Weber, and M. Rungger, "Feedback refinement relations for the synthesis of symbolic controllers," *IEEE Trans. Automat. Control*, vol. 62, no. 4, pp. 1781–1796, Apr. 2017.

[2] M. Kreuzer, A. Weber, C. Leupolz, and A. Knoll, "Symbolic control applied to miniature quadcopter mission guidance," in *2023 European Control Conference (ECC)*, 2023, pp. 1–8.

[3] L. Sha *et al.*, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[4] U. Mehmood, S. Sheikhi, S. Bak, S. A. Smolka, and S. D. Stoller, "The black-box simplex architecture for runtime assurance of autonomous cps," in *NASA Formal Methods Symposium*. Springer, 2022, pp. 231–250.

[5] K. Hsu, R. Majumdar, K. Mallik, and A. Schmuck, "Lazy abstraction-based control for safety specifications," in *Proc. IEEE Conf. Decision and Control (CDC)*. New York: IEEE, Dec. 2018, pp. 4902–4907.

[6] A. Weber, M. Kreuzer, and A. Knoll, "A generalized Bellman-Ford algorithm for application in symbolic optimal control," in *Proc. European Control Conf. (ECC)*, May 2020, pp. 2007–2014. [Online]. Available: https://arxiv.org/abs/2001.06231

[7] E. Macoveiciuc and G. Reissig, "On-the-fly symbolic synthesis with memory reduction guarantees," *IEEE Trans. Automat. Control*, pp. 1–8, 2022.

[8] J. Gao, Q. Zhao, W. Ren, A. Swami, R. Ramanathan, and A. Bar-Noy, "Dynamic shortest path algorithms for hypergraphs," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1805–1817, 2015.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[10] G. Reissig and M. Rungger, "Symbolic optimal control," *IEEE Trans. Automat. Control*, vol. 64, no. 6, pp. 2224–2239, June 2019.

[11] ——, "Abstraction-based solution of optimal stopping problems under uncertainty," in *Proc. IEEE Conf. Decision and Control (CDC)*. New York: IEEE, Dec 2013, pp. 3190–3196.

[12] K. J. Åström and R. M. Murray, *Feedback systems*. Princeton University Press, Princeton, NJ, 2008.

[13] A. Weber and A. Knoll, "Approximately optimal controllers for quantitative two-phase reach-avoid problems on nonlinear systems," in *Proc. IEEE Conf. Decision and Control (CDC)*, 2020, pp. 430–437. [Online]. Available: http://arxiv.org/abs/2006.03862

[14] A. Weber, E. Macoveiciuc, and G. Reissig, "ABS: A formally correct software tool for space-efficient symbolic synthesis," in *25th ACM Intl. Conf.Hybrid Systems: Computation and Control (HSCC)*, ser. HSCC '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3501710.3519519

[15] M. Ramírez and H. Geffner, "Plan recognition as planning," in *Twenty-First intl. joint conference on artificial intelligence*, 2009.

[16] ——, "Probabilistic plan recognition using off-the-shelf classical planners," in *Twenty-Fourth AAAI Conf. on Artificial Intelligence*, 2010.

[17] G. Fitzpatrick, N. Lipovetzky, M. Papasimeon, M. Ramirez, and M. Vered, "Behaviour recognition with kinodynamic planning over continuous domains," *Frontiers in Artificial Intelligence*, p. 156, 2021.

[18] G. A. Kaminka, M. Vered, and N. Agmon, "Plan recognition in continuous domains," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] P. Masters and S. Sardina, "Cost-based goal recognition in navigational domains," *Journal Artificial Intelligence Research*, vol. 64, pp. 197–242, 2019.

[20] ——, "Cost-based goal recognition for the path-planning domain." in *IJCAI*, 2018, pp. 5329–5333.

[21] M. Vered, G. A. Kaminka, and S. Biham, "Online goal recognition through mirroring: Humans and agents," in *The Fourth Annual Conf. Advances in Cognitive Systems*, 2016.

[22] C. Hanson, "Capability description for nasa's f/a-18 tn 853 as a testbed for the integrated resilient aircraft control project," Tech. Rep., 2009.

[23] T. S. VanZwieten, E. T. Gilligan, J. H. Wall, J. S. Orr, C. J. Miller, and C. E. Hanson, "Adaptive augmenting control flight characterization experiment on an f/a-18," in *2014 American Astronautical Society Guidance & Control Conf.*, no. M14-3205, 2014.

[24] D. Cofer, I. Amundson, R. Sattigeri, A. Passi, C. Boggs, E. Smith, L. Gilham, T. Byun, and S. Rayadurgam, "Run-time assurance for learning-based aircraft taxiing," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conf. (DASC)*. IEEE, 2020, pp. 1–9.

[25] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari, "Soter: a runtime assurance framework for programming safe robotics systems," in *2019 49th Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 138–150.

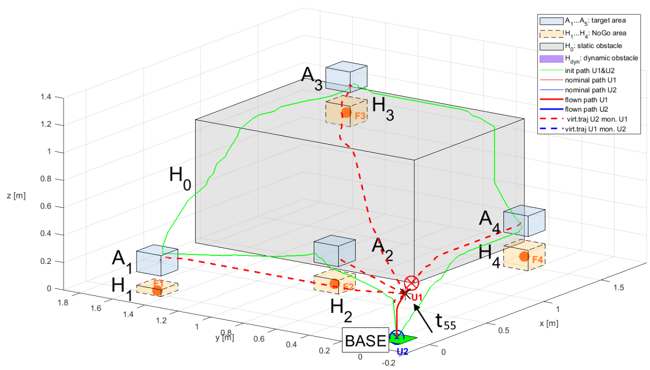
[26] J. D. Schierman, M. D. DeVore, N. D. Richards, and M. A. Clark, "Runtime assurance for autonomous aerospace systems," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 12, pp. 2205–2217, 2020.

[27] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe online control system upgrades," in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No. 98CH36207)*, vol. 6. IEEE, 1998, pp. 3504–3508.

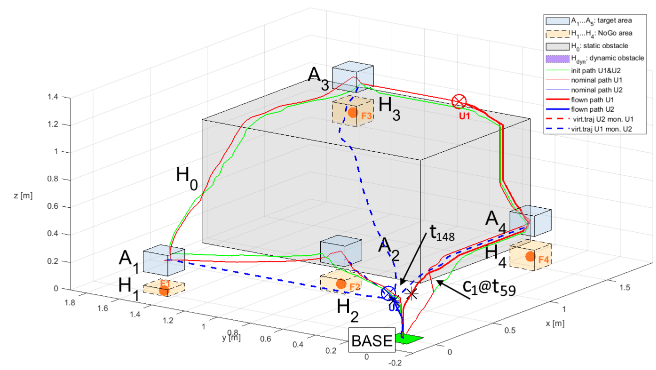
[28] S. Ghori, T. Khamvilai, E. Feron, and M. Pakmehr, "Runtime assurance for distributed avionics architecture," in *2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC)*. IEEE, 2022, pp. 1–6.

[29] U. Mehmood, S. Roy, A. Damare, R. Grosu, S. A. Smolka, and S. D. Stoller, "A distributed simplex architecture for multi-agent systems," *Journal of Systems Architecture*, vol. 134, p. 102784, 2023.

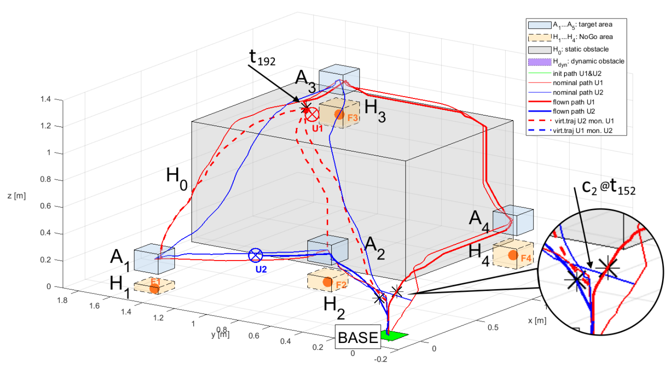
[30] A. Weber, F. Fiege, and A. Knoll, "Vehicle mission guidance by symbolic optimal control," in *Proc. European Control Conf. (ECC)*, Jul. 2022, pp. 1235–1241. [Online]. Available: https://arxiv.org/abs/2205.14085



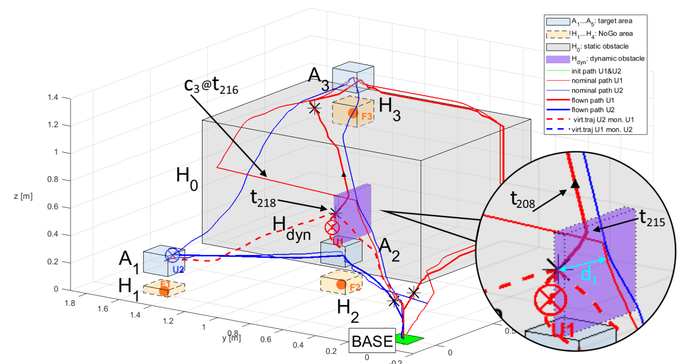
(a) Takeoff(t_{45}) of UAV1 and first PR(t_{55}) of UAV2, still on ground, monitoring UAV1. Red dotted lines represent virtual trajectories.



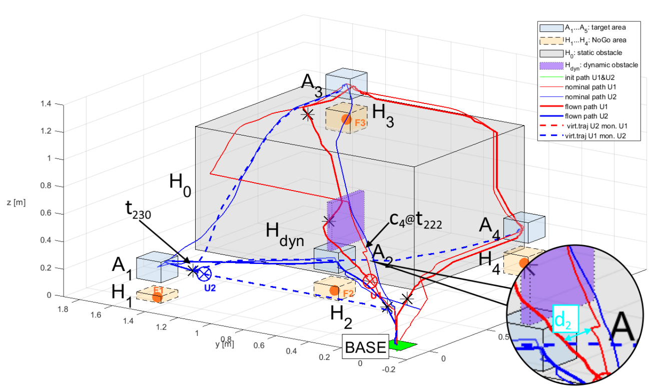
(b) Adaption of UAV1's estimated trajectory by UAV2's PRM at c_1. Takeoff(t_{128}) of UAV2 and first PR(t_{148}) of UAV1 monitoring UAV2.



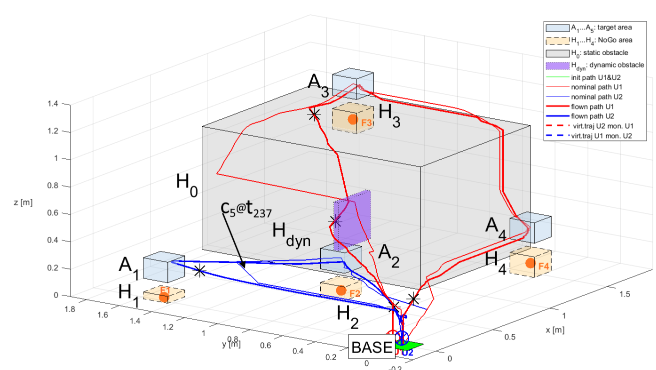
(c) UAV1's PRM adopts its replanned estimation of UAV2's mission, see c_2 at t_{152}. UAV1 skips A_1 at t_{186} causing a trajectory violation observed by UAV2 (t_{192}).



(d) H_{dyn} is injected at t_{208}. UAV2 finds a new prediciton of UAV1's path (c_3@t_{216}), which is discarded two time steps later (t_{218}) due to the collision avoidance maneuver (t_{215}) causing the offset $d_1 > \kappa$.



(e) The PRM of UAV2 infers the correct target of UAV1 (c_4@t_{222}) with a slight offset in trajectory prediction. UAV1's PRM observes UAV2 skipping A_3 and triggers a PR process at t_{230}.



(f) UAV1 completes the PR (c_5@t_{237}) by identifying A_{base} as UAV2's target. UAV2 finishes its mission by landing at the base. This scene represents the end state of successful mission execution.

Fig. 8: Mission execution shown in scenes.