SkyTrakx: A Toolkit for Simulation and Verification of Unmanned Air-Traffic Management Systems

Chiao Hsieh, Hussein Sibai, Hebron Taylor, Yifeng Ni, and Sayan Mitra

University of Illinois at Urbana-Champaign
Email: {chsieh16,sibai2,hdt2,yifengn2,mitras}@illinois.edu

***Abstract*— The key concept for safe and efficient traffic management for Unmanned Aircraft Systems (UAS) is the notion of *operation volume* (OV). An OV is a 4-dimensional block of airspace and time, which can express an aircraft's *intent*, and can be used for planning, de-confliction, and traffic management. While there are several high-level simulators for UAS Traffic Management (UTM), we are lacking a framework for creating, manipulating, and reasoning about OVs for heterogeneous air vehicles. In this paper, we address this and present SkyTrakx—a software toolkit for simulation and verification of UTM scenarios based on OVs. First, we illustrate a use case of SkyTrakx by presenting a specific air traffic coordination protocol. This protocol communicates OVs between participating aircraft and an airspace manager for traffic routing. We show how existing formal verification tools, *Dafny* and *Dione*, can assist in automatically checking key properties of the protocol. Second, we show how the OVs can be computed for heterogeneous air vehicles like quadcopters and fixed-wing aircraft using another verification technique, namely *reachability analysis*. Finally, we show that SkyTrakx can be used to simulate complex scenarios involving heterogeneous vehicles, for testing and performance evaluation in terms of workload and response delays analysis. Our experiments delineate the trade-off between performance and workload across different strategies for generating OVs.**

I. Introduction

Unmanned Aircraft Traffic Management (UTM) is an ecosystem of technologies that aim to enable unmanned, autonomous and human-operated, air vehicles to be used for transportation, delivery, and surveillance. By 2024, 1.48 million recreational and 828 thousand commercial unmanned aircraft are expected to be flying in the US national airspace [1]. Unlike the commercial airspace, this emerging area will have to accommodate heterogeneous and innovative vehicles relying on real-time distributed coordination, federated enforcement of regulations, and lightweight training for safety. NASA, FAA, and a number of corporations are vigorously developing various UTM concepts, use cases, information architectures, and protocols towards the envisioned future where a large number of autonomous air vehicles can safely operate beyond visual line-of-sight.

FAA's UTM ConOps [2] defines the basic principles for safe coordination in UTM and the roles and responsibilities for the different parties involved such as the vehicle operator, manufacturer, the airspace service provider, and the FAA. The building-block concept in UTM is the notion of *operation volumes (OVs)* which are used to share *intent information* that, in turn, enables interactive planning and strategic de-confliction for multiple UAS [2]. Roughly, OVs are 4D blocks of airspace with time intervals. They are used to specify the space that UAS is allowed to occupy over an interval of time (see Figures 1 and 2). While there have been small-scale field tests for UTM protocols using OVs [3], there remains a strong need for a general-purpose framework for simulating and verifying UTM protocols based on OVs. Such a framework will need to (i) manipulate and communicate OVs for traffic management protocols, (ii) reason about dynamic OVs for establishing safety of the protocols, (iii) compute OVs for heterogeneous air vehicles performing different maneuvers, and (iv) evaluate UTM protocols in different simulation environments.

In this paper, we address this need and present *SkyTrakx—an open source toolkit for simulation and verification of UTM scenarios*. The toolkit offers a framework that (i) provides automata theory-based APIs for designing UTM protocols that formalize the communication of OVs, (ii) integrates existing tools, Dafny and Dione, to assist in verifying the safety and liveness of the protocols, (iii) uses the reachability analysis tool DryVR to compute OVs for heterogeneous air vehicles, and (iv) expands the ROS and Gazebo-based CyPhyHouse framework [4] to simulate and evaluate configurable UTM scenarios. Benefit from [4], the protocols can be ported from simulations to hardware implementations. The detailed contributions of SkyTrakx are as follows:

Provably safe De-conflicting using OVs: For the first time, we show how the intention expressed as OVs can ensure provably safe distributed de-conflicting in Sections III and IV. As an example, we develop an automata-based de-conflicting protocol using SkyTrakx APIs. This protocol specifies how the participating agents, the air vehicles, should interact with the Airspace Manager (*AM*). We then formally verify the safety and liveness of this protocol. In general, verification of distributed algorithms is challenging, but our safety analysis shows that the use of OVs helps decompose the global de-conflicting of the UAS into local invariant on the *AM* and local real-time requirements on each agent. We further show that Dione [5], a proof assistant for Input/Output Automata (IOA) built with the Dafny program analyzer [6], can prove the local invariant on the *AM* automatically. We prove that the safety of the protocol is achieved when individual agents follow their declared OVs. The liveness analysis further shows that every agent can eventually find a non-conflicting OV, under a stricter set of assumptions.

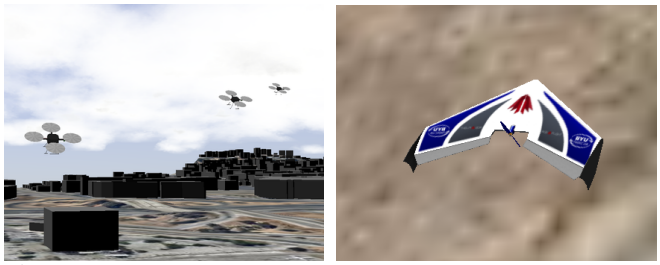


Fig. 1: Hector Quadrotor [8] (*Left*) and ROSplane [9] (*Right*) models in Gazebo simulator.

Reachability Analysis for OV Conformance: The guarantees of our protocol rely on the assumption that the agents do not violate their declared OVs. In Section V, we show how to use an existing data-driven reachability analysis tool, DryVR [7], to create OVs for heterogeneous air vehicles with low violation probability. We apply such analysis on a quadrotor model, Hector Quadrotor [8], and a fixed-wing aircraft model, ROSplane [9], and incorporate them in SkyTrakx. We show both air vehicles in Figure 1 and visualize their OVs for a landing scenario in Figure 2.

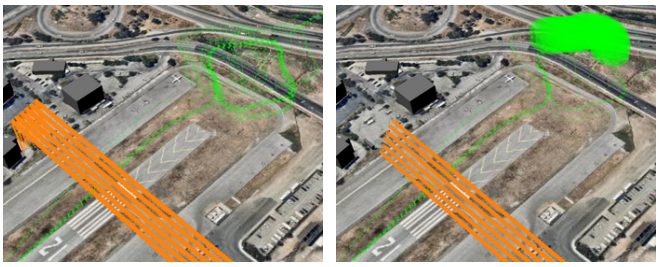
Performance Evaluation: In Section VI, we first discuss the implementation of SkyTrakx. Then, we perform a detailed empirical analysis of our protocol in a number of representative scenarios using SkyTrakx. We compare two strategies for the generation of OVs with different aggressiveness, namely CONSERVATIVE and AGGRESSIVE. Our experiments quantify the performance and workload on the *AM*, and we measure these metrics with respect to the number of participating agents and different strategies for generating OVs. Our results suggest that the workload on the *AM* scales linearly with the number of agents, and AGGRESSIVE provides 1.5-3X speedup but leads to 2-5X increased workload on the *AM*.

II. RELATED WORK

Collision Avoidance Protocols: Prior to the development of the UTM ecosystem, traffic management protocols for manned aircraft include the family of Traffic Alert and Collision Avoidance Systems (TCAS) [10]–[15]. UTM and TCAS are complementary—the former is for long range strategic safety against loss of separation with other aircraft and static obstacles, weather events, and anomalous behaviors, while the latter is for shorter-range tactical safety. Accordingly the protocol we discuss (in Section IV) coordinates over longer range and not *only* for potential collision avoidance. SkyTrakx could be augmented with existing collision avoidance protocols in the future. For instance, if an aircraft violates its OV in our protocol, then a TCAS-like protocol can be used to avoid collision.

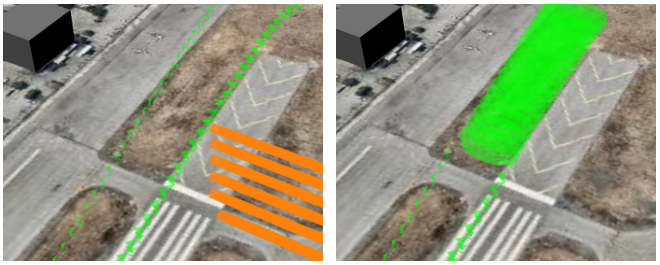
Formal Approaches to UTM and Collision Avoidance: The formal methods' research community has engaged with the problem of air-traffic management in a number of different ways. There have been several works on formal analysis of TCAS [16]–[19], ACAS X [20]–[22], and other protocols [23], [24], [24]–[28].[1] These verification efforts

[1] https://ti.arc.nasa.gov/news/acasx-verification-software/



(a) ROSplane reserved OVs for loitering and descending.

(b) ROSplane loiters and waits for Quadrotors.



(c) Quadrotors passed the runway before ROSplane descends.

(d) ROSplane descends.

Fig. 2: Visualization of a landing scenario with heterogeneous air vehicles in an airport. The OVs for Hector Quadrotors are annotated with orange and OVs for the ROSplane are shown in green. Reserved OVs are outlined with dots, and OVs in use are represented with solid tubes.

rely on various simplifying assumptions such as precise state estimates, straight-line trajectories, constant velocity of the intruder and ownership. Algorithms to synthesize safe-by-construction plans for multiple drones flying in a shared airspace have been developed in [4], [29]–[31]. These approaches rely on predicting and communicating future behavior of participating aircraft under different sources of uncertainty [25], [29], [30].

In [32], the authors present an approach for decentralized policy synthesis for route planning of individual vehicles modeled as Markov decision processes. Our approach decouples the low-level dynamically feasible planning from the distributed coordination, and solves the latter problem using a centralized coordinator (Airspace Manager) via distributed mutual exclusion over regions of the airspace (Section IV). In [33], the authors present a framework for decentralized controller synthesis for different managers of neighboring airspaces. They use finite game and assume-guarantee approaches to generate decision-making mechanisms that satisfy linear temporal logic specifications. An application of their approach is to design policies for airspace managers that enforce a maximum number of vehicles in the airspace or maximum loitering time. Their framework assumes the operating regions for actions such as takeoff or loitering are predefined. Our framework is complementary to this work as we show how a vehicle can generate an OV based on its vehicle dynamics from infinite choices of regions and time.

III. A FORMAL MODEL OF OPERATION VOLUMES

In this section, we formalize the notion of OVs described in [2] which is the fundamental building block for UTM pro-

tocols. This formalization is also implemented in SkyTrakx for creating, manipulating, and reasoning about OVs. We refer to a UAS participating in the UTM system as an *agent*, or equivalently, an *air vehicle*. Every agent in the system has a unique identifier. The set of all possible identifiers is ID. We assume that each agent has access to a common global clock which takes non-negative real numbers. The *airspace* is modeled as a compact subset $\mathcal{X} \subseteq \mathbb{R}^3$. Large airspaces may have to be divided into several smaller airspaces, and one has to deal with hand-off across airspaces. In this paper, we do not handle this problem of air vehicles entering and leaving $\mathcal{X}$. Other works have synthesized safe protocols for this problem (e.g. [33]). The airspace is different from the state space of individual air vehicles which may have many other state components like velocity, acceleration, pitch and yaw angles, etc. Informally, an OV is a schedule for an air vehicle for occupying airspace.

Definition 1: An *operating volume (OV)* is a finite sequence of pairs $C = (R_1, T_1), (R_2, T_2), \ldots, (R_k, T_k)$ where each $R_i \subseteq \mathcal{X}$ is a compact subset of the airspace, and T_i's is a monotonically increasing sequence of time points.

The total *time duration* $T_k - T_1$ of the OV C is denoted by $C.dur$, and the length k of C is denoted by $C.len$. Further, we denote the last time point T_k by $C.T_{last}$, the last region R_k by $C.R_{last}$, and the union of all regions, $\bigcup_{i=1}^{k} R_i$, by $C.R_{all}$ as shorthands. We denote the set of all possible contracts as $\mathbf{OV}$. An air vehicle meets an OV at real-time t if (1) $t \in [T_i, T_{i+1})$ for any $i < k$ implies that the air vehicle is located within R_i, and (2) $t \geq T_k$ implies that the agent is located within R_k *ever after* T_k.

Definition 2: Two OVs are *time-aligned* if they use the same sequence of time points. Given two time-aligned OVs, $C^a = (R_1^a, T_1), \ldots, (R_k^a, T_k)$ and $C^b = (R_1^b, T_1), \ldots, (R_k^b, T_k)$, and a set operation $\oplus \in \{\cap, \cup, \setminus\}$, we define

$$C^a \oplus C^b \triangleq (R_1^a \oplus R_1^b, T_1), \ldots, (R_k^a \oplus R_k^b, T_k).$$

We can generalize the definition to OVs that are not *time-aligned*, and the detailed derivation is provided in the extended version of this paper [34].

Several concepts are defined naturally as set operations on OVs. We abuse notation sometimes and use C as the set represented by contract C, i.e. the set

$$C \triangleq \bigcup_{i=1}^{k-1} \{(r,t) \mid r \in R_i \wedge T_i \leq t < T_{i+1}\} \cup \{(r,t) \mid r \in R_k \wedge T_k \leq t\}.$$

For example, checking if C^a *refines* C^b is to simply check if C^a uses less space-time than C^b does, i.e., $C^a \subseteq C^b$, or equivalently $C^a \setminus C^b = \emptyset$.

We will use the defined operations in our protocol in Section IV to update OVs of individual agents and check intersections. We will show how to create such OVs using reachability analysis in Section V.

IV. A Simple Coordination Protocol using OVs

We present an example protocol for safe traffic management using OVs and its correctness argument. We further implement the protocol with SkyTrakx. The protocol involves a set of agents communicating OVs with an *airspace manager or controller (AM)*. The overall system is the composition of the airspace manager (*AM*) and all agents ($agent_i$):

$$Sys \triangleq AM || \{agent_i\}_{i \in ID}.$$

In Section IV-A and IV-B, we describe the protocol by showing the interaction between participating agents and the *AM* through `request`, `reply`, and `release` messages. We then analyze the safety of the protocol under instant message delivery in Section IV-C, and its liveness in Section IV-D.

A. Airspace Manager

We design the *AM* as an Input/Output Automaton (IOA) [35] defined in Figure 3. The *AM* keeps track of all contracts and checks for conflicts before approving new contracts. It uses a mapping `contr_arr` in which `contr_arr[i]` records the contract held by agent i, and a set `reply_set` to store the IDs of the agents whose requests are being processed and pending reply.

Whenever the *AM* receives a $\texttt{request}_i(\texttt{contr})$ from agent i (line 11), i is first added to `reply_set`. Then, `contr` is checked against all contracts of other agents by checking disjointness (line 14). Only if the check succeeds, `contr` is included in `contr_arr[i]` via set union (line 15).

When i is in `reply_set`, the $\texttt{reply}_i(\texttt{contr})$ action is triggered to reply to agent i with the recorded `contr=contr_arr[i]` (line 7). Note that the *AM* replies with the *recorded contract* `contr_arr[i]` at line 7 irrespective of whether the *requested contract* `contr` in line 11 was included in `contr_arr[i]` or not. Finally, if the *AM* receives a $\texttt{release}_i(\texttt{contr})$, then it removes `contr` from `contr_arr[i]` via set difference (line 18).

B. Agent's Protocol

The agent's coordination protocol sits in between a *planner/navigator* that proposes OVs and a *controller* which drives the air vehicle to its target. We will discuss approaches to estimate OVs for waypoint-based path planners and waypoint-following controllers in Section V. Figure 4

```
1  automaton AirspaceManager
2
3  variables:
4      contr_arr: [ID → OV]
5      reply_set: Set⟨ID⟩
6
7  output reply_i(contr: OV = contr_arr[i])
8      pre: i ∈ reply_set
9      eff: reply_set := reply_set \ {i}
10
11 input request_i(contr: OV)
12     eff:
13         reply_set := reply_set ∪ {i}
14         if ⋀_{j≠i, j∈ID} (contr ∩ contr_arr[j] = ∅):
15             contr_arr[i] := contr_arr[i]∪contr
16
17 input release_i(contr: OV)
18     eff: contr_arr[i] := contr_arr[i] \ contr
```

Fig. 3: Airspace Manager automaton. A model in Dione language [5] with automated invariant checking for IOA is available in [34].

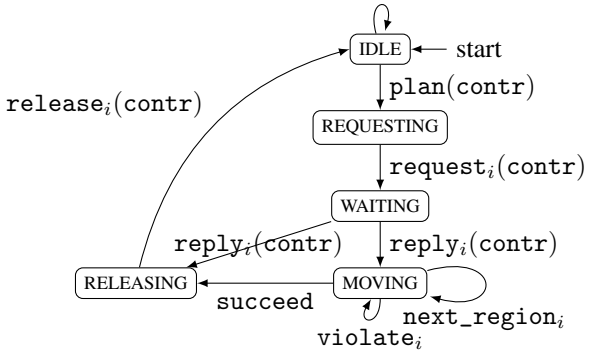


Fig. 4: Simplified state diagram for Agent.

shows the simplified state diagram of the agent protocol. At a high level, agent i's protocol starts in the idle state and initiates when a `plan` action with a given `contr` is triggered by the agent's planner. Then, the protocol requests this contract from the *AM*, and waits for the reply. If the requested contract is a subset of the one replied by the *AM*, the agent protocol enters the moving state. At this point, the agent's controller starts moving the air vehicle and ideally making it follow the contract strictly. Once the air vehicle reaches the last region of OV successfully, the protocol releases the unnecessary portion of the contract and goes back to idle state. In the case that the requested contract is not a subset of the one replied by the *AM*, the protocol directly releases and retries. If the agent violates the contract while moving, it notifies the *AM* that the contract is violated. We provide the formally specified automaton and detail explanation of agent's protocol in the extended version [34].

C. Protocol Correctness: Safety

We now discuss the safety property ensured by our protocol. Here, $agent_i$.`curr_contr` denotes the contract that the i^{th} agent is following. Assuming that none of the agents triggered their `violate` action, then an agent always follows its local contract `curr_contr`. In that case, collision avoidance is defined naturally as the disjointness between the `curr_contr`s of all agents. Our goal therefore is to show that the following proposition is an invariant of the system:

Proposition 1 (Safety): If none of the agents triggered their `violate` action, the current contracts followed by all agents are pairwise disjoint, i.e.,

$$\bigwedge_{i\in ID}\bigwedge_{j\neq i, j\in ID} agent_i.\texttt{curr\_contr} \cap agent_j.\texttt{curr\_contr} = \emptyset.$$

Our proof strategy is to show that first the global record of contracts maintained by the *AM* are pairwise disjoint by Lemma 1. Then, we ensure the local copy by each agent is as restrictive as the global record and hence preserves disjointness by Lemma 2. With Lemma 1 and Lemma 2, Proposition 1 is derived following basic set theory. We start from Lemma 1 for the *AM*.

Lemma 1: If none of the agents triggered their `violate` action, all contracts recorded by the *AM* are pairwise disjoint, i.e.,

$$\bigwedge_{i\in ID}\bigwedge_{j\neq i, j\in ID} AM.\texttt{contr\_arr}[i] \cap AM.\texttt{contr\_arr}[j] = \emptyset.$$

Proof: This is a direct result from examining all actions of the *AM* automaton. The `request`$_i$ action ensures that a `contr` is only included into `contr_arr`[`i`] if it is disjoint from all other contracts `contr_arr`[`j`]. The `reply`$_i$ action does not modify `contr_arr` at all, and `release`$_i$ action only shrinks the contracts. ■

Lemma 2: If none of the agents triggered their `violate` action, the `curr_contr` of agent i is always as restrictive as `contr_arr`[`i`], i.e.,

$$\bigwedge_{i\in ID} agent_i.\texttt{curr\_contr} \subseteq AM.\texttt{contr\_arr}[i].$$

Proof: This is proven by examining all actions of agent automaton regardless of the order of execution. Due to the space limit, we only consider when actions are delivered instantaneously. The `curr_contr` is only modified in `reply` and `release` actions. In `reply` action, `curr_contr` is to copy `contr` sent by the *AM* and thus Lemma 2 holds. In `release` action, `curr_contr` removes `contr` first; then `release` is delivered to the *AM* to remove `contr`. As a result, Lemma 2 still holds. In [34], we extend the proof so that, even under delayed communication settings, the lemma still holds when the order of received messages is preserved. ■

D. Protocol Correctness: Liveness

For liveness property, we would like to see every agent eventually reaches its target. In our protocol, this is formulated as every agent eventually reaches the last region of its OV that it proposed in `plan` action and triggers its `succeed` action. The overall proof is to show that an agent can always find an OV which the *AM* approves.

Since a newly proposed OV may be rejected, we denote it as `plan_contr` to distinguish from `curr_contr` which an agent always follows. It is worth noting that liveness depends on the OV for each agent. A simple scenario where liveness cannot be achieved is when the final destinations of two agents are too close; thus the last region where one agent stays at the end could block the other agent forever. Therefore, we first require the following assumption:

Assumption 1 (Disjointness of different agents' regions): For any agent $i \in ID$, all regions that it plans to traverse are disjoint from the last regions of all other agents. Formally,

$$\bigwedge_{j\neq i} \texttt{plan\_contr}_i.R_{all} \cap AM.\texttt{contr\_arr}[j].R_{last} = \emptyset.$$

Assumption 1 can be achieved by querying the *AM* when planning since Lemma 2 ensures the *AM*'s record of OVs includes the agents' OVs.

Definition 3: Given an OV $C = (R_1, T_1), \ldots, (R_k, T_k)$ and a time duration δ, we define $reschedule(C, \delta)$ as:

$$reschedule(C, \delta) \triangleq (R_1, T_1 + \delta), (R_2, T_2 + \delta), \ldots, (R_k, T_k + \delta)$$

Now we start our argument for liveness. By our protocol design, if agent i never violates its OV, it must reach the last region successfully. Therefore, we only have to prove that agent i's request to the *AM* must be accepted eventually. With Assumption 1, we prove the claim that an agent i can always *reschedule* a plan so that the *AM* approves its OV.

Proposition 2 (Liveness): If $\texttt{plan\_contr}_i$ satisfies Assumption 1, then there is a time duration δ_0 such that the *AM* approves $reschedule(\texttt{plan\_contr}_i, \delta)$ for all $\delta \geq \delta_0$. Formally,

$$\bigwedge_{j \neq i, j \in ID} reschedule(\texttt{plan\_contr}_i, \delta) \cap AM.\texttt{contr\_arr}[j] = \emptyset.$$

Proof: Following Assumption 1, we first derive the disjointness of regions of airspace. For any $j \neq i$ and any δ,

$$reschedule(\texttt{plan\_contr}_i, \delta).R_{all} \cap AM.\texttt{contr\_arr}[j].R_{last} = \emptyset, \quad (1)$$

because $reschedule$ does not modify the regions. Further, we derive that any $\delta_j \geq AM.\texttt{contr\_arr}[j].T_{last}$, the following two OVs are disjoint:

$$reschedule(\texttt{plan\_contr}_i, \delta_j) \cap AM.\texttt{contr\_arr}[j] = \emptyset. \quad (2)$$

The proof is to expand the definition and is skipped here. Intuitively, this is because every agent j is expected to reach and stay in $AM.\texttt{contr\_arr}[j].R_{last}$ ever after $\delta_j \geq AM.\texttt{contr\_arr}[j].T_{last}$. Therefore, the rescheduled OV for agent i does not overlap with OVs of any other agent j.

Finally, let $\delta_0 \triangleq \max_{j \neq i} AM.\texttt{contr\_arr}[j].T_{last}$ and it directly leads to the proof of Proposition 2. ■

In addition to the manual proof presented, we have also explored using Dione [5] with Dafny proof assistant [6] to generate induction proof for invariants of IOA. We chose this tool due to its support for IOA and automated SMT solving for set operations on OVs. We discovered that these tools can automatically prove the local invariant Lemma 1 for the *AM*. However, they lack support for continuous time to model agents and communication delay; hence we cannot use Dione to prove other lemmas and propositions directly.

V. Reachability Analysis and Operation Volumes

In Section IV, we show that the protocol ensures safety and liveness. However, the proof assumes that the air vehicle does not violate its OV. In this section, we discuss how to use existing reachability analyses to over-approximate regions of space-time that an air vehicle may visit. This over-approximation can be used to (i) generate OVs that are unlikely to be violated, or (ii) monitor air vehicles at runtime to predict and avoid possible violations.

Formally, given a dynamical system with state space D, a set of initial states $Q_0 \subseteq D$, and a time horizon $[T_0, T_1)$, reachability analysis tools can compute *reachtube*, a set of states Q_1 reachable within $[T_0, T_1)$. We further require a function $\hat{\pi} : \mathbf{P}(D) \mapsto \mathbf{P}(\mathcal{X})$ to transform state space to air-space. Then, one can build an OV $C_{reach} = (-\infty, \hat{\pi}(Q_0)), (T_0, \hat{\pi}(Q_1)), (T_1, \mathcal{X})$. This means that when air vehicle stays within $\hat{\pi}(Q_0)$ before T_0, it will then stay within $\hat{\pi}(Q_1)$ between T_0 and T_1, and it can be anywhere after T_1. We then can merge C_{reach} for different time horizons to propose OVs.

In this work, we use DryVR [7] to compute reachtubes from simulation traces. DryVR uses collected traces to learn the sensitivity of the trajectories of the air vehicle, and generates reachtubes for a new simulation trace with probabilistic guarantees. We use DryVR to generate OVs for a quadcoptor model, Hector Quadrotor [8] and a fixed-wing model, ROSplane [9], using the Gazebo simulator.

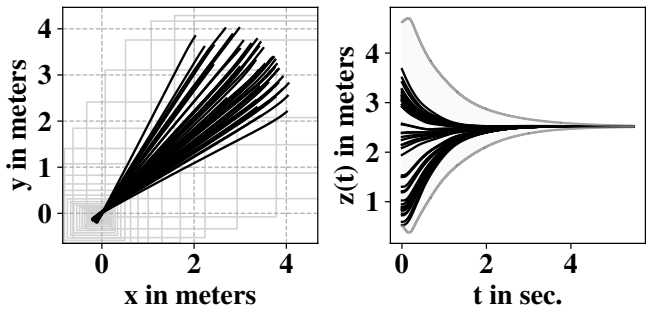


Fig. 5: Simulation traces in *Black* and boundary of the reachtube computed by DryVR in *Gray* for Hector Quadrotor going to the waypoint at (0, 0, 2.5). The reachtube is projected to xy-plane (*Left*) and z-axis over time (*Right*).

a) Hector Quadrotor: The state variables for Hector Quadrotor already include x, y, and z for its position. They also include other variables for orientation and velocity. Hence, $\hat{\pi}$ for this model is to simply apply projections to the x, y, and z axes. We compute C_{reach} for a scenario where the air vehicle follows the waypoint (0, 0, 2.5). Figure 5 shows the projection of C_{reach} as hyper-rectangles to the xy-plane (left) and to the z-axis against time (right). We can generate OVs using a CONSERVATIVE strategy that covers C_{reach} for the entire time horizon with a bounding rectangle, or an AGGRESSIVE strategy to use the gray rectangles as an OV with short time intervals. In general, we can generate a spectrum of OVs from C_{reach} between CONSERVATIVE and AGGRESSIVE strategies, and all OVs in this spectrum can guarantee, using reachability analysis, a low probability of violations. We further explore the performance trade-off between the two strategies in Section VI.

b) ROSplane: Similarly, the state variables for ROSplane include x, y, and z representing its position but in North-East-Down (NED) coordinates. Hence, $\hat{\pi}$ for this model is to apply projections to x, y, and z axes and transform to the coordinates used by the Airspace Manager. We simulate some of its traces and then divide them into segments to analyze several path primitives denoted as modes for ROSplane [9]. In Figure 6, we show the reachtubes for two modes, namely loiter and descend. Unsurprisingly, the plane may not maintain the desired altitude (z-axis) precisely while loitering, and thus it is important to reserve enough range of altitude in OVs for ROSplane.

In summary, we are able to derive useful, i.e., not overly conservative, OVs using DryVR, even with noisy simulations, as shown in Figure 6. The main engineering difficulty we faced using DryVR is to divide traces into proper segments that are from the same mode for ROSplane. This requires domain knowledge on each air vehicle model, for which we refer the readers to [8] and [9].

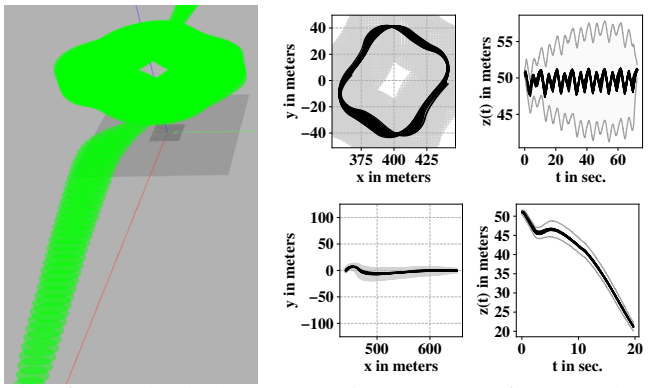


Fig. 6: Reachtube by DryVR in 3D (*Left*) for ROSplane to loiter and then descend. The traces and reachtube for loiter (*Top Row*) and descent (*Bottom Row*) are projected to xy-plane (1^{st} *column*) and z-axis over time (2^{nd} *column*).

VI. SkyTrakx implementation and evaluation

Our experiments are conducted using SkyTrakx. SkyTrakx and all simulation scripts are available at our GitHub repository.[2] To better present our results within the page limits, we only include experiments with the Hector Quadrotor model [8] with its default waypoint-following controller. We first describe SkyTrakx, then the scenarios, then the experimental results followed by a brief discussion.

A. SkyTrakx: System Details

SkyTrakx consists of four major components: (1) Dione verification discussed in Section IV, (2) reachability analysis and reachtubes from DryVR described in Section V, (3) an executable reference UTM protocol implemented in Python of Section IV, and (4) UTM protocol simulation and visualization with CyPhyHouse [4]. Here we focus on the executable UTM protocol and its simulation.

To faithfully follow the semantics of our example UTM protocol, we first provide a data structure to represent and easily manipulate rectangular OVs. We provide APIs for designing executable (timed) input/output automata that can interact with simulated vehicles in CyPhyHouse, and implement an execution engine to simulate the input/output automata alongside CyPhyHouse. To reuse reachtube from DryVR, we also design APIs to load pre-computed reachtubes for estimating OVs. Finally, we also provide several scripts to setup desired scenarios and environments in CyPhyHouse, and implement a plugin to better visualize OVs in the Gazebo simulation backend of CyPhyHouse.

B. Evaluation Scenarios

Following the protocol defined in Section IV, a *scenario* for evaluation is specified by (1) the set of agents *ID* which we consider $\#\mathbf{A} = |ID|$ (2) the world map and the predefined sequence of waypoints for each agent denoted as the *map*, and (3) the strategy that the agents use to generate OVs from their waypoints. For example, the *Left* figure in Figure 7 shows a scenario with $\#\mathbf{A} = 6$ drones in the Corridor map. It uses the Aggressive strategy to generate OVs, which are visualized in the red and blue frames.

We evaluate our protocol in the following maps shown in Figure 7:

(1) Corridor simulates two sets of drones on the opposite sides of a tight air corridor trying to pass through. This may happen in a garage-like space where a fleet of air vehicles enter or leave.
(2) Loop simulates each drone following the vertices of the same closed polygonal chain. This models common segments in the routes of air vehicles such as pickup packages or return to bases' routes.
(3) CitySim is a more realistic scenario which simulates drones flying in a city block.
(4) Random$N$ are scenarios where each drone follows a sequence of N random waypoints inside a $25m \times 25m$ arena. This is to validate our protocol via random testing.

In addition, a designated landing spot for each drone is specified as the last waypoint in all maps to ensure the liveness property. This avoids the situation where a landed drone blocks other air vehicles.

Conservative *and* Aggressive *OVs:* We implemented two strategies, namely Conservative and Aggressive, to generate OVs from given waypoints and positions. Both strategies are deterministic and use only *hyper-rectangles* for specifying regions in OVs. As discussed in Section V, Conservative reserves large rectangles covering consecutive waypoints with longer durations between time points. Thus, it acquires unnecessarily large volumes and may obstruct other agents. In contrast, Aggressive heuristically selects smaller rectangles and shorter durations. Therefore, Aggressive is less likely to block other agents but increases the workload of the *AM* because the OVs (numbers of rectangles) are more complex.

C. Experimental Results

Setup: Our simulation experiments were conducted on a machine with 4 CPUs at 3.40GHz, 8GB memory, and an Nvidia GeForce GTX 1060 3GB video card. The software platform is Ubuntu 16.04 LTS with ROS Kinetic and Gazebo 9. For the time usage, we report the simulation time from Gazebo (time elapsed in the simulated world), instead of wall clock time to help reduce the variations in the results due to irrelevant workload on our machine. To address the nondeterminism arising from concurrency in simulating multiple agents, we simulate each scenario three times, and report the average value of each metric.

Response Time and Workload: Figure 8 shows the response time for each drone starting from sending the first request to finish traversing all waypoints using the Conservative strategy in the Corridor, Loop, and Random$N$ maps. As expected, the maximum response time per agent grows linearly against the number of participating agents because, in the worst case, all agents are accessing the shared narrow air-corridor, and the last agent has to wait until all other agents finish. The average response time shows that it is possible to finish faster if agents can execute concurrently

[2] https://github.com/cyphyhouse/CyPhyHouseExperiments

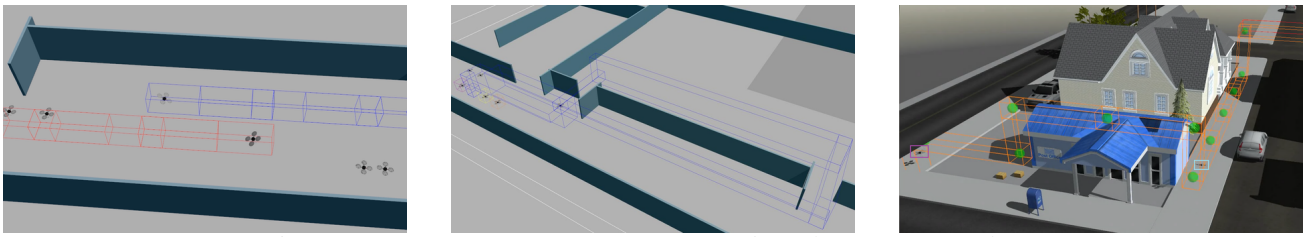


Fig. 7: Maps: CORRIDOR (*Left*), LOOP (*Mid*), and CITYSIM (*Right*)

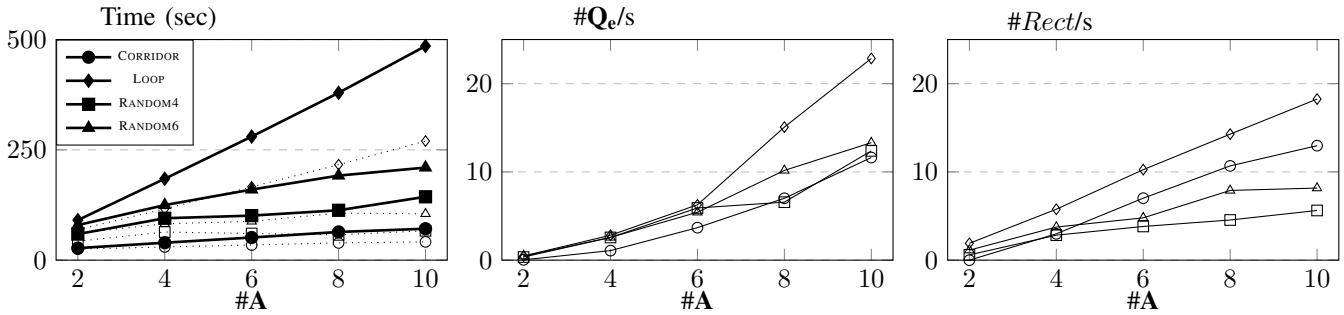


Fig. 8: Response time per agent (*Left*), #emptiness queries per second (*Mid*), and #rectangles checked by the *AM* per second (*Right*) for each map using CONSERVATIVE strategy. Max is in *Solid marks and lines* and Avg. is in *Hollow marks and dotted lines*

TABLE I: Comparison of simulation time between CONSERVATIVE and AGGRESSIVE. #**A** is the number of agents, Time(s) is the total time for simulation according to *the simulated clock* in seconds, #*Rect*/s is the number of rectangles per second in the disjointness query of OVs by the *AM*.

Map	#**A**	CONSERVATIVE		AGGRESSIVE		Speedup	Increased
		Time(s)	#*Rect*/s	Time(s)	#*Rect*/s		#*Rect*/s
CORRIDOR	2	27.52	0.00	21.30	0.00	1.29X	N/A
	4	39.78	2.99	27.24	6.16	1.46X	2.71X
	6	51.63	7.02	34.14	14.10	1.51X	2.06X
	8	64.18	10.68	37.91	22.13	1.69X	2.01X
	10	95.47	12.97	41.94	35.14	2.28X	2.07X
LOOP	2	91.05	1.91	37.63	6.85	2.42X	3.59X
	4	184.88	5.77	70.89	23.33	2.61X	4.04X
	6	280.51	10.26	103.28	40.52	2.72X	3.95X
	8	379.53	14.28	134.62	63.71	2.82X	4.46X
	10	485.58	18.26	169.25	90.94	2.87X	4.98X
CITYSIM	2	77.42	1.77	49.92	4.48	1.55X	2.53X

in disjoint airspaces. For example, the average time for 10 agents is smaller than the time for 8 agents in RANDOM6.

In Figure 8, we consider the number of emptiness/disjointness queries (denoted as #$\mathbf{Q_e}$) and of hyper-rectangles to check (denoted as #*Rect*) per second for the *AM*. #*Rect* provides a finer estimation of computation resources needed by the *AM* than #$\mathbf{Q_e}$. The growth of #$\mathbf{Q_e}$ as expected is roughly quadratic against #**A** in the worst scenario due to checking pairwise disjointness. However, the growth of #*Rect* is not as fast and is seemingly linear to #**A** in the worst scenario. Therefore, it is very likely that the workload increases only linearly instead of quadratically when we use a simple representation of OVs such as hyper-rectangles.

CONSERVATIVE *vs.* AGGRESSIVE.*:* We compare the time between the CONSERVATIVE and AGGRESSIVE strategies in the CORRIDOR, LOOP, and CITYSIM maps. Due to the heavier demand for computational resources required, we only simulated two drones in CITYSIM. Table I shows that the AGGRESSIVE strategy can reduce the overall response time and provides a 1.3-2.8X speedup with larger number of participating agents. This experiment shows that our framework is suitable for comparing and quantifying the trade-offs between performance, safety, and workload under different OVs' generation strategies.

VII. DISCUSSIONS AND CONCLUSIONS

There is a strong need for a toolkit for formal safety analysis and larger scale empirical evaluations of different UTM concepts and protocols. In this paper, we present SkyTrakx, a toolkit with an executable formal model of UTM operations and study its safety, scalability, and performance.

Our toolkit SkyTrakx offers open and flexible reference implementation of a UTM coordination protocol using ROS and Gazebo. Our formal analyses in SkyTrakx illustrate how formal reasoning can be applied to the family of UTM deconflicting protocols. We discovered the capability but also the lack of features of Dione [5] and Dafny [6] for providing automated proofs, and to our knowledge, there is no other proof assistant for IOA that also supports the modeling of OVs. We further studied the connection between OVs and reachabilty analysis, and we showcased how to use DryVR to over-approximate the reachable regions of airspace using simulation traces. The simulator also makes it possible to study different strategies for reserving OVs.

Some of the simplifying assumptions made can be removed with careful engineering, while others require brand

new ideas. Handling timing and positioning inaccuracies and heterogeneous vehicles fall in the former category. We have partly addressed this category using existing reachability analyses in Section V. In the latter category, a major concern is when there are unavoidable violations of OVs due to, for example, hardware failures. Possible solutions include integration with existing predictive failure detection or failure mitigation strategies and collision avoidance protocols, incorporation of human operators, or generation of notifications to other participating agents for collision avoidance. Finally, an important extension is the design of a coordination protocol for multiple airspace managers having the same guarantees.

ACKNOWLEDGMENT

The authors were supported in part by research grants from the National Science Foundation (CyPhyHouse: 1629949 and FMitF: 1918531) and The Boeing Company. We thank John L. Olson, Aaron A. Mayne, and Michael R. Abraham from The Boeing Company for valuable technical discussions.

REFERENCES

[1] Federal Aviation Administration. (2020) FAA Aerospace Forecast Fiscal Year 2020-2040. [Online]. Available: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2020-40_FAA_Aerospace_Forecast.pdf

[2] ——. (2020, Mar.) Unmanned Aircraft System Traffic Management (UTM) Concept of Operations Version 2.0. [Online]. Available: https://www.faa.gov/uas/research_development/traffic_management/media/UTM_ConOps_v2.pdf

[3] ——. (2019, Oct.) UTM Pilot Program (UPP) Summary Report. [Online]. Available: https://www.faa.gov/uas/research_development/traffic_management/utm_pilot_program/media/UPP_Technical_Summary_Report_Final.pdf

[4] R. Ghosh, J. P. Jansch-Porto, C. Hsieh, A. Gosse, M. Jiang, H. Taylor, P. Du, S. Mitra, and G. Dullerud, "Cyphyhouse: A programming, simulation, and deployment toolchain for heterogeneous distributed coordination," 10 2019.

[5] C. Hsieh and S. Mitra, "Dione: A protocol verification system built with dafny for i/o automata," in *Integrated Formal Methods*, W. Ahrendt and S. L. Tapia Tarifa, Eds. Cham: Springer International Publishing, 2019, pp. 227–245.

[6] K. R. M. Leino, "Dafny: An Automatic Program Verifier for Functional Correctness," in *LPAR'10*, ser. LNCS. Springer Berlin Heidelberg, 2010, pp. 348–370.

[7] C. Fan, B. Qi, S. Mitra, and M. Viswanathan, "Dryvr: Data-driven verification and compositional reasoning for automotive systems," in *Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017)*, 2017, pp. 441–461.

[8] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *Simulation, Modeling, and Programming for Autonomous Robots*, I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 400–411.

[9] G. Ellingson and T. McLain, "Rosplane: Fixed-wing autopilot for education and research," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017, pp. 1503–1507.

[10] Federal Aviation Administration. (2011, Feb.) Introduction to TCAS II version 7.1. [Online]. Available: https://www.faa.gov/documentlibrary/media/advisory_circular/tcas%20ii%20v7.1%20intro%20booklet.pdf

[11] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos, "Next-generation airborne collision avoidance system," Massachusetts Institute of Technology Lincoln Laboratory, Tech. Rep., 2012.

[12] M. J. Kochenderfer, C. Amato, G. Chowdhary, J. P. How, H. J. D. Reynolds, J. R. Thornton, P. A. Torres-Carrasquillo, N. K. Ure, and J. Vian, *Optimized Airborne Collision Avoidance*, 2015, pp. 249–276.

[13] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–10.

[14] J. K. Kuchar and L. C. Yang, "A review of conflict detection and resolution modeling methods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, pp. 179–189, 2000.

[15] X. Yu and Y. Zhang, "Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects," *Progress in Aerospace Sciences*, vol. 74, pp. 152 – 166, 2015.

[16] J. Lygeros and N. Lynch, "On the formal verification of the tcas conflict resolution algorithms," in *Proceedings of the 36th IEEE Conference on Decision and Control*, vol. 2, 1997, pp. 1829–1834.

[17] C. Livadas, J. Lygeros, and N. A. Lynch, "High-level modeling and analysis of TCAS," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99),Phoenix, Arizona*, 1999, pp. 115–125.

[18] N. Lynch, "High-level modeling and analysis of an air-traffic management system," in *Hybrid Systems: Computation and Control*. Springer Berlin Heidelberg, 1999, p. 3.

[19] C. Livadas, J. Lygeros, and N. Lynch, "High-level modeling and analysis of the traffic alert and collision avoidance system (tcas)," *Proceedings of the IEEE*, vol. 88, pp. 926–948, 2000.

[20] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "A formally verified hybrid system for the next-generation airborne collision avoidance system," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, 2015, pp. 21–36.

[21] J. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "Formal verification of acas x, an industrial airborne collision avoidance system," in *2015 International Conference on Embedded Software (EMSOFT)*, 2015, pp. 127–136.

[22] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, pp. 97–117.

[23] T. Johnson and S. Mitra, "A small model theorem for rectangular hybrid automata networks," 2012.

[24] P. S. Duggirala, L. Wang, S. Mitra, C. Munoz, and M. Viswanathan, "Temporal precedence checking for switched models and its application to a parallel landing protocol," in *International Conference on Formal Methods (FM 2014), Singapore*, 2014.

[25] H.-D. Tran, L. V. Nguyen, P. Musau, W. Xiang, and T. T. Johnson, "Decentralized real-time safety verification for distributed cyber-physical systems," in *Formal Techniques for Distributed Objects, Components, and Systems*, J. A. Pérez and N. Yoshida, Eds. Cham: Springer International Publishing, 2019, pp. 261–277.

[26] M. Webster, M. Fisher, N. Cameron, and M. Jump, "Formal methods for the certification of autonomous unmanned aircraft systems," in *Computer Safety, Reliability, and Security*, F. Flammini, S. Bologna, and V. Vittorini, Eds. Springer Berlin Heidelberg, 2011, pp. 228–242.

[27] O. McAree, J. M. Aitken, and S. M. Veres, "A model based design framework for safety verification of a semi-autonomous inspection drone," in *2016 UKACC 11th International Conference on Control (CONTROL)*, 2016, pp. 1–6.

[28] S. Umeno and N. A. Lynch, "Safety verification of an aircraft landing protocol: A refinement approach." in *HSCC 2007*, 2007, pp. 557–572.

[29] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-logic: Control of multi-drone fleets with temporal logic objectives," in *ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[30] A. Desai, I. Saha, J. Yang, S. Qadeer, and S. A. Seshia, "Drona: A framework for safe distributed mobile robotics," in *Proceedings of the 8th International Conference on Cyber-Physical Systems*, ser. ICCPS '17. ACM, 2017, p. 239–248.

[31] T. Schouwenaars, "Safe trajectory planning of autonomous vehicles," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[32] S. Bharadwaj, S. Carr, N. Neogi, H. Poonawala, A. B. Chueca, and U. Topcu, "Traffic management for urban air mobility," in *NASA Formal Methods Symposium*. Springer, 2019, pp. 71–87.

[33] S. Bharadwaj, S. P. Carr, N. A. Neogi, and U. Topcu, "Decentralized control synthesis for air traffic management in urban air mobility," *IEEE Transactions on Control of Network Systems*, pp. 1–1, 2021.

[34] C. Hsieh, H. Sibai, H. Taylor, Y. Ni, and S. Mitra, "Skytrakx: A Toolkit for Simulation and Verification of Unmanned Air-Traffic Management Systems (extended version)," 2020. [Online]. Available: https://arxiv.org/abs/2009.04655

[35] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.