Decentralized Control Synthesis for Air Traffic Management in Urban Air Mobility

Suda Bharadwaj, Steven Carr, Natasha Neogi and Ufuk Topcu

Abstract— Urban air mobility (UAM) refers to air transportation services within an urban area, often in an on-demand fashion. We study air traffic management (ATM) for vehicles in a UAM fleet, while guaranteeing system safety requirements such as traffic separation. Existing ATM methods for unmanned aerial systems such as UTM utilize alternative approaches which do not provide strict safety guarantees. No established infrastructure exists for providing ATM at scale for UAM. We provide a decentralized, hierarchical approach for UAM ATM that allows for scalability to high traffic densities as well as providing theoretical guarantees of correctness with respect to user provided safety specifications. Our main contributions are two-fold. First, we propose a novel UAM ATM architecture that divides control authority between vertihubs that are each in charge of all UAM vehicles in their local airspace. Each vertihub also contains a number of vertiports that are in charge of UAM vehicle takeoffs and landings. The resulting architecture is decentralized and hierarchical, which not only enables scalability, but also robustness in the event of any individual vertihub or vertiport no longer being operational. Second, we provide a contract-based correct-by-construction reactive synthesis approach that provably guarantees safety properties with respect to user-provided safety specifications in linear temporal logic. We demonstrate the approach on large-volume UAM air traffic data.

Index Terms— Air traffic management, reactive synthesis, system safety.

I. Introduction

A. Urban air mobility setting

Currently, major metropolitan areas endure pressure on their transportation infrastructure, manifesting as traffic jams or commuter delays which can negatively impact productivity [19]. As population and congestion increase in these urban and suburban areas, mobility challenges are expected to intensify. One proposed solution is the introduction of urban air mobility (UAM) which refers to urban air services to carry passengers and cargo across metropolitan areas. Recent advances in electric vertical take-off and landing (eVTOL) aircraft have the potential to revolutionize UAM and make it commercially feasible in the near future [1].

Manuscript submitted on May 12, 2021

This work was partially supported by AFRL FA9550-19-1-0169, AFOSR FA9550-19-1-0005, DARPA D19AP00004.

S. Bharadwaj, S. Carr and U. Topcu are at the Department of Aerospace Engineering and Engineering Mechanics at the University of Texas at Austin, email: {suda.b,stevencarr,utopcu}@utexas.edu

N. Neogi is at NASA-Langley Research Center, Hampton, VA, USA, email: natasha.a.neogi@nasa.gov

UAM can not only help cities from an economic standpoint by allowing for faster movement of goods and people, but also has the potential to add to the public good by allowing for expedited public health services like air ambulances. However, establishing a framework, which allows for safe, orderly, and efficient flights in what will be a complex, high-traffic environment with competing requirements and priorities, remains crucial for UAM to be practically realized.

In this paper, we develop a method for scalable air traffic management (ATM) for UAM with provable guarantees of safety properties.

B. Challenges in air traffic management for UAM

UAM presents challenges that cannot yet be handled in existing ATM approaches. Current and next generation ATM services, as described in [12], [34], are designed to manage scheduled flights between established airports located in or near cities separated by a significant distance and occurring at conventional flight altitudes (e.g., above 10 000 ft). UAM will require management for on-demand, high-volume, short-range flights in close proximity to urban airspace (e.g., below 10 000 ft) with increasingly autonomous aircraft. Since these vehicles will be operating in urban airspaces with high traffic densities, they need to be able to operate with smaller separation standards than current ATM services can accommodate (e.g., closely spaced altitude separation). Any traffic management system for UAM will also need to be able to handle unpredictable situations in a safe manner without overly compromising the performance of the entire system.

Currently, there is no established infrastructure for traffic management of a UAM-like environment. A traffic management system for small unmanned aerial systems (UAS) called UTM (UAS Traffic Management) has been proposed, and takes a federated approach to ensuring airspace access [30]. This approach may enable the incorporation of multiple safety oriented services [25] such as aircraft separation [26] and geo-fencing [27]. However, there may be concerns in this approach with respect to scalability, in terms of the size of UAM vehicles as well as UAM traffic density. There is a growing need to explore the design of an ATM system architecture capable of safely and efficiently managing UAM operations.

Guaranteeing global satisfaction of safety properties for UAM operations has the following key challenges

- The setting encompasses multiple service providers and stakeholders, each with potentially competing priorities and requirements. Consequently, the full state of the entire fleet of vehicles is unlikely to be controlled or even observed by a single entity.
- UAM will operate in a complex and diverse airspace environment(e.g., Class G [13], Class B etc.) that must support both conventional operations using legacy aircraft as well as emerging operations. Therefore verifying safety of such an evolving system at design-time is not practical.
- While a centralized solution process allows for easier verification of correctness, the resulting state space explosion entailed in synthesis, especially under the projected traffic demands, makes a centralized solution computationally untenable.

Removing reliance on full state-information for control requires a version of distributed synthesis. However, except for a few restricted classes of architectures, the distributed synthesis problem is undecidable [32]. The decidable versions of the problem lack practical solutions due to their non-elementary complexity [31]. Significant effort in runtime monitoring in this area is focused on providing efficient solutions by exploiting the structure of the system [11], [15] or the specification [4], [17].

C. UAM ATM architecture

We propose a decentralized, hierarchical UAM ATM architecture for provably correct operations. We divide up the responsibilities of an ATM architecture for UAM into two broad classes

1. Pre-flight authorization: receiving flight requests with little notice, identifying a safe route, and authorizing the departure.
2. Dynamic airspace management: managing routes and in-flight aircraft in response to an unpredictable environment stemming from the on-demand trip scheduling.

Pre-departure planning and de-conflicting flight routes before take-off have been studied extensively in the literature [20], [22], [37]. More recently, there have been efforts in applying these works in an on-demand UAM setting [18]. In this work, we primarily focus on the latter case of guaranteeing safety during dynamic flight operations. We will assume the existence of an assured scheduler that is able to give pre-flight authorization for routes given passenger requests.

In our proposed UAM ATM architecture, we leverage the geographical location of infrastructure to divide the region into sectors that are each overseen by vertihubs. Each vertihub controls the flow of vehicles in and out of its sector. Within the purview of a vertihub are several vertiports that control individual vehicle takeoff and landing. Such an architecture is similar to how airspace in the Terminal Radar Approach Control is managed, but is more general in its approach to tackling balkanization.

The UAM setting is unique as most flights will be on-demand and hence will require a controller that can react to an unpredictable environment and provide guarantees of safety and liveness. Reactive synthesis [8] is a natural candidate to produce such controllers. A user (such as a regulatory body) can specify specifications in linear temporal logic for the operations of each vertihub and vertiport in the system. The task is to synthesize controllers for each vertihub and vertiport separately, guaranteeing that, together, the joint operation of the global system satisfies the conjunction of all specifications while still guaranteeing progress for the vehicles. In order to ensure that each controller does not impede the ability of other controllers to satisfy their requirements, we introduce a contract-based synthesis method which we formulate as a Generalized Reactivity(1) (GR(1)) [9] synthesis problem that can be solved efficiently [3], [36]. Hence, our proposed solution architecture is scalable without sacrificing any safety or liveness guarantees.

D. Related work

Some preliminary work is being done in cooperative ATM for next generation air traffic management [29], but this work considers a scheduled approach for large passenger aircraft and cannot handle management for on-demand flights. Similarly, work has been done on distributed control for ATM of small unmanned aerial systems (UAS) [16], but this work relies on cloud based architectures that do not currently satisfy strict aviation safety requirements. Hybrid control approaches have been applied [35], however scalability proves to be an issue.

To the best of our knowledge, this is the first approach to controller synthesis with safety guarantees for large-scale UAM ATM operations. Formally verified tools such as DAIDALUS [26] provide safety guarantees at lower levels of operations, however, they do not handle the fleet-level operations. The most similar approach to the one presented in this paper is runtime enforcement [14], [33] of a specified property, in which a synthesized module detects and alters the behavior of the system in a way that maintains the desired property. An existing approach called shielding [10], [21] uses reactive synthesis and assumes that the shield has full knowledge and control of the whole system — in this case the entire UAM system and the vehicles it handles.

A technique for synthesizing quantitative shields for multi-agent systems in a fully centralized manner was presented in [5]. All these approaches rely on restrictive assumptions on runtime communication (i.e., full network coverage) and the extent of awareness and control authority of the shield (e.g., the shield can affect any agent in the network instantaneously). This requirement was relaxed in [7] where a local shield was synthesized for each sector with contracts between neighbors to guarantee global correctness. However, the approach was formulated only for specific safety properties (e.g., minimum-separation) and not more general properties such as liveness as is done here. We do not consider quantitative properties or optimality of behavior in this work, as the primary focus is the guarantee of specifications.

The work in this paper directly extends [7] by generalizing the class of allowable safety properties to any property in the GR(1) [9] fragment of linear temporal logic. Furthermore, the work in [7] was limited to very specific vehicle behaviors, and could not handle take-off or landing requests. In this work, we introduce vertiport controllers that operate in the sector regions to additionally handle take-off and landing requests. The induced hierarchical structure allows for separation of concerns between the vertihub and vertiport controllers. A decentralized, hierarchical approach for ATM was proposed in [37], but unlike the setting in this paper, cannot handle temporal logic specifications. Hence, we are able to systematically synthesize controllers for ATM that can guarantee complex temporal requirements unlike conventional ATM approaches such as [37].

E. Contributions of the paper

This work is the first that considers a hierarchical, decentralized synthesis procedure for UAM air traffic management. We break down our contributions as follows:

- We design an architecture that allows a user (such as a regulatory body interested in guaranteeing safe operations) to specify safety requirements for the operations at each vertihub and vertiport.
- The architecture allows the controller for each vertihub and vertiport to be synthesized separately, hence avoiding the state-space explosion of centralized synthesis.
- We use contracts to guarantee that the joint interactions of all the individual controllers still satisfy all the safety requirements, and that vehicles will still make progress towards their goals.
- We provide high-fidelity simulations on large-volume projected UAM traffic data to showcase the applicability of our proposed architecture.

II. Problem Setting and Preliminaries

Consider an environment consisting of an operating space and a network formed from a series of k UAM vertiport hubs labeled $V_1, \cdots, V_k$. A UAM vertiport hub (henceforth referred to as a vertihub) consists of a grouping of multiple vertiports, each of which may have multiple takeoff/landing pads. A vertihub is responsible for managing requests by UAM vehicles (henceforth referred to as vehicles) to either land at or take off from a desired vertiport in its region or pass through to a neighboring region. Each vertiport inside a vertihub is in charge of taking off and landing vehicles at its landing pads. An example of such an environment is illustrated in Figure 1a. Note that vertihub control regions need not be circular.

Example 1: The vertihub controller for region V_6 in Figure 1a controls the operational area defined by $V_6 - (V_6 \cap V_5)$. The area of overlap $H_{56} = V_5 \cap V_6$ is the region where the handoff takes place, wherein the vertihub controller of the region the vehicle is about to enter takes responsibility for the vehicle. Hence, V_6 can force vehicles incoming from V_5 to loiter in the handoff region H_{56} until it is safe to allow them to enter.

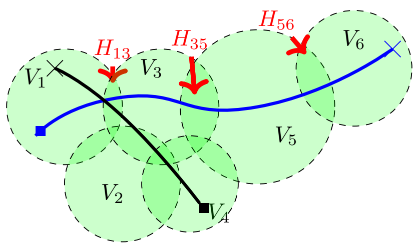
The number of vehicles allowed inside each vertihub is upper bounded both by the separation standards between the vehicles as well as with the complexity of the airspace (e.g., intersection with general aviation traffic etc.). Vertihubs cannot accept vehicles (i.e., accept a handoff) if the additional vehicle exceeds the maximum operational capacity or induces a conflict. Furthermore, vertiports cannot allow vehicles to take-off if it will violate the maximum operational capacity of the vertihub, and must force incoming vehicles to loiter if all of their pads are occupied. In order to avoid violating airspace requirements and to avoid build up of loitering vehicles (which can delay vehicles desiring to pass through or create safety issues), a vertiport must coordinate with its corresponding vertihub. We model the vertihub controller and vertiport controllers as reactive systems.

Definition: We consider a finite set I (O) of Boolean inputs (outputs). The input alphabet is $\Sigma_{\mathrm{I}} = 2^{\mathrm{I}}$, the output alphabet is $\Sigma_{\mathrm{O}} = 2^{O}$, and $\Sigma = \Sigma_{\mathrm{I}} \times \Sigma_{\mathrm{O}}$. The set of finite (infinite) traces over Σ is denoted by Σ^* (Σ^ω), and we define $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A reactive system is a tuple $\mathcal{D} = (\mathrm{Q}, q_0, \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, \delta, \lambda)$, where Q is a finite set of states, $q_0 \in \mathrm{Q}$ is the initial state, Σ_{I} is the input alphabet, Σ_{O} is the output alphabet, $\delta : \mathrm{Q} \times \Sigma_{\mathrm{I}} \to \mathrm{Q}$ is the complete transition function, and $\lambda : \mathrm{Q} \times \Sigma_{\mathrm{I}} \to \Sigma_{\mathrm{O}}$ is the output function. Given an input trace $\overline{\sigma}_{\mathrm{I}} = x_0 x_1 \ldots \in \Sigma_{\mathrm{I}}^\infty$, a reactive system $\mathcal{D}$ produces an output trace $\overline{\sigma}_{\mathrm{O}} = \mathcal{D}(\overline{\sigma}_{\mathrm{I}}) = \lambda(q_0, x_0)\lambda(q_1, x_1) \ldots \in \Sigma_{\mathrm{O}}^\infty$ with $q_{i+1} = \delta(q_i, x_i)$ for all $i \geq 0$. The set of words produced by $\mathcal{D}$ is denoted $L(\mathcal{D}) = \{\overline{\sigma}_{\mathrm{I}} \parallel \overline{\sigma}_{\mathrm{O}} \in \Sigma^\infty \mid \mathcal{D}(\overline{\sigma}_{\mathrm{I}}) = \overline{\sigma}_{\mathrm{O}}\}$.

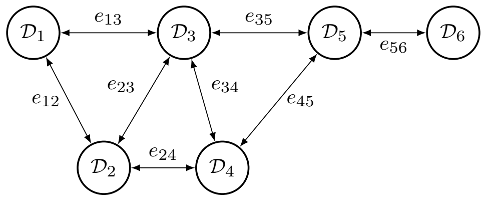
Ensuring the safety of the takeoff and landing operations at vertiports that share the same airspace must be balanced with bounding the delays experienced by vehicles. Furthermore, vehicles cannot loiter indefinitely due to energy constraints. Hence, vertihubs must additionally guarantee a finite upper bound on the delays experienced by vehicles. All of these requirement guarantees (and others) can be expressed as temporal logic specifications that controllers must satisfy.

Definition: A linear temporal logic (LTL) specification φ defines a set of allowed traces $L(\varphi) \subseteq L(\mathcal{D})$ for the reactive system $\mathcal{D}$. A reactive system $\mathcal{D}$ is winning with respect to specification φ iff $L(\mathcal{D}) \subseteq L(\varphi)$ and is denoted $\mathcal{D} \models \varphi$. Given a set of propositions AP, a formula in LTL describes a language in $(2^{\mathrm{AP}})^\omega$. LTL extends Boolean logic by the introduction of temporal operators such as $\bigcirc$ (next time), G (always), F (eventually), and $\mathcal{U}$ (until).

Informally, the main problem addressed in this paper is designing controllers for vertihubs and vertiports that guarantee all safety and progress requirements, assuming they have been correctly captured in the design process. More formally, the task of computing a satisfying controller in reactive systems involves constructing the



(a) Example UAM operating environment



(b) Connectivity graph

Fig. 1: (a) Green circles correspond to the region of a vertihub. UAM vehicles (blue and black) move between origin-destination vertiports in the environment. (b) The corresponding connectivity graph $G_{\mathcal{D}}$ of the vertihub controllers $\mathcal{D}$ modeling the sectors V. Each edge e_{ij} corresponds to $\mathcal{D}_i$ and $\mathcal{D}_j$ being connected, i.e., the outputs of $\mathcal{D}_i$ are inputs to $\mathcal{D}_j$ and vice versa.

function λ and can be typically framed as computing the winning strategy of a game.

Definition: A game structure is a tuple $\mathcal{G} = (\mathrm{Q}, q_0, \Sigma, \delta, \mathsf{Acc})$, where

- Q is a finite set of states, $q_0 \in \mathrm{Q}$ the initial state,
- $\Sigma = (\Sigma_{\mathrm{I}} \times \Sigma_{\mathrm{O}})$ is the alphabet of actions available to the environment and the controller respectively,
- $\delta : \mathrm{Q} \times \Sigma \to \mathrm{Q}$ is a complete transition function, that maps each state, input (environment action) and output (controller action) to a successor state.
- $\mathsf{Acc} : (\mathrm{Q} \times \Sigma \times \mathrm{Q})^{\omega} \to \mathbb{B}$ is the winning condition of the game.

At every state $q \in \mathrm{Q}$ (starting with q_0), the environment chooses an input $\sigma_{\mathrm{I}} \in \Sigma_{\mathrm{I}}$, and then the controller chooses some output $\sigma_{\mathrm{O}} \in \Sigma_{\mathrm{O}}$. These choices define the next state $q' = \delta(q, (\sigma_{\mathrm{I}}, \sigma_{\mathrm{O}}))$, and the process then continues from q'. This order of moves ensures that at each step the controller's action reacts to the current action of the environment. The resulting (infinite) sequence $\pi = (q_0, \sigma_{\mathrm{I},0}, \sigma_{\mathrm{O},0}, q_1)(q_1, \sigma_{\mathrm{I},1}, \sigma_{\mathrm{O},1}, q_2)\ldots$ is called a play, where q_0 is the initial state, and for every $i \geq 0$ we have that $q_{i+1} = \delta(q_i, \sigma_{\mathrm{I},i}, \sigma_{\mathrm{O},i})$. A play π is winning if $\mathsf{Acc}(\pi) = \top$.

We consider winning conditions expressed from a fragment of LTL specifications called Generalized Reactivity 1 (GR(1)), which is common in a variety of practical applications [2], [6], [23], [24]. A GR(1) winning condition is defined by sets of states $S_{\mathrm{I}}, S_{\mathrm{O}} \subseteq \mathrm{Q}$, $E_i \subseteq \mathrm{Q}$ for $i = 1, \ldots, m$ and $F_j \subseteq \mathrm{Q}$ for $j = 1, \ldots, n$, and consists of all plays π such that if $\pi \in \mathsf{G} S_{\mathrm{I}} \cap \mathsf{G}\,\mathsf{F}\, E_i$ for all $i = 1, \ldots, m$, then $\pi \in \mathsf{G} S_{\mathrm{O}} \cap \mathsf{G}\,\mathsf{F}\, F_j$ for all $j = 1, \ldots, n$. Intuitively, for a play π to be winning, whenever the environment satisfies the assumptions $\mathsf{G}\, S_{\mathrm{I}}, \mathsf{G}\,\mathsf{F}\, E_1, \ldots, \mathsf{G}\,\mathsf{F}\, E_m$, then the controller must satisfy all the guarantees $\mathsf{G}\, S_{\mathrm{O}}, \mathsf{G}\,\mathsf{F}\, F_j, \ldots, \mathsf{G}\,\mathsf{F}\, F_n$. By abuse of logical operators, we abbreviate GR(1) conditions as

$$\left(\mathsf{G}\, S_{\mathrm{I}} \wedge \bigwedge_{i=1}^{m} \mathsf{G}\,\mathsf{F}\, E_i\right) \implies \left(\mathsf{G}\, S_{\mathrm{O}} \wedge \bigwedge_{j=1}^{n} \mathsf{G}\,\mathsf{F}\, F_j\right).$$

Definition: A strategy for the controller is a function $\rho_{\mathrm{O}} : \mathit{Prefs}(\mathcal{G}) \times \Sigma_{\mathrm{I}} \to \Sigma_{\mathrm{O}}$ which maps a prefix of a run (the history of the play so far) and an action of the environment to an action of the controller. A strategy for the environment is a function $\rho_{\mathrm{I}} : \mathit{Prefs}(\mathcal{G}) \to \Sigma_{\mathrm{I}}$ that maps the prefix of the play so far to an action of the environment. We denote the sets of all strategies for the controller and for the environment by $\mathcal{M}_{\mathrm{O}}$ and $\mathcal{M}_{\mathrm{I}}$ respectively.

Every pair of strategies $\rho_{\mathrm{O}} \in \mathcal{M}_{\mathrm{O}}$ for the controller and $\rho_{\mathrm{I}} \in \mathcal{M}_{\mathrm{I}}$ for the environment define a play, denoted by $\Pi(\rho_{\mathrm{O}}, \rho_{\mathrm{I}})$. More precisely, $\Pi(\rho_{\mathrm{O}}, \rho_{\mathrm{I}}) = \pi = (q_0, \sigma_{\mathrm{I},0}, \sigma_{\mathrm{O},0}, q_1)(q_1, \sigma_{\mathrm{I},1}, \sigma_{\mathrm{O},1}, q_2)\ldots \in L(\mathcal{G})$ where for every $i \geq 0$, $\sigma_{\mathrm{I},i} = \rho_{\mathrm{I}}(\pi[0,i])$ and $\sigma_{\mathrm{O},i} = \rho_{\mathrm{O}}(\pi[0,i], \sigma_{\mathrm{I},i})$. Similarly, we define the set of plays starting at a state g that are consistent with ρ_{O}, denoted $L(\mathcal{D}, \rho_{\mathrm{O}}, g)$.

Given a game structure $\mathcal{D}$ and a winning condition φ for the agent, the synthesis problem is to generate a strategy $\rho_{\mathrm{O}} \in \mathcal{M}_{\mathrm{O}}$ for the controller such that for every strategy $\rho_{\mathrm{I}} \in \mathcal{M}_{\mathrm{I}}$ for the environment it holds that $\Pi(\rho_{\mathrm{I}}, \rho_{\mathrm{O}}) \in \varphi$, i.e., all resulting plays satisfy φ. In such cases we say that ρ_{O} satisfies φ, denoted $\rho_{\mathrm{O}} \models \varphi$.

III. UAM ATM Architecture

In this section, we introduce our novel reactive system architecture for UAM ATM. We first define the reactive controller models for the individual components in the architecture. We then construct the composition of all the controllers and present the formal synthesis problem statement. In the following section, we present our contract-based synthesis method for decentralized reactive synthesis of vertiport hubs and vertiport controllers.

A. Controller models

Vertihub controller: We model the controller of each vertihub V_i as a reactive system $\mathcal{D}_i = (\mathrm{Q}_i, q_{0_i}, \Sigma_{\mathrm{I}_i}, \Sigma_{\mathrm{O}_i}, \delta_i, \lambda_i)$ with input and output variables I_i and O_i respectively. The specific instantiation of such a controller is problem-specific, however, we present an illustrative example.

Example 2: Consider the environment in Figure 1a with vehicles moving between origin-destination vertiports. Each region V_i is a vertihub with corresponding vertihub controller $\mathcal{D}_i$ where

- The state space Q_i is the number of vehicles currently in the airspace of V_i, as well as the current delay time of any loitering vehicles in the airspace.
- q_{0_i} is the starting airspace configuration of aircraft in V_i.
- The input alphabet is given by $\Sigma_{\mathrm{I}_i} = 2^{\mathrm{I}_i}$. I_i is the set of input variables and corresponds to requests for the

hub controller and the number of available landing slots in the region. We divide the requests into the following: landing, pass-through, take-off.

- The output alphabet is given by $\Sigma_{\mathrm{O}_i} = 2^{\mathrm{O}_i}$. O_i is the set of output variables and corresponds to the following actions: allow vehicles to pass through, send vehicles to a vertiport in the region to land, or force vehicles to loiter until it is safe to allow them to enter. We note that the output can allow multiple requests to be granted simultaneously.
- The transition function δ_i increments or decrements the number of vehicles and their corresponding delay times in the region based on the environment inputs and the resulting controller output.

Together, all k vertihub controllers form a connected system which we define as set of reactive systems $\mathcal{D} = \{\mathcal{D}_1, \dots \mathcal{D}_k\}$ with a corresponding connectivity graph $G_{\mathcal{D}}$. We define a connectivity graph as a directed graph with each vertex corresponding to a reactive system. We say two reactive systems are connected if they share an edge in the graph. We define the set of reactive systems $\mathcal{D}_j \in \mathcal{D}$, $i \neq j$ that share an edge with $\mathcal{D}_i$ as *connect*$(\mathcal{D}_i)$.

Example 3: In Figure 1a, overlapping operational regions share an edge in the corresponding directed graph in Figure 1b and therefore the corresponding reactive systems are connected. We say that *connect*$(\mathcal{D}_1) = \{\mathcal{D}_2, \mathcal{D}_3\}$.

Note that a hub controller forcing vehicles to loiter in the handoff region affects the connected hub controller. For example, in Figure 1a, $\mathcal{D}_5$ forcing a vehicle loiter in H_{56} affects the airspace of $\mathcal{D}_6$ and will as a result limit the number of vehicles that $\mathcal{D}_6$ can accept. These handoffs necessitate the use of contracts between hubs to guarantee the global system behaves as desired.

Vertiport controller: We assume without loss of generality that all regions V_i contain m vertiports. We model the vertiport controller corresponding to region V_i as a reactive system $\mathcal{S}_i^j = (\mathrm{Q}_i^j, q_{0_i}^j, \Sigma_{\mathrm{I}_i}^{j'}, \Sigma_{\mathrm{O}_i}^j, \delta_i^j, \lambda_i^j)$ with Q_i^j being the state space and $q_{0_i}^j$ being the initial state. The input alphabet $\Sigma_{\mathrm{I}_i}^{j'}$ is defined to be $\Sigma_{\mathrm{I}_i}^{j'} = \Sigma_{\mathrm{I}_j}^i \times \Sigma_{\mathrm{O}_i}$. This construction allows for the output of $\mathcal{D}_i$ to form part of the input for $\mathcal{S}_j^i$ for $j = 1, \dots, m$.

Figure 2 illustrates the relationship between hubs and vertiport controllers as well as the architecture of the composition.

Example 4: Continuing the running example based on Figure 1a, each vertiport controller in the region V_i has a corresponding set of input variables denoted I_i^j and a resulting alphabet $\Sigma_{\mathrm{I}_i}^j$. This input corresponds to vehicles requesting to land that have been cleared by the corresponding vertihub controller $\mathcal{D}_i$ and vehicles desiring to take off at one of its pads. Hence, the output of $\mathcal{D}_i$ forms part of the input of $\mathcal{S}_j^i$ for $j = 1, \dots, m$. The output of the vertiport controller is the accepted or rejected take-off and landing requests as well as the number of remaining available landing pads which will then form part of the input to the hub controller.

In order for the reactive hub controller to be able to guarantee liveness properties, such as an upper bound on delay for all agents (a requirement), it needs to know the maximum time the vertiport controller will occupy a landing slot, i.e., it needs to know the worst case length of time a landing slot will be unavailable. Such an interaction between vertiport controller and hub controller is an example of a contract and will be formally detailed in section IV.

B. Controller composition

As mentioned previously, the inputs and outputs of the vertihub and vertiport controllers are linked. In this section, we formalize this notion by constructing the composition of controllers. We first define the composition of the m vertiport controllers $\mathcal{S}_i^j$ for all $j = 1, \dots, m$ corresponding to hub controller $\mathcal{D}_i$. Formally, the vertiport controllers $\{\mathcal{S}_i^1, \dots, \mathcal{S}_i^m\}$ in region V_i can be composed as $\mathcal{S}_i = \mathcal{S}_i^1 \circ \dots \circ \mathcal{S}_i^m$. The resulting composition is also a reactive system $\mathcal{S}^i = (\overline{\mathrm{Q}}_i, \overline{q}_{0_i}, \overline{\Sigma}_{\mathrm{I}_i}, \overline{\Sigma}_{\mathrm{O}_i}, \overline{\delta}_i, \overline{\lambda}_i)$ defined as follows:

- the set $\overline{\mathrm{Q}}_i = \bigotimes_j \mathrm{Q}_i^j$ of states is formed by the product of the states of all vertiports $\mathcal{S}_j^i \in \mathcal{S}^i$.
- The initial state $\overline{q}_0$ is formed by the initial states $q_{0_i}^j$ of all $\mathcal{S}_i^j \in \mathcal{S}^i$.
- The input alphabet $\overline{\Sigma}_{\mathrm{I}_i}$ is given by $\overline{\Sigma}_{\mathrm{I}_i} = \bigotimes_{j=1}^m \Sigma_{\mathrm{I}_i}^{j'}$.
- The output alphabet $\overline{\Sigma}_{\mathrm{O}_i}$, of the joint system $\mathcal{S}^i$ is given by $\overline{\Sigma}_{\mathrm{O}_i} = \bigotimes_{j=1}^m \Sigma_{\mathrm{O}_i}^j$.
- The transition function $\overline{\delta}_i$ updates, for each vertiport controller $\mathcal{S}_i^j \in \mathcal{S}_i$, the Q_i^j part of the state in accordance with the transition function δ_i^j.
- The output function $\overline{\lambda}_i$ labels each state with the union of the outputs of all $\mathcal{S}_j^i \in \mathcal{S}^i$ according to λ_i^j.

Next, we define the composition of the joint vertiport controllers $\mathcal{S}^i$ Formally, we compose the two systems in serial fashion in the following way: $\mathcal{D}_i \circ \mathcal{S}_i := \mathcal{V}_i = (\hat{\mathrm{Q}}_i, \hat{q}_{0_i}, \hat{\Sigma}_{\mathrm{I}}, \Sigma_{\mathrm{O}_i}, \hat{\delta}_i, \hat{\lambda}_i)$, with

- states $\hat{\mathrm{Q}}_i = \mathrm{Q}_i \times \overline{\mathrm{Q}}_i$, $\hat{q_0}_i = (q_{0_i}, \overline{q}_{0_i})$,
- input alphabet $\hat{\Sigma}_{\mathrm{I}} = \Sigma_{\mathrm{I}_i} \times \overline{\Sigma}_{\mathrm{O}_i}$,
- transition function

$$\hat{\delta}_i((q_i, \overline{q}_i), \hat{\sigma}_{\mathrm{I}}) = (\delta_i(q_i, \sigma_{\mathrm{I}_i}), \overline{\delta}_i(\overline{q}_i, (\overline{\sigma}_{\mathrm{O}_i}, \sigma_{\mathrm{I}_i}))),$$

- and output function

$$\hat{\lambda}_i((q_i, \overline{q}_i), \hat{\sigma}_{\mathrm{I}}) = \lambda_i(q_i, (\sigma_{\mathrm{I}_i}, \overline{\lambda}_i(\overline{q}_i, \overline{\sigma}_i))).$$

Note that, by construction, the output alphabet of the composition Σ_{O_i} is the same as that of the hub controller.

Finally, the global system $\mathcal{V} = \{\mathcal{V}_1 \dots \mathcal{V}_k\}$ is the composition of all V_i and the definition proceeds analogously to the composition of the vertiports. The architecture of the defined composition in an example environment with two hub controllers is illustrated in Figure 2. We remark that the global composition is of the form of a multi-agent reactive system as studied in [5].

In the next 2 subsections, we define the properties that we want the global system to satisfy and formally define

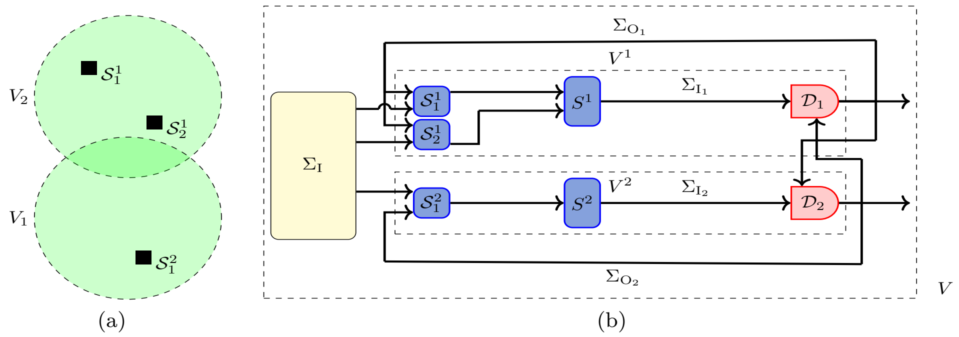


Fig. 2: (a) Example region space with three vertiports and two operating regions with corresponding vertihubs and (b) Architecture of the composition of reactive systems.

the synthesis problem. In Section IV, we present our synthesis procedure for the vertiport and hub controllers and prove the global system stemming from their composition will satisfy all required properties.

C. Specifications

Recall that each hub controller is required to satisfy a user provided specification. Formally, the specification for hub controller $\mathcal{D}_i$ must be of the form

$$\varphi_{\mathcal{D}_i} = \mathsf{G} S_i \wedge \mathsf{GF} P_i \tag{1}$$

where $S_i, P_i \subseteq Q_i$.

Similarly, each vertiport controller must also satisfy its own user-provided specification, and is defined the same way. For vertiport controller $\mathcal{S}_i^j$ we have $\varphi_{\mathcal{S}_i^j} = \mathsf{G} S_i^j \wedge \mathsf{GF} P_i^j$ where $S_i^j, P_i^j \subseteq Q_i^j$.

For each region V_i, we denote the conjunction of all the vertiport specifications $\varphi_{\mathcal{S}_i^j}$ for $j = 1, \ldots, m$ with the corresponding vertihub specification $\varphi_{\mathcal{D}_i}$ as $\varphi_i := \varphi_{\mathcal{D}_i} \bigwedge \left(\varphi_{\mathcal{S}_i^1} \wedge \ldots \wedge \varphi_{\mathcal{S}_i^m}\right)$.

Example 5: A simple example of a specification in the form of (1) can be G (fewer than N vehicles in region) $\wedge$ GF(vehicle request is granted). Informally, such a specification requires that no more than N vehicles be in the region at any given time and that always eventually any vehicle is allowed to pass-through or land in the region if they request it, i.e., they cannot be made to wait forever.

D. Synthesis Problem

Now we define the basic requirements that the overall composed system must satisfy: namely it should enforce correctness with respect to a given user specification. Informally, this translates to enforcing safety while guaranteeing progress.

Given (1) environment structure with k vertihubs, (2) m corresponding vertiports for each hub, (3) hub connectivity graph G_D, and (4) GR(1) specifications $\varphi_{\mathcal{D}_1}, \ldots, \varphi_{\mathcal{D}_k}$ for each vertihub and vertiport specifications $\varphi_{\mathcal{S}_i^j}$ for all $i = 1, \ldots, k$ and $j = 1, \ldots, m$ synthesize a set of hub controllers $\mathcal{D} = [\mathcal{D}_1, \ldots, \mathcal{D}_k]$ and their corresponding vertiport controllers $\mathcal{S}_j^i$ for all $i = 1, \ldots, k$ and $j = 1, \ldots, m$ such that the global composition of all controllers is winning with respect to $\bigwedge_{i=1\ldots k} \varphi_i$. Formally, synthesize hub and vertiport controllers such that:

$$\left(\mathcal{D}_1 \circ \left(S_1^1 \circ \ldots S_1^m\right)\right) \circ \ldots \left(\mathcal{D}_k \circ \left(S_k^1 \circ \ldots S_k^m\right)\right) \models \bigwedge_{i=1\ldots k} \varphi_i \tag{2}$$

In the next section, we present the decentralized contract-based synthesis framework to generate the controllers.

IV. Decentralized Controller Synthesis Framework

We first formalize the notion of assume-guarantee contracts, then we demonstrate the incorporation of these contracts as a GR(1) winning condition to a two-player game, which we solve using reactive synthesis [9], [28].

A. Assume-guarantee contracts

We employ a decentralized synthesis process, whereby each controller is synthesized without an awareness of the specification and implementation details of both the vehicles in the fleet, as well as the controllers in connected vertihubs. However, a vertihub controller's outputs impact the controllers of its neighboring vertihubs. In order to ensure that controllers don't hinder each others' abilities to satisfy their local specifications, every controller must additionally satisfy contract specifications for each neighboring vertihub. Similarly, all vertiport controllers also impact the implementation of their corresponding vertihub controllers. Therefore, the vertiport controllers must also satisfy contract specifications with their hub controller and vice versa.

These contract specifications take the form of assume-guarantee contracts. Informally, a vertihub controller gives a guarantee of satisfying a contract specification with a neighboring vertihub controller. This guarantee is used as an assumption for the synthesis of the neighboring controller and vice-versa. These contract specifications are taken into account in the synthesis process for all controllers.

In this setting, we introduce two classes of contracts: contracts between connected vertihub controllers and contracts between a vertihub and its corresponding vertiport controllers.

Vertihub controller contracts: An assume-guarantee contract for a vertihub controller $\mathcal{D}_i$ with connected hub controller $\mathcal{D}_j$ is a tuple $\phi_{\mathcal{D}_i}^{\mathcal{D}_j} = (A_{\mathcal{D}_i}^{\mathcal{D}_j}, B_{\mathcal{D}_i}^{\mathcal{D}_j})$ where

- $A_{\mathcal{D}_i}^{\mathcal{D}_j}$ is an assumption on the outputs of $\mathcal{D}_j$ as it pertains to $\mathcal{D}_i$ expressed as a specification in the form shown in Equation (1).
- $B_{\mathcal{D}_i}^{\mathcal{D}_j}$ is a specification also in the form shown in Equation (1) which $\mathcal{D}_i$ must guarantee on the outputs pertaining to $\mathcal{D}_j$.

Similarly, the contract for $\mathcal{D}_j$ with $\mathcal{D}_i$ is denoted $\phi_{\mathcal{D}_j}^{\mathcal{D}_i} = (A_{\mathcal{D}_j}^{\mathcal{D}_i}, B_{\mathcal{D}_j}^{\mathcal{D}_i})$.

Example 6: Consider Figure 2. An example of a contract that $\mathcal{D}_1$ will make with $\mathcal{D}_2$ is an upper bound on the length of time $\mathcal{D}_1$ can refuse to accept vehicles from $\mathcal{D}_2$ when requested. Such a contract is expressed as $\phi_{\mathcal{D}_2}^{\mathcal{D}_1} = (A_{\mathcal{D}_2}^{\mathcal{D}_1}, B_{\mathcal{D}_2}^{\mathcal{D}_1})$ where $A_{\mathcal{D}_2}^{\mathcal{D}_1} = B_{\mathcal{D}_2}^{\mathcal{D}_1} = \mathsf{G}(\text{delay} \leq \text{T})$ for some integer value T. Hence, the controller $\mathcal{D}_1$, in addition to satisfying its local specifications, must also guarantee $B_{\mathcal{D}_2}^{\mathcal{D}_1}$, i.e., it does not keep an aircraft from $\mathcal{D}_2$ waiting for more than T timesteps. In order to help satisfy the additional specification, $\mathcal{D}_1$ makes the assumption $A_{\mathcal{D}_2}^{\mathcal{D}_1}$ that $\mathcal{D}_2$ will satisfy $B_{\mathcal{D}_1}^{\mathcal{D}_2}$. Recall that the user given specification for $\mathcal{D}_1$ is denoted $\varphi_{\mathcal{D}_1}$. Under this contract, the augmented requirement $\varphi'_{\mathcal{D}_1}$ for $\mathcal{D}_1$ is

$$\varphi'_{\mathcal{D}_1} := A_{\mathcal{D}_2}^{\mathcal{D}_1} \implies \varphi_{\mathcal{D}_1} \wedge B_{\mathcal{D}_2}^{\mathcal{D}_1}$$

Simply, if the assumption on $\mathcal{D}_2$ holds, then the original specification must be satisfied in addition to contract specification $B_{\mathcal{D}_2}^{\mathcal{D}_1}$. The same procedure follows for $\mathcal{D}_2$.

Vertiport controller contracts: Each vertihub controller must also make contracts with the vertiport controllers in its region. The procedure follows analogously as in the previous section as we assume that all outputs from the individual controllers are pairwise disjoint. Hence, we can form separate contracts with each vertiport controller. Formally, we define a contract between a vertihub controller $\mathcal{D}_i$ and one of its vertiport controllers $\mathcal{S}_i^j$ as $\phi_{\mathcal{S}_i^j}^{\mathcal{D}_i} = (A_{\mathcal{S}_i^j}^{\mathcal{D}_i}, B_{\mathcal{S}_i^j}^{\mathcal{D}_i})$ where $(A_{\mathcal{S}_i^j}^{\mathcal{D}_i}, B_{\mathcal{S}_i^j}^{\mathcal{D}_i})$ are, analogous to the hub contracts, the assumptions and guarantees on the vertiport controller respectively, expressed as temporal logic specifications of the form in (1).

Symmetric contracts: We say contracts between two reactive systems $\mathcal{D}_i$ and $\mathcal{D}_j$ denoted by $\phi_{\mathcal{D}_i}^{\mathcal{D}_j}$ and $\phi_{\mathcal{D}_j}^{\mathcal{D}_i}$ are symmetric if $A_{\mathcal{D}_i}^{\mathcal{D}_j} = B_{\mathcal{D}_j}^{\mathcal{D}_i}$ and $A_{\mathcal{D}_j}^{\mathcal{D}_i} = B_{\mathcal{D}_i}^{\mathcal{D}_j}$.

Informally, this means that the assumptions $\mathcal{D}_i$ makes on the outputs of $\mathcal{D}_j$ corresponds to the guarantees $\mathcal{D}_j$ gives on its own outputs. All the contracts we will use in the synthesis procedure will be symmetric as this will guarantee that the assumptions each reactive system make on the outputs of other systems will actually hold.

Joint winning condition: The addition of the contract specifications to the original user-provided specifications modifies the winning condition presented in (2). The following lemma presents the modified winning condition for hub controller $\mathcal{D}_i$ when contracts are incorporated.

Lemma 1: For a set of vertihub controllers $\mathcal{D}$, hub connectivity graph $G_{\mathcal{D}}$, the winning condition for controller $\mathcal{D}_i \in \mathcal{D}$ with the set of vertiport controllers $\mathcal{S}_i^j$ for $j = 1, \ldots, m$ is given by

$$\varphi'_i := \left(\bigwedge_{j=1}^{m} A_{\mathcal{S}_i^j}^{\mathcal{D}_i} \bigwedge_{\mathcal{D}_j \in J} A_{\mathcal{D}_j}^{\mathcal{D}_i} \implies \varphi_{\mathcal{D}_i} \wedge B_{\mathcal{S}^i}^{\mathcal{D}_i} \bigwedge_{\mathcal{D}_j \in J} B_{\mathcal{D}_j}^{\mathcal{D}_i} \right) \tag{3}$$

where $J = \mathit{connect}(\mathcal{D}_i)$.

Note that (3) is in the same form as the specifications in (1) which allows us to use efficient GR(1) synthesis techniques [9]. We state this formally in the following lemma.

Lemma 2: The winning condition in (3) is GR(1).

Remark: We note that the values in the assume-guarantee contracts are heavily dependent on the topology of the environment and the connections (see Fig. 3). For example, a hub with many connecting hubs may not be able to guarantee quick transit of vehicles through its regions. Generating these contracts automatically based on the given graph $G_{\mathcal{D}}$ is a subject of future work. In this paper, the contract values are chosen manually.

B. Controller synthesis

We first present an overview of the synthesis procedure. Informally, we avoid the centralized synthesis problem by creating a series of smaller synthesis problems that are connected through the architecture introduced in Section III and the contract specifications introduced in Section IVA. The synthesis procedure for a vertihub controller $\mathcal{D}_i$ consists of the following steps:

1) First, we synthesize all the vertiport controllers $\mathcal{S}_i^j$ for $j = 1, \ldots, m$ operating under $\mathcal{D}_i$. For each controller $\mathcal{S}_i^j$ we construct a game $\mathcal{G}_i^j$ with the acceptance condition given by the user-provided specification $\varphi_{\mathcal{S}_i^j}$. We then augment the acceptance condition to include the contract specification $\phi_{\mathcal{S}_i^j}^{\mathcal{D}_i}$.
2) As shown in Lemma 2, the winning condition stemming from augmenting φ_i with the contract specifications is a GR(1) condition. We solve each game $\mathcal{G}_i^j$ using GR(1) reactive synthesis [9].
3) We compose the joint vertiport controllers to form $\mathcal{S}_i$. We construct a game $\mathcal{G}_i$ from the given specification $\varphi_{\mathcal{D}_i}$ for the vertihub controller $\mathcal{D}_i$, again augmenting the acceptance condition with the vertiport contract specifications $\phi_{\mathcal{D}_i}^{\mathcal{S}_i^1} \wedge \ldots \wedge \phi_{\mathcal{D}_i}^{\mathcal{S}_i^m}$. We now additionally augment the acceptance condition with contract specifications $\phi_{\mathcal{D}_i}^{\mathcal{D}_j}$ from all connected vertihub controllers $\mathcal{D}_j \in \mathit{connect}(\mathcal{D}_i)$. Similar to before, this results in another GR(1) acceptance condition and we can compute a winning strategy.
4) The process is repeated for each hub controller.

Game construction: Recall that computing the output function λ_i corresponding to the reactive system $\mathcal{D}_i$ is framed as finding the winning strategy obtained from solving a game. The game construction is identical for the vertiport and vertihub controllers, just with different specifications and contracts. For brevity, we present the game construction for the synthesis for vertihub controller $\mathcal{D}_i$.

Let $\varphi_{\mathcal{D}_i} = \mathsf{G}S_i \wedge \mathsf{GF}P_i$ with $P_i, S_i \in Q_i$ be the user-provided specification for reactive system $\mathcal{D}_i = (Q_i, q_{0_i}, \Sigma_{\mathrm{I}_i}, \Sigma_{\mathrm{O}_i}, \delta_i, \lambda_i)$. The corresponding game $\mathcal{G}$ is constructed as follows: $\mathcal{G} = (Q, q_0, \Sigma, \delta, \mathsf{Acc})$ where $Q = Q_i$, $q_0 = q_{0_i}$, $\Sigma = (\Sigma_{\mathrm{I}_i} \times \Sigma_{\mathrm{O}_i})$, $\delta = \delta_i$, and Acc is the winning condition presented in Equation (3). The task is to synthesize a strategy ρ that maps the current state q_i and environment input σ_i to an action σ_0 such that for all possible input sequences $\sigma_i \in \Sigma_{\mathrm{I}_i}^*$ we will have $\mathsf{Acc}(\pi) = \top$. From Lemma 2, we know that Acc is a GR(1) winning condition. Hence, we are able to use the techniques in [9] in order to efficiently synthesize a strategy $\rho : Q \times \Sigma_{\mathrm{I}_i} \to \Sigma_{\mathrm{O}_i}$. Setting the output function λ_i corresponding to hub controller $\mathcal{D}_i$ to be the winning strategy ρ for game $\mathcal{G}$ as defined above leads to the following lemma.

Lemma 3: $\mathcal{D}_i \models \varphi_i$ if $\rho \models \varphi_i'$ and $\left(\bigwedge_{j=1}^{m} A_{\mathcal{S}_i^j}^{\mathcal{D}_i} \bigwedge_{\mathcal{D}_j \in J} A_{\mathcal{D}_j}^{\mathcal{D}_i}\right)$ holds.

Proof: The proof relies on the game construction having an identical state space as the reactive system and the definition of φ_i' being both the original user-provided specification φ_i as well as the contract specification. Since the game strategy satisfies φ_i', then the same strategy must also satisfy φ iff the assumptions made on neighboring vertihubs and interior vertiports are satisfied. Informally, the lemma states that the hub controller $\mathcal{D}_i$ satisfies its given specification φ_i if the synthesized strategy ρ from the game $\mathcal{G}$ is winning with respect to the contract augmented specification φ_i' and the contract assumptions hold.

Remark: The above lemma is only a statement of sufficiency. This is because the contract specifications that form φ_i' are not unique. If $\rho \not\models \varphi_i'$, for a particular set of contracts, this does not mean there does not exist a λ_i such that $\mathcal{D}_i \models \varphi$.

C. Correctness

In this section we present the correctness result that states that if there exists a set of controllers synthesized using the construction detailed earlier, they must be winning with respect to the conjunction of all user provided specifications.

Theorem 1: If there exists a set of controllers $\mathcal{D} = [\mathcal{D}_1, \ldots, \mathcal{D}_k]$ with corresponding vertiport controllers $\mathcal{S}_i^j$ for $j = 1, \ldots, m$ and $i = 1, \ldots, k$ such that $\mathcal{D}_i \models \varphi_i'$ where φ_i' is given in Equation (3), then the controllers must also satisfy Equation (2).

We remark that this construction guarantees correctness if there exists a set of winning strategies in the game construction, leading to a proof by construction using Lemmas 2, 3. However, since the game acceptance condition is augmented with contract-induced specifications, the construction cannot always guarantee that such a controller exists.

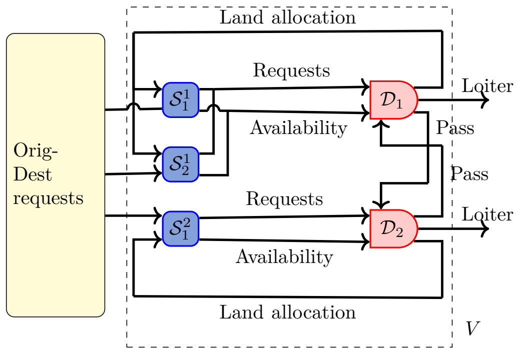


Fig. 3: Implementation of the architecture in Figure 2 for UAM ATM case study.

V. Case Study Simulation

A. Simulation setting

We demonstrate our approach on large-volume UAM air traffic data. The data used in the simulation was generated by NASA Langley in conjunction with partners performing UAM demand studies, and is in a format compatible with the Mission Planner Algorithm [18] developed at NASA Langley. The data contains simulated, timestamped on-demand requests for origin-destination trips. The dataset includes the position of 1000 vertiports which serve as the origin-destination pairs. In practice, the vertihub placement and size will correspond to physical infrastructure and we thus treat them as given inputs to the controller synthesis problem. Requests are spawned when the timestamp corresponding to a trip request in the data is reached. Each vertihub handles the requests of the UAM vehicles as they come into range (defined by the circles in Fig. 5). Note that increasing the number of vertihubs leads to the balkanization of the airspace.

Each vehicle has a process flow from take-off to landing (Fig. 4). As a vehicle enters the range of a vertihub, it sends the vertihub controller a request to either land or pass through. Then the vertihub controller sends one of three commands:

1) If permission is granted to pass-through, the vehicle (green) flies to the next vertihub and requests access.
2) If permission is granted to land, the vehicle (yellow) approaches the requested vertiport and sends a landing request to the vertiport controller.
3) If neither permission is granted, the vehicle loiters (orange) and repeats its request.

In order to demonstrate the qualitative behavior of the global system, we examine the procedure with multiple operating vertihubs in three different settings where the vehicles are operating in: (i) an airspace covered by vertihubs that have many overlapping regions – the many intersecting regions of control require a vehicle to

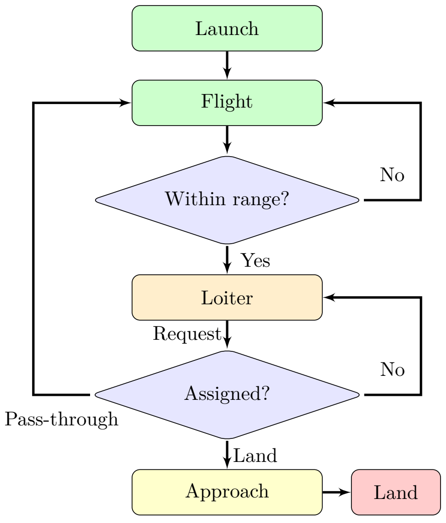
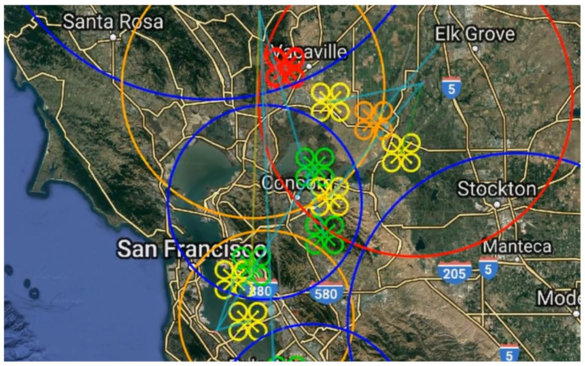


Fig. 4: Process flow for an individual vehicle - during flight it checks whether it is in range of a vertihub. Once within range, the vehicle loiters and creates a request to either land or pass-through. The colors in the figure are used in the simulations to represent the current status of the vehicle.



(a)

Fig. 5: Screenshot of the simulation environment - Operating regions for each vertihub controller and their ranges are indicated by blue circles. Green agents are in flight mode, orange agents are loitering, yellow agents are allocated for landing/passing-through and red agents are those that have landed. If a vertihub controller circle is red, then it will accept no more agents until the agents inside have landed or have been accepted to pass through to another region.

receive many permissions to complete its mission. (ii) sparsely interacting vertihubs – an airspace with only a few overlapping regions of control. Vehicles passing through these vertihubs negotiate few permissions. (iii) closely linked vertihubs – a special condition where the overall airspace has few overlapping regions of control but the overlapping regions are physically close together. In this scenario, the vehicles' missions compel them to pass through these regions simulating congestion a potential bottleneck.

All vertihub controllers must satisfy the following specifications: (1) landing or pass-through requests must be approved in less than T steps and (2) there can be no more than N vehicles in the region of the vertihub. All vertiport controllers must satisfy the following: no more than M vehicles can land at any given time. The landing time is treated as an environment input and can vary based on the vehicle and weather conditions. The contract between the vertiport and vertihub states that a vertiport may not allow a vehicle to take off for up to L steps and must clear a landing spot in at most K steps. The vertihub controllers have contracts with connected controllers agreeing to let vehicles loiter in their regions for $t < T$ timesteps. These contracts allow vertihub controllers to land or let vehicles pass through while incoming vehicles loiter in neighboring regions. The resulting video simulations for each setting can be seen in https://u-t-autonomous.github.io/Decentralized-UAM-Traffic-Management/. Also included in the link are further comparisons with different physical architectures for vertihub placement - in particular we compare scenario (i) when the airspace is covered by double the number of the vertihubs. The average loiter time for vehicles in setting (i) was approximately 3% less per vehicle for the environment with more vertihubs, however, we note that this is not true in general and will depend on the specific network topology and specifications.

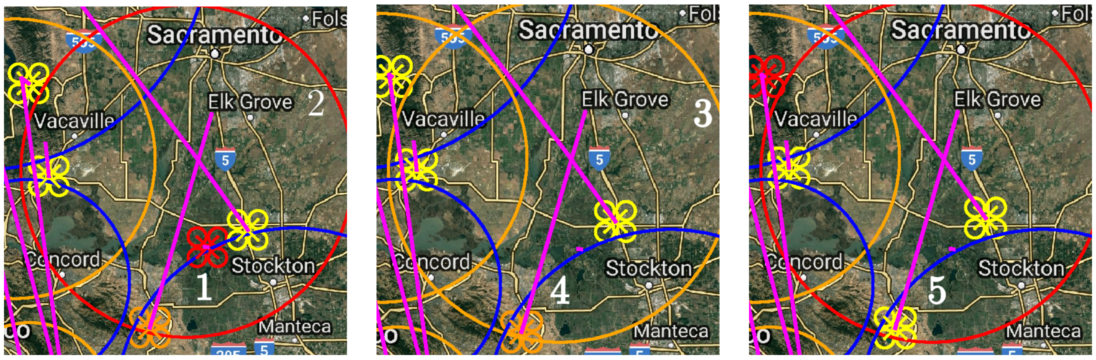
B. Qualitative Results

Figure 6 contains an example of the allocation behavior for multiple agents approaching a single vertihub. Initially the vertihub is informed by the vertiports that there are no slots available for landing (see 2 in Fig. 6a). Subsequent to an agent landing (1 lands), an additional slot opens (see 3 in Fig. 6b). Finally, the vertihub allocates a slot to a requesting vehicle (4 becomes 5 in figure 6c).

For each simulation setting, the vertihub interactions affect how vehicles are allocated and subsequently where they loiter. In the case of (i), allocation bottlenecks tend to occur upon vehicle launch – as a vehicle launches it waits for permission to move and thus bottlenecks are limited by the operating number of vehicles in the vertihub. For (ii), there are significantly fewer bottlenecks but they mostly occur as vehicles either enter or exit a region. For (iii) we observe the cascading of allocations in each region – to allocate vehicles in the top region, the vehicles in the middle region must clear and similarly for the middle-lower region interactions.

C. Synthesis time

We present the synthesis times for the vertihub and vertiport controllers in Table I. There is an exponential increase in the synthesis time as the number of vertiports in each region is increased. The state space of the controllers does not depend on the number of vehicles, allowing us to simulate datasets involving large numbers of vehicles. The centralized synthesis method in [6] could not be run even in the smallest case as the state space is too large. A decentralized, hierarchical procedure is necessary to handle systems of the necessary size and complexity that UAM will require. Other decentralized



(a) Landing (b) Free slot (c) Allocated

Fig. 6: Time sequence of a UAV landing (1) at a vertiport, which opens up a free slot for the vertihub ($2 \rightarrow 3$). The hub then allocates another UAV ($4 \rightarrow 5$) to land at a free slot in one of the vertiports.

techniques such as [7] can only handle very restrictive classes of specifications and could not be used to generate controllers in this setting.

We note that since the synthesis for each vertihub is independent of one another once the contracts are generated, the synthesis procedure for the global system is trivially parallelizable.

VI. Conclusion

We introduced a novel UAM ATM architecture for high volume operations in urban environments. We show that by using contract-based reactive synthesis we can achieve correctness guarantees in a scalable fashion that is not attainable via centralized synthesis. Controllers can be synthesized independently of one another as long as they satisfy additional contract specifications. These contract specifications guarantee that individual controllers will not impede the ability of connected systems to satisfy the desired emergent properties while satisfying their own specifications. Hence, the global connected system is theoretically guaranteed to be correct with respect to all user-provided temporal logic requirements. For future work, we aim to: (1) investigate the automated generation of assume-guarantee contracts based on a given network structure, (2) use the designed framework as an operating envelope in which to optimize for metrics such as loiter times while always guaranteeing safety requirements, (3) directly include resource-constrained planning in the presented UAM ATM framework, and (4) perform airspace design exploration for vertihub and vertiport infrastructure placement.

Acknowledgement

The authors would like to thank Ali Husain and Dr. Bruce Porter from SkyGrid Inc. for inspiring discussions. They would also like to thank Mr. Rick Butler, Dr. George Hagen, Mr. Jeffrey Maddalon and Mr. Nelson Guerreiro for their insight.

TABLE I: Synthesis times for vertihub/vertiport controllers

	No. of vertiports / No. of landing pads	No. of states in controller ($\|Q_i\|$)	Synthesis time (s)
Vertihub	3	2.1×10^4	22.75
	4	2.7×10^6	1272.95
	5	3.4×10^8	14300.65
Vertiport	2	6.1×10^2	0.45
	4	4.3×10^4	19.12
	8	1.9×10^8	12403.86

References

[1] Flight plan 2030: An air traffic management concept for urban air mobility. EmbraerX (2019)

[2] Alonso-Mora, J., DeCastro, J.A., Raman, V., Rus, D., Kress-Gazit, H.: Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles. Auton. Robots 42(4), 801–824 (2018)

[3] Alur, R., Moarref, S., Topcu, U.: Compositional synthesis of reactive controllers for multi-agent systems. In: International Conference on Computer Aided Verification. pp. 251–269. Springer (2016)

[4] Bauer, A., Falcone, Y.: Decentralised LTL monitoring. Formal Methods in System Design 48(1-2), 46–93 (2016)

[5] Bharadwaj, S., Bloem, R., Dimitrova, R., Konighofer, B., Topcu, U.: Synthesis of minimum-cost shields for multi-agent systems. In: 2019 American Control Conference (ACC). pp. 1048–1055 (2019)

[6] Bharadwaj, S., Dimitrova, R., Topcu, U.: Synthesis of surveillance strategies via belief abstraction. In: 2018 IEEE Conference on Decision and Control (CDC). pp. 4159–4166 (2018)

[7] Bharadwaj, S., Carr, S., Neogi, N., Poonawala, H., Chueca, A.B., Topcu, U.: Traffic management for urban air mobility. In: NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings. pp. 71–87 (2019)

[8] Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: Handbook of Model Checking, pp. 921–962. Springer (2018)

[9] Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa ar, Y.: Synthesis of reactive (1) designs. Journal of Computer and System Sciences 78(3), 911–938 (2012)

[10] Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 533–548. Springer (2015)

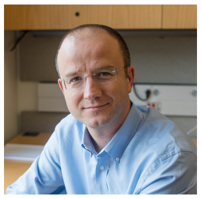
[11] Cassar, I., Francalanza, A.: On implementing a monitor-oriented programming framework for actor systems. In: Integrated Formal Methods - 12th International Conference, IFM 2016, Reyk-

javik, Iceland, June 1-5, 2016, Proceedings. LCNS, vol. 9681, pp. 176–192. Springer (2016)
[12] Cook, A.: European air traffic management: principles, practice, and research. Ashgate Publishing, Ltd. (2007)
[13] FAA: Order JO 7400.9Y Air Traffic Organization Policy (2014)
[14] Falcone, Y.: You should better enforce than verify. In: Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings. pp. 89–105 (2010)
[15] Falcone, Y., Jaber, M., Nguyen, T., Bozga, M., Bensalem, S.: Runtime verification of component-based systems in the BIP framework with formally-proved sound and complete instrumentation. Software and System Modeling 14(1), 173–199 (2015)
[16] Foina, A.G., Sengupta, R., Lerchi, P., Liu, Z., Krainer, C.: Drones in smart cities: Overcoming barriers through air traffic control research. In: 2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS). pp. 351–359 (2015)
[17] Francalanza, A., Seychell, A.: Synthesising correct concurrent runtime monitors. Formal Methods in System Design 46(3), 226–261 (2015)
[18] Guerreiro, N.M., Butler, R.W., Maddalon, J.M., Hagen, G.E.: Mission planner algorithm for urban air mobility–initial performance characterization. In: AIAA Aviation 2019 Forum. p. 3626 (2019)
[19] Harriet, T., Poku, K., Emmanuel, A.K.: An assessment of traffic congestion and its effect on productivity in urban Ghana. International Journal of Business and Social Science 4(3) (2013)
[20] Hong, Y., Choi, B., Lee, K., Kim, Y.: Conflict management considering a smooth transition of aircraft into adjacent airspace. IEEE Transactions on Intelligent Transportation Systems 17(9), 2490–2501 (Sep 2016)
[21] Könighofer, B., Alshiekh, M., Bloem, R., Humphrey, L., Könighofer, R., Topcu, U., Wang, C.: Shield synthesis. Formal Methods in System Design 51(2), 332–361 (2017)
[22] Liu, Z., Sengupta, R.: An energy-based flight planning system for unmanned traffic management. In: 2017 Annual IEEE International Systems Conference (SysCon). pp. 1–7 (April 2017)
[23] Maoz, S., Ringert, J.O.: Gr(1) synthesis for ltl specification patterns. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. pp. 96–106. ESEC/FSE 2015, ACM, New York, NY, USA (2015)
[24] Moarref, S., Kress-Gazit, H.: Reactive synthesis for robotic swarms. In: Formal Modeling and Analysis of Timed Systems - 16th International Conference, FORMATS 2018, Beijing, China, September 4-6, 2018, Proceedings. pp. 71–87 (2018)
[25] Moore, A., Balachandran, S., Young, S.D., Dill, E.T., Logan, M.J., Glaab, L.J., Munoz, C., Consiglio, M.: Testing enabling technologies for safe UAS urban operations. In: Proceedings of the 2018 Aviation, Technology, Integration, and Operations Conference. No. AIAA-2018-3200, Atlanta, Georgia (June 2018)
[26] Muñoz, C., Narkawicz, A., Hagen, G., Upchurch, J., Dutle, A., Consiglio, M., Chamberlain, J.: Daidalus: Detect and avoid alerting logic for unmanned systems. In: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC). pp. 5A1–1–5A1–12 (Sep 2015)
[27] Neogi, N., Cuong, C., Dill, E.: A risk based assessment of a small UAS cargo delivery operation in proximity to urban areas. In: Proceedings of the 37th Digital Avionics Systems Conference (DASC). London, England, UK (September 2018)
[28] Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 364–380. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
[29] Prevot, T., Callantine, T., Lee, P., Mercer, J., Battiste, V., Johnson, W., Palmer, E., Smith, N.: Co-operative air traffic management: a technology enabled concept for the next generation air transportation system. In: 5th USA/Europe Air Traffic management Research and Development Seminar, Baltimore, MD (June 2005)
[30] Prevot, T., Rios, J., Kopardekar, P., Robinson, J.E., Johnson, M., Jung, J.: UAS Traffic Management (UTM) concept of operations to safely enable low altitude flight operations. In: Proceedings of the 2018 Aviation, Technology, Integration, and Operations Conference. No. AIAA-2016-3292, Washington, DC (June 2016)
[31] Schewe, S.: Synthesis of distributed systems. Ph.D. thesis, Saarland University, Saarbrücken, Germany (2008)
[32] Schewe, S.: Distributed synthesis is simply undecidable. Information Processing Letters 114(4), 203 – 207 (2014)
[33] Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (2000)
[34] Swenson, H., Barhydt, R., Landis, M.: Next generation air transportation system (ngats) air traffic management (atm)-airspace project. Tech. rep., Technical report, National Aeronautics and Space Administration (2006)
[35] Tomlin, C., Pappas, G., Lygeros, J., Godbole, D., Sastry, S., Meyer, G.: Hybrid control in air traffic management systems. IFAC Proceedings Volumes 29(1), 5512–5517 (1996)
[36] Wolff, E.M., Topcu, U., Murray, R.M.: Efficient reactive controller synthesis for a fragment of linear temporal logic. In: 2013 IEEE International Conference on Robotics and Automation. pp. 5033–5040. IEEE (2013)
[37] Zhang, W., Kamgarpour, M., Sun, D., Tomlin, C.J.: A hierarchical flight planning framework for air traffic management. Proceedings of the IEEE 100(1), 179–194 (Jan 2012)

Suda Bharadwaj received B.Sc. and B.E degrees in applied mathematics and aerospace engineering from the University of Sydney, NSW, Australia, in 2014. In 2016, he received an M.S. degree in aerospace engineering from the University of Texas at Austin, TX, USA. He is currently pursuing his Ph.D degree at the Department of Aerospace Engineering and Engineering Mechanics at the University of Texas at Austin. His research interests include the intersection of formal methods, reinforcement learning, and control with a focus on provable safety guarantees.

Steven Carr is currently pursing his Ph.D. degree from the University of Texas at Austin in the Department of Aerospace Engineering. He received the B.Eng./B.Sc. in aerospace and mathematics from the University of Sydney in 2014 and the M.Sc in aerospace engineering in 2018. His research interests include the intersection of control and learning in autonomous systems with a focus on aerospace applications.

Natasha Neogi is currently a Subproject Manager of NASA's System-Wide Safety Project. She is a also a Senior Researcher at the NASA Langley Research Center where she serves as the Certification Technical Lead on the Advanced Air Mobility Project. Her primary research interests are in the verification and validation of software-intensive safety-critical infrastructure systems, as well as certification issues concerning airworthiness of non-conventionally piloted vehicles. Previously, she was a staff scientist in the Office of the Chief Scientist, NASA Headquarters. She received her Ph.D in Aeronautical and Astronautical Engineering from the Massachusetts Institute of Technology. She is an associate fellow of the AIAA and was the recipient of the AIAA Robert A. Mitcheltree and PEC Doug P. Ensor Young Engineer awards. She has numerous awards and publications in AIAA, IEEE and ACM conferences and journals.



Ufuk Topcu is an associate professor at The University of Texas at Austin. He holds the W.A. “Tex” Moncrief, Jr. Professorship in Computational Engineering and Sciences I. He received his B.S. from Bogazici University in 2003, his M.S. from the University of California, Irvine in 2005, and his Ph.D. from the University of California, Berkeley in 2008. Ufuk held a postdoctoral research position at California Institute of Technology until 2012 and was a research assistant professor at the University of Pennsylvania until 2015. His honors include the recipient of the Antonio Ruberti Young Researcher Prize, the NSF CAREER Award and the Air Force Young Investigator Award. His research focuses on the theoretical, algorithmic and computational aspects of design and verification of autonomous systems through novel connections between formal methods, learning theory and controls.