Reinforcement Learning Project Proposal

Joseph Muffoletto, Haroon Mushtaq

March 2022

1 Introduction

Urban Air Mobility (UAM) is the field of research regarding air transportation in urban environments. Due to the projected scale of operations, traditional aerial traffic management techniques (ATM) are not viable. Instead, current research has turned towards increasingly automated ATM solutions. With this increasing automation, researchers must consider ways to ensure the safety-critical nature of computerized flight in urban spaces. A recently proposed solution for safe UAM traffic management utilizes minimum-violation planning (Bharadwaj et. al). While this solution provides theoretical guarantees by verifying a set of LTL safety specifications, its exhaustive nature suffers in scalability.

In this paper, we propose a solution to this issue of scalability by using a reinforcement learning agent to replace parts of the planning (defined in detail in further sections). To ensure that safety standards are maintained, we will keep elements of the original minimum violation planning to restrict the RL agent.

2 Background

2.1 UAM Terminology and Notation

- Verticopter: Aerial vehicle with vertical takeoff and landing capabilities (e.g. drone). Since each verticopter must make a "request" to a vertihub to land, we refer to verticopters as requests.
- Vertiport: Takeoff and Landing locations for verticopters
- Vertihub: Automated controllers responsible for scheduling the landings of verticopters onto vertiports in its airspace. Similar to an air traffic control tower in commercial aviation. Denoted $v_i \in V$, where v_i is the i-th verithub in V, the set of all vertihubs

2.2 Minimum Violation Planning

(will expand this section for the full project)

Minimum violation planning is an approach to automated decision making which guarantees that the generated plan violates the specification less than all other possible plans. For a deterministic weighted transition system, the set of all possible plans is the set of all paths which begin in an initial state and end in a terminating state.

For a fuller explanation: Minimum Violation Planning)

2.3 Baseline Background

The environment defined in our baseline paper is a predetermined set of vertihubs in a discrete sequence of time steps. At each time step, the environment generates a set of $n \geq 0$ requests for each vertihub to schedule.

At each time step, each vertihub performs the following steps:

1. Request Auctioning
2. Request Landing

Request Auctioning: For each vertihub: The vertihub computes which request in its airspace incurs the most violation. It then "broadcasts" this maximally-violating request to each vertihub it is "connected" to (the original paper assumes a fully connected network of vertihubs). Each other vertihub then computes the violation cost of scheduling this request. The vertihub which can schedule this request with the least amount of violation (must be less than the violation incurred in the original vertihub) "accepts" this request into its airspace. The goal of this phase is to minimize global violation - the sum of each vertihubs local violation.

Request Landing: Each vertihub synthesizes a minimally-violating plan to satisfy the requests in its airspace. We assume that each vertihub may land one verticopter per time step (subject to change - we may make it so that each vertihub may land a verticopter per each vertiport in its airspace). The intuitive result of this assumption is that the plan for a vertihub with n requests at time step t will be a sequence of n+1 state-action pairs ending at time step t+n, where "action" refers to the verticopter that was chosen to land. In a sense, at each time step, each vertihub plans its actions for the future based on its current state. If the future does not change (e.g., the environment does not generate any new requests), then the plan, which was previously determined to be minimally-violating, does not require any modification.

After these steps have been performed, a new set of requests for each vertihub are generated by the environment and the process repeats.

2.4 Motivation

The original paper set out to develop a verifiable and scalable method for UAM traffic management. As part of verifying safety, the authors used minimum violation planning, a technique which models each vertihub as a weighted transition system and uses prioritized safety specifications to rank the performance of each possible trace of the system (for more information, see here)

For scalability, the authors took a decentralized approach and split up the traffic management into independent computational units (vertihubs). The consequence of this decision is that the system is no longer "globally" minimized, in that each vertihub tries to minimize its own violation while being unaware of the total violation across all vertihubs. To remedy this, the authors introduced request auctioning, which guarantees that violation will be globally minimized. Unfortunately, this is an incredibly expensive process, and in the fully connected case, is responsible for over 99 percent of the computation time.

2.5 Proposed Solution

As such, we would like to replace the exhaustive request auctioning with an approximate solution, such as a deep neural network. This would greatly reduce computation time, but we would lose our theoretical guarantees of minimal violation. Nevertheless, the goal of our project is to demonstrate that with an approximated request auctioning, we can guarantee that the global violation will never be more than the case where no request auctioning exists at all.

Thus, our proposed solution utilizes both minimum violation planning and reinforcement learning. The process of choosing requests to land will use minimum violation planning, and the process of request auctioning will be done by a reinforcement learning agent.

3 Problem Definition

3.1 State Space

3.1.1 Verticopters - "Requests"

We define a request as a tuple $O = (r, T, c)$ where

- $r \in \mathcal{R}$ is the request class from a predefined set of classes $\mathcal{R}$. Request classes include *landing at a particular vertiport in the region*, *pass-through region*, *take-off from vertiport in the region.* Depending on the nature of the problem, $\mathcal{R}$ can be defined to capture all types of desired outcomes for vehicles.
- $T \in \mathbb{N}$ is the amount of time left before the request *must* be granted. t can function as an analogue for fuel reserves as vehicles requesting to land cannot hover indefinitely.
- $c \in C$ is the class of UAM vehicle from a predefined set of vehicles C.

3.1.2 Vertihubs

We define the state of a vertihub v_i at time t as the current set of requests in its airspace, $s_t^{v_i} = \{O_0, O_1, ..., O_n\}$, where n is the total number of requests for this vertihub at time step t.

3.1.3 Global Planner

(Note: this definition may not be necessary)

Subject to change.

We define the global state at time t as the set of all vertihub states in our environment, $g_t = \{s_t^{v_0}, s_t^{v_1}, ..., s_t^{v_n}\}$, where n is the total number of vertihubs in our system (n = $|V|$, $v \in V$).

3.2 Action Space

3.2.1 Vertihub

NOTE: Our intended "simplest implementation" will not have an RL agent controlling the Vertihubs. This action space is used by the minimum violation planner.

As stated in section 2.2, during the "request landing" phase of each time step, each vertihub synthesizes a plan to satisfy the requests in its system. We define this plan as a sequence of state-action pairs, using the notation $s_t a_t$, where t is the time step corresponding to this state-action pair.

For example, we define the plan for a vertihub with n requests at time t as the following sequence: $\{s_t a_t, s_{t+1} a_{t+1}, ..., s_{t+n} a_{t+n}\}$

Since a vertihub can only land one verticopter per time step, its "action-space" is the set of all requests in its airspace for that time step. As such, the action-space changes over time.

3.2.2 Global Planner

At each time step, the global planner is responsible for performing the request auction process. Given the states of the vertihubs, the planner will select one request to reallocate to another vertihub. It will continue reallocating requests until a vertihub rejects the decision of the global planner. Recall that if a vertihub is offered a request that will incur violation $n \geq$ to the original violation, then it will deny that request. Intuitively, once a denial occurs in an optimal planner, we can safely assume that there are no more request reallocations that would reduce the global violation, and thus the global violation is minimized relative to the current knowledge of the states.

Thus, our action space for a given time step is the product of the set of all requests with the set of all vertihubs.

Concern: since the global planner will take $n \geq 1$ actions and recieve a corresponding number of rewards per time step, is this formulation not sequential?

3.3 Reward Function

(Subject to change)

We want to reward the global planner for reallocations which lower the global violation. We will use the violation computed by the vertihubs as our reward function.

For example, say the chosen request O belonging to vertihub v_i incurs a violation of 2. The planner then chooses a new vertihub v_j to allocate O, which incurs violation of 4. Since the violation is higher, v_j would reject O, and the planner would receive a reward of -2.

Unfortunately, this reward function could potentially incentivize the global planner to only select requests that have a low initial violation (and thus it would be less likely to make a choice which incurs more violation, but also less likely to minimize the global violation). We will need to revise the reward function.

4 Methods

4.1 Algorithm

Our current plan is to train a policy which maps the global state defined in section 3.1.3 to the global action space defined in section 3.2.2. To learn this policy, we will (tentatively) use a policy-gradient RL agent parameterized by a deep neural network.

5 Data

5.1 Baseline

Our baseline comparison for the request auctioning is the existing exhaustive implementation. Since the exhaustive implementation is optimal, our metric for performance is how close the approximated solution can get to it. Moreover, since our motivation was to reduce the computation time of the request auctioning process, we would expect that our approximator is at least faster than the exhaustive implementation.

6 Future Work

Increase each vertihub's action space by allowing for more than one landing per time step.

Approximate planner for the request landing process. Generate a policy which maps the state of the vertihub to the request it should accept for that time step.