A software architecture for autonomous taxiing of aircraft

Yuhao Zhang[*], Guillaume Poupart-Lafarge[†], Huaiyuan Teng[‡], Joshua Wilhelm[†], Jean-Baptiste Jeannin[§], and Necmiye Ozay[¶]
University of Michigan, Ann Arbor, MI 48109, U.S.A.

Eelco Scholte[‖]
Collins Aerospace, Bloomington, MN 55438, U.S.A.

Recent progress in self-driving capabilities of cars suggests that one could also automate aircraft taxi operations, a seemingly easier problem. In this paper we describe a high-level software architecture for self-taxiing, and we identify its specific challenges. The architecture is selected to ease the mapping of specifications to the different implemented functionalities, allowing for modular verification. We then focus on two of the modules in this architecture. We describe how to obtain a low-level list of taxiways from high-level Air Traffic Control instructions, and how to design GPS-based controllers for lateral and longitudinal control of the aircraft. This architecture is implemented in simulation based on the X-Plane flight simulator, for which different controllers for one of the low-level functionalities is evaluated using falsification tools S-TaLiRo and Breach.

I. Introduction

Commercial air transport has become extremely safe, and 2017 was called the safest year on record for passenger air travel, with no commercial passenger jet fatalities that year. As impressive as this feat is, it hides the fact that airline operations still resulted in the death of 35 people on the ground [1]. Therefore, while air travel is extremely safe for passengers, ground workers loading and maintaining aircraft still have a dangerous job. In fact, ground operations involve several types of vehicles operating in close proximity to each other, all driven or taxied by human operators, in an environment where human errors can cause grave accidents.

With the aircraft systems becoming more and more complex, the pilot workload increases proportionally. Autonomous systems can reduce the pilot workload by reducing the amount of information displayed to the pilots and helping them focus on critical tasks. While autopilots have been used for decades for air operations of aircraft, comparatively much less attention has been devoted to automating their ground operations, and today airliners are still taxied by hand from their gate to the runway. Autonomous taxiing of aircraft is attractive because it has the potential of being safer than manual taxiing and avoid runway incursions, taxiing mistakes, as well as collisions with other ground vehicles on the ramp, especially in congested airports where maintaining situational awareness is hard. Additionally, it can free up pilot's minds so they can focus on safety-critical aspects or getting ready for the flight.

The problem of avoiding runway incursions has been well-studied, see [2], and [3] for a survey. If done correctly, self-taxiing could significantly reduce or even eliminate such runway incursions. Recently, Liu and Ferrari [4] have studied a related problem for autonomous taxiing. Their focus is on vision-based algorithms, and their modeling of the airport as a graph is closely related to ours. However, their approach is more quantitative rather than focusing on certification and constraint satisfaction.

A main challenge for aerospace systems, whether it be for in-air or on-ground operations, is to ensure the system meets its specification. Standard processes are defined for both systems and software to ensure any errors are discovered during development. As the autonomy of systems increases, in particular by utilizing algorithms that have adaptive

[*]Master's Student, Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI.
[†]Master's Student, Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI.
[‡]Master's Student, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI.
[§]Assistant Professor, Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI.
[¶]Associate Professor, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI.
[‖]Associate Director, Model Based Systems Engineering Technology, Foundational Capabilities, Enterprise Engineering, Collins Aerospace, Bloomington, MN.

behavior, the verification is non-trivial. As a solution, one approach is to use a correct-by-construction approach combined with the use of formal methods to detect any gaps during design time [5–7].

In this paper, we describe a proposed high-level architecture for autonomous taxiing of aircraft. We describe our implementation of a path planner and controller, that we ran on taxiing examples in the X-Plane flight simulator. Within this architecture, several low-level path following controllers are implemented and their performance is evaluated using falsification techniques to identify corner-cases and the amount of inaccuracy in sensing/perception that different controllers can or cannot handle.

II. Software architecture

A schematic of the proposed autonomous taxiing architecture is shown in Figure 1. At the highest-level of the architecture, we have the Air Traffic Control (ATC) route validation modules. These modules combine information about the airport map and the aircraft taxiway design group to reason about the validity of the command from the ATC. The goal is to ensure that the instructions on the taxiways to be followed are feasible and unambiguous. At the middle-level the route information is combined with the taxiway geometry to generate a path to be followed by the aircraft. The lower-level then aims to control the aircraft motion to ensure that the generated path is followed with a specified accuracy. Finally, there are sensing modules that generate localization and state information for the low-level controllers. In this work, we consider GPS as a sensor that provides precise state information, but the sensing module can be realized by other types of sensors and corresponding perception, sensor-processing algorithms. The architecture can also be expanded to allow for exception handling at the middle and lower-levels if following the ATC-specified route becomes infeasible due to unexpected external factors (e.g., obstacles or other aircraft on the taxiway) or internal failures (e.g., problems with the aircraft subsystems). This last aspect is not considered in the current work.

There are several specifications set by the Federal Aviation Administration (FAA) that pilots should follow with regard to taxiing. These specifications can be summarized as follows:

- *High level specifications:* ATC sends a route, pilot confirms the route and proceeds; if there is a problem (e.g., another plane or ground vehicle blocking the way) in following the route, the pilot gets the plane to a stop and notifies ATC.
- *Low level lateral specifications:* Pilot should ensure “cockpit over centerline" taxiing [8]; they should follow the lane markings to ensure the satisfaction of this specification.
- *Low level longitudinal specifications:* Pilot should obey several speed limits, depending on whether the plane is on a straight taxiway or on a turn. Moreover, he/she needs to put the plane to a full stop before entering a runway (such runway entrances are appropriately marked). FAA guidance also recommends applying brakes intermittently rather than continuously to avoid wear and tear on the brakes and to avoid excessive heating.
- *Sensing specifications:* When this functionality is to be accomplished by an autonomous system, detection of the lane markings and other entities on the taxiways, becomes a crucial part of the functionality. An acceptable autonomous taxiing system should have a reliable sensing and perception system to allow proper localization of the aircraft (and other entities) on the taxiway.

We can map these specifications to different parts of the architecture as shown in Figure 1. The choice of the architecture presents a separation of concerns in terms of specifications. Several emergency handling requirements can also be associated with the low-level lateral and longitudinal control, such as avoiding collisions with unexpected obstacles and with other aircraft, when exception handling modules are added to the architecture.

In what follows, we describe specific components of the architecture for autonomous taxiing together with a prototype implementation that interfaces with the X-Plane flight simulator. This is followed by applying falsification analysis to low-level controllers developed to identify their domain of safety, i.e., the set of initial conditions from which they can ensure the satisfaction of the specifications relevant to their functionality.

III. Air traffic control route validation

When taxiing at an airport with a control tower, pilots have access to two main sources of information: a ground map of the airport, such as the one in Fig. 2 representing the Ann Arbor airport, including all its runways and taxiways; and a list of instructions from ATC, consisting of a list of taxiways, followed by a destination. Given these instructions, the pilot (or an automated system) needs to determine their path to arrive safely and accurately at their destination.

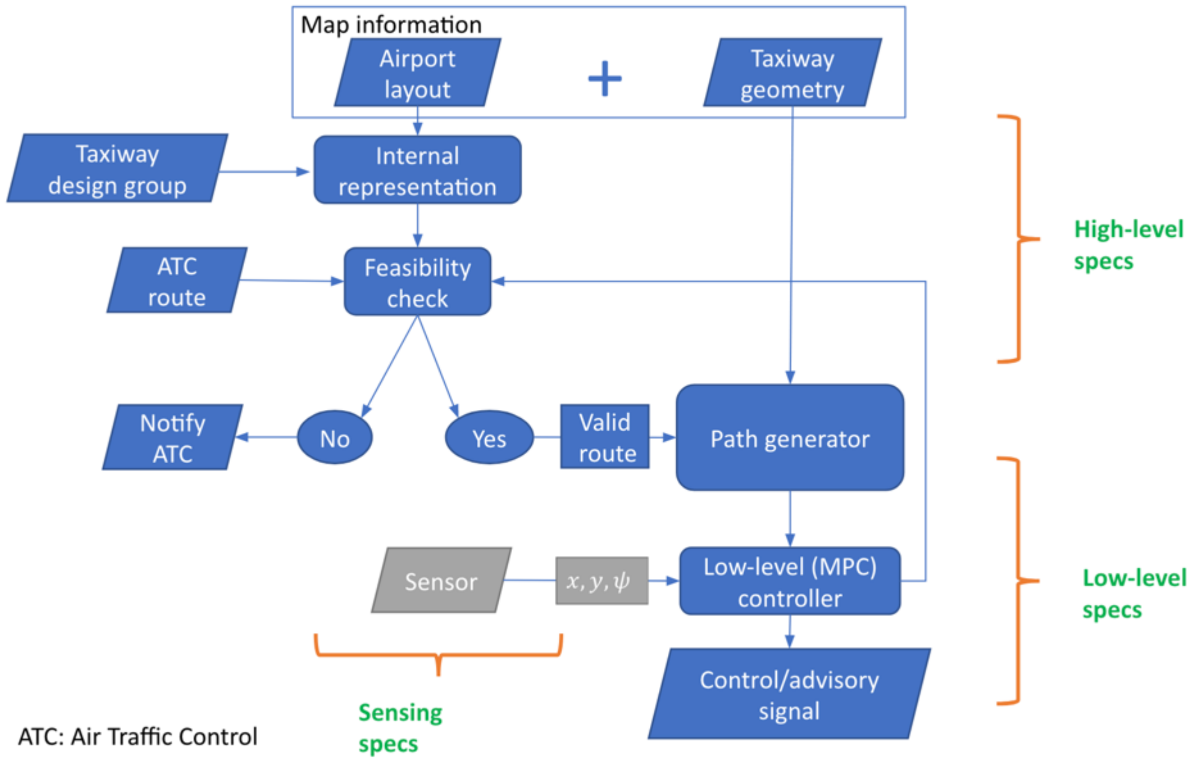


Fig. 1 Proposed architecture

For example, on the layout of Fig. 2, an airplane currently at the North-West hangars (top left, vertex Ch) can receive an instruction from the control to *"Taxi Charlie Bravo Alpha Alpha-3 to holding point runway 6"*, meaning that it should follow taxiways C (Charlie), B (Bravo), A (Alpha) and A3 (Alpha-3) to the beginning of runway 6 (point A3r). The low-level path is to follow taxiway C to point BC, then taxiway B to point AB and taxiway A to point AA3, and finally the short stretch of taxiway A3 to point A3r. The four contiguous taxiway branches followed by the aircraft would thus be Ch-BC, BC-AB, AB-AA3, and AA3-A3r. Determining those low-level taxiway branches from the high-level ATC description automatically, or determining potential problems in the instructions, is the goal of the module developed in this section. The Ann Arbor example might seem trivial, but on bigger airports such as Detroit Metropolitan [10], finding the accurate path may be much more difficult and prone to error.

An important factor is that not every aircraft can go on every taxiway: due to size or weight, bigger aircraft are unable to use smaller taxiways. Taxiways are grouped in *Taxiway Design Groups (TDG)* [8], that are defined by the FAA and determine which aircraft can use which taxiway. The FAA currently defines eight taxiway design groups: TDG-1A, TDG-1B, TDG-2, TDG-3, TDG-4, TDG-5, TDG-6, and TDG-7, from the smallest aircraft to the biggest aicraft. Any aircraft of a certain TDG can use all taxiways rated for this TDG, as well as all taxiways rated for a higher TDG.

On large airports, finding the correct sequence of taxiways to follow adds workload in the cockpit, and thus automating this task enables a reduction of pilot workload, so that more attention can be paid on other tasks. Moreover, there could be human errors in providing the taxiway instructions or in perceiving the instructions: air traffic controllers might make mistakes, pilots might mishear the provided route, or an autonomous system might misinterpret the command. Therefore, it is important to verify the validity of the routing commands provided by ATC.

Our algorithm proceeds in two phases. From the airport map consisting of an undirected graph of taxiways (Fig. 3), the first phase builds a directed expanded graph encoding the possible paths of an airplane (Fig. 4). The second phase then runs an adapted depth-first search algorithm on the directed expanded graph, looking for all possible routes from the current position to the destination that match the instructions of ATC. If the number of routes is different from 1 (either 0 or greater or equal than 2), the algorithm returns an error, as the path is either nonexistent or ambiguous.

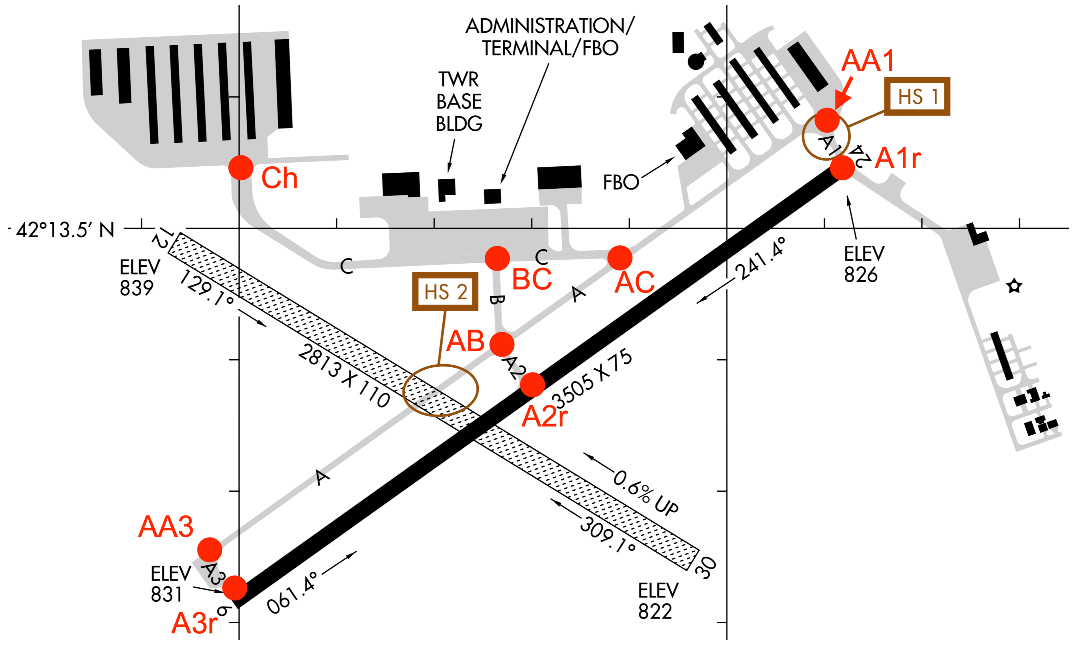


Fig. 2 Official ground map of the Ann Arbor airport (KARB) (FAA documentation [9]), retrieved December 2019). Superimposed on the FAA map are red vertices representing the different taxiway endpoints and intersections, which will become vertices in the undirected graph of the taxiways of the airport (Fig. 3).

Otherwise, the (unique) path that was found is returned by the algorithm. Note that the first phase is independent of the route and can be done once and for all for each airport.

A. Graph representations of airport layout

Given an airport ground layout, a natural way to represent the taxiway structure for a path finding algorithm is as an *undirected* graph G of taxiway branches and intersections. For example, for the Ann Arbor airport ground layout of Fig. 2, the corresponding undirected graph is represented in Fig. 3. For each edge we associate the name of the taxiway, as well as the TDG of the taxiway (not represented on Fig. 3). This undirected graph also encodes input/output vertices that are connected to hangars or the ramp (vertices Ch, BC, A1r, A2r and A3r), marked in Fig. 3 with double circles. However, we cannot run a simple path finding algorithm, because aircraft cannot do a U-turn on taxiways, and because some turns might be too tight for certain aircraft depending on their TDG.

We therefore build a *directed expanded* graph D preventing routes requiring an aircraft to turn around. We also remove from D the taxiways that are not compatible with our aircraft's taxiway design group. We can build this directed graph from the undirected graph using Algorithm 1 (assuming all turns to another taxiway are legal, but not U-turns). In the future, we hope that this directed graph can also be directly provided by the airport authority or the FAA, to include forbidden turns, depending on the aircraft's TDG.

The directed expanded graph of Ann Arbor airport is presented in Fig. 4. Intuitively, for two vertices u and v of the undirected graph G, the aircraft is on the vertex "u from v" of D when the aircraft is on vertex u and just arrived from vertex v. Keeping track of where the aircraft came from enables the prevention of U-turns. Similarly, the aircraft is on the vertex "u in" of D when the aircraft just entered the input vertex from the runway or ramp.

Algorithm 1 first removes all taxiways from G that are not compatible in terms of taxiway design groups. Then, following the intuition above, the algorithm adds a vertex "v in" in D for every input vertex v of G, and a vertex "u from v" in D for every edge (u, v) of G: an aircraft can only be in vertex "u from v" if it arrived to u from v, hence only if u and v are connected in G. Finally, an aircraft can only arrive in vertex "v from u" if it was previously in u, that is in any vertex of the form "u in" or "u from w" in D, provided $w \neq v$ to explicitly avoid U-turns. For example, on the directed

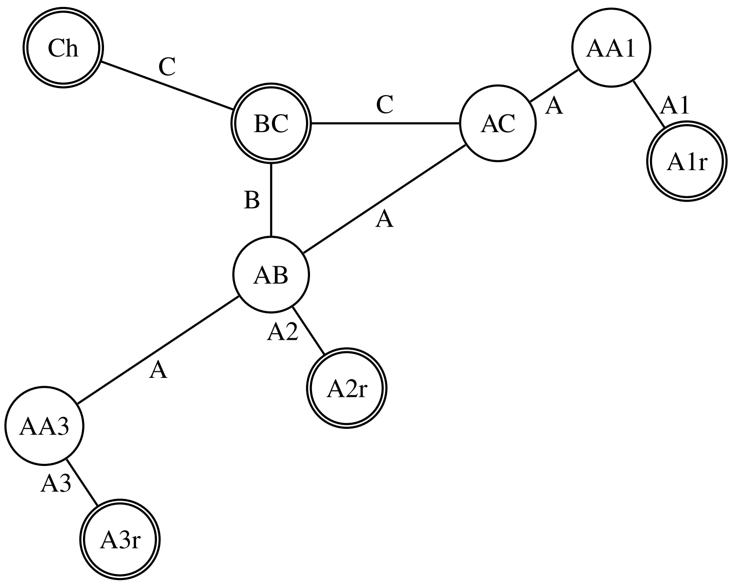


Fig. 3 Undirected graph of the taxiways at Ann Arbor airport (KARB). Input vertices (connected to a ramp or hangars on Fig. 2) are represented with double circles. Note that we chose not to make AA1 an input vertex to not complicate Fig. 4.

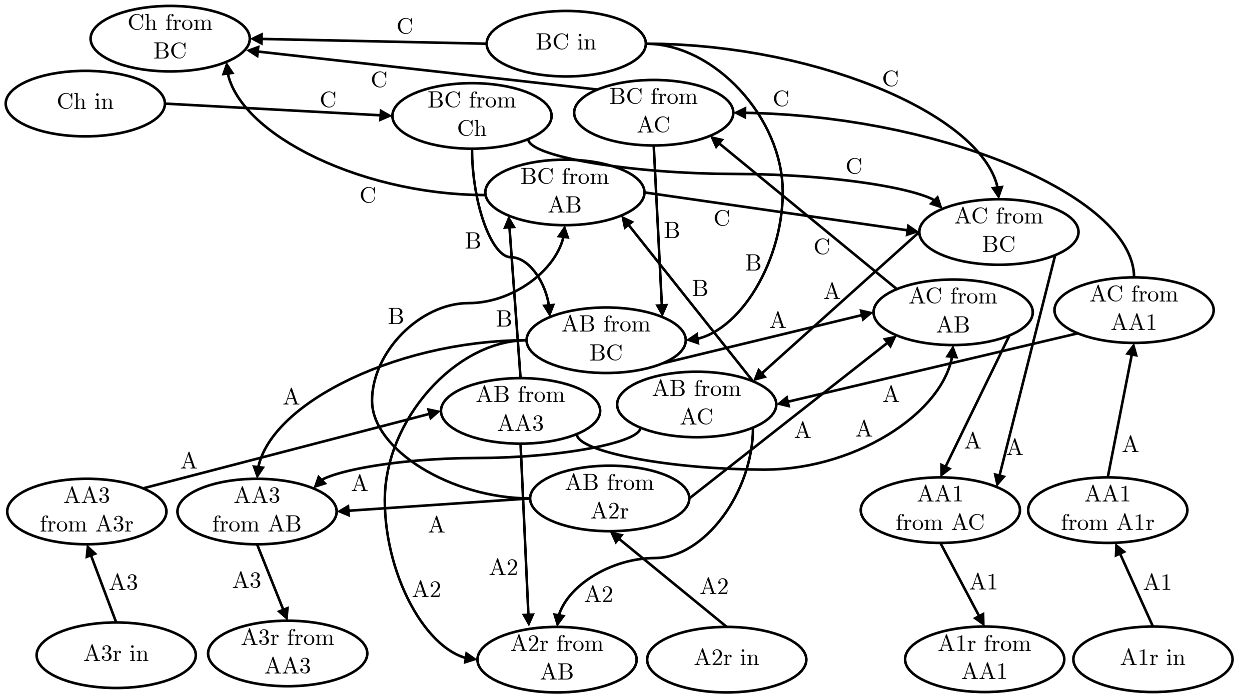


Fig. 4 Directed expanded graph of the taxiways at Ann Arbor airport (KARB).

```
DIRECTED-EXPANDED-GRAPH($G$, tdg)
    Data: undirected graph $G$, aircraft's taxiway design group tdg
    Result: directed expanded graph $D$
    foreach edge $e \in G$ do
        if tdg(e)<tdg then remove $e$ from $G$;
    end
    create $D$ empty directed graph
    foreach input vertex $v \in G$ do
        add vertex "$v$ in" to $D$
    end
    foreach vertex $u \in G$ do
        foreach vertex $v \in G$ connected to $u$ do
            add vertex "$u$ from $v$" to $D$
        end
    end
    foreach undirected edge $e = (u, v) \in G$ with taxiway name $n$ do
        foreach vertex $p \in D$ of the form "$u$ in" or "$u$ from $w$" for $w \neq v$ do
            add directed edge ($p$, "$v$ from $u$") to $D$ with name $n$
        end
        Symmetrically, foreach vertex $p \in D$ of the form "$v$ in" or "$v$ from $w$" for $w \neq u$ do
            add directed edge ($p$, "$u$ from $v$") to $D$ with name $n$
        end
    end
    return $D$
```

Algorithm 1: Generation of directed expanded graph

expanded graph of Ann Arbor airport, the vertex "AB from BC" is accessible from exactly three vertices: "BC in", "BC from Ch" and "BC from AC", because the undirected graph of the airport contains an edge from BC to AB; but note that there is no edge from vertex "BC from AB" to vertex "AB from BC". The edges of D are added accordingly, keeping the same names as in G.

B. Path finding algorithm

From the directed expanded graph G and an instruction of ATC in the form of a list of taxiway names, we now deduce the explicit path to be followed by the pilot. We note that such an explicit path should exist and also be unique, otherwise the algorithm should return an error. To achieve this, we build a modified recursive depth-first search on the directed expanded graph, described in Algorithm 2. At each recursive call, the algorithm keeps track of current and destination vertices and the list of remaining taxiway names, but also of the current taxiway name. Keeping track of the current taxiway name enables the repetition of several taxiway branches with the same name, an essential feature of the algorithm. We assume that inputs and outputs can be a normal value or the special value *None*, which stands for no value. We use *None* to encode that there is no `current-taxiway-name` initially, and as a result when the Algorithm has not found a path in this recursive branch. In practice *None* can typically be implemented using an option type (in functional languages) or the NULL pointer (in C or C++).

Once we get to a vertex, there are three possibilities: either we have reached our destination and there are no more taxiways to follow, in which case the empty list of edges should be returned (case (a)); or we stay on another branch of the current taxiway, with the same name (case (b)); or we turn to a different taxiway (case (c)). In the latter two cases the algorithm steps one edge and calls itself recursively. We keep track of all valid possibilities in a variable `results`, and only return a valid result if this variable is a singleton. If it stays empty, then this branch of the algorithm did not find a path; more importantly, if `results` has more than two elements, we found at least two paths, meaning that the path is not unique and the algorithm should return an error.

```
FIND-PATH-AUX($D$, current, destination, taxiway-names, current-taxiway-name)
    Data: directed expanded graph $D$, vertex current$\in D$, vertex destination$\in G$, ATC instructions as a list of
          taxiway names taxiway-names, current taxiway current-taxiway-name
    Result: route as a list $\ell$ of edges of $D$
    create results := $\emptyset$
    if vertex current is of the form "destination from w" and taxiway-names = [ ] then
        // case (a) where we have arrived at our destination
        add the empty list [ ] to results
    end
    foreach edge $e$ = (current, $v$) $\in G$ do
        if current-taxiway-name$\neq$None and name(e)=current-taxiway-name then
            // case (b) where we stay on the same taxiway
            create res := FIND-PATH-AUX ($D$, $v$, destination, taxiway-names, current-taxiway-name) ;
            if res$\neq$None then add $e$::res to results ;
        end
        if name(e)=head(taxiway-names) then
            // case (c) where we turn to a different taxiway
            create res := FIND-PATH-AUX ($D$, $v$, destination, tail(taxiway-names), head(taxiway-names)) ;
            if res$\neq$None then add $e$::res to results ;
        end
    end
    if results = $\emptyset$ then return None ;
    else if results = $\{\ell\}$ then return $\ell$ // size(results)= 1;
    else return Error "Invalid ATC route" // size(results)$\geq$ 2 ;
```

Algorithm 2: Path Finding: Auxiliary Recursive Algorithm

```
FIND-PATH($G$, start, destination, taxiway-names, tdg)
    Data: undirected graph $G$, vertex start$\in G$, vertex destination$\in G$, ATC instructions as a list of taxiway
          names taxiway-names, aircraft's taxiway design group tdg
    Result: route as a list $\ell$ of edges
    $D$ := DIRECTED-EXPANDED-GRAPH($G$, tdg)
    res := FIND-PATH-AUX($D$, start, destination, taxiway-names, None)
    if res$\neq$ None then return res;
    else return Error "Invalid ATC route";
```

Algorithm 3: Path Finding: Main Algorithm

Finally, Algorithm 3 puts together Algorithms 1 and 2, by calling 2 at the highest level with the current taxiway set to None (to ensure the first taxiway actually has an edge). If it does not find a valid path, it returns an error.

At the end of this step, the algorithm returns a list of taxiway branches, in order, that can be followed to go from the origin point to the destination point. This list is passed onto a low-level path planner to be refined into a set of waypoints on the taxiways to be followed as described in Section IV. If such a list does not exist, or if it is not unique, the algorithm returns an error, so that the pilot can call ATC on the radio and clarify.

C. Implementation and discussion

We have implemented our ATC route validation algorithm in Python, and have tested it on airports of different sizes, including Ann Arbor (KARB) and Willow Run (KYIP). The algorithm performs well in practice, and can run on airports of different sizes with computation times in a few seconds at most.

It might seem at first glance that a simpler path finding algorithm running directly on the undirected KARB graph, and just avoiding going back to a vertex just previously visited, might be simpler and as effective. However, our approach is more general. For example, the paths BC-AC-AB is explicitly allowed on Ann Arbor Airport, even though it involves a rather tight turn. Such a tight turn might be disallowed on some other airports, or not be allowed for certain TDGs. Our algorithm can easily encode this fact by simply removing edge A from vertex "AC from BC" to vertex "AB from AC" in Fig. 4. Symmetrically, if path AB-AC-BC was not allowed, our algorithm could easily encode this fact by removing edge C from vertex "AC from AB" to vertex "BC from AC".

IV. Path planner and low-level controllers

Once a verified route is obtained from the high-level, this information is used at the planning layer to generate a set of waypoints that can be sent to a low-level tracking controller. At this point we use a priori map information for the aircraft that includes taxiway geometry together with the current position of the aircraft. Taxiways and runways are made up of straight segments and curved parts (e.g., centerline while changing taxiways) with constant radius R. For airports of interest, the map is abstracted using this information and prestored in the system. Waypoints are essentially a consecutive sequence of positions along the centerline from current position to destination through the given route. The waypoint data structure also contains, for each point, the curvature of the centerline at that point and the width of the taxiway (which is relevant to taxiway design groups that determine what type of aircraft can use which taxiways [8]). Figure 5 shows a sample path generated through this process.

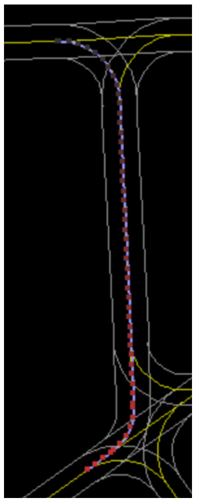


Fig. 5 Visualization of the abstracted view of a portion of the Ann Arbor airport together with a generated path (shown with red dots).

Once a path (set of waypoints) is generated, a lower-level tracking controller is used to track this path. The low-level controllers are separately designed for lateral and longitudinal dynamics. For the design of the tracking controller for longitudinal dynamics, we use the following model:

$$\frac{d}{dt}\underbrace{\begin{bmatrix} v \\ s \end{bmatrix}}_{x_{long}} = \underbrace{\begin{bmatrix} -\frac{f_1}{m} & 0 \\ 1 & 0 \end{bmatrix}}_{A_{long}} \begin{bmatrix} v \\ s \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{1}{m} \\ 0 \end{bmatrix}}_{B_{long}} F_w + \begin{bmatrix} -\frac{f_0}{m} \\ 0 \end{bmatrix}. \tag{1}$$

The system states consist of the aircraft's longitudinal velocity (v) and taxied distance along the taxiways (s). Control input F_w represents the net force acting on the aircraft. Finally, constants m, f_0 and f_1 are parameters of the model. The values of these parameters and the bounds of the variables can be found in Table 1. In particular, the domain the dynamics are defined on is $X_{long} := [v^{min}, v^{max}] \times [s^{min}, \infty)$.

As for specifications for longitudinal dynamics we impose different speed profiles for different curvatures (as per FAA recommendations), faster travel through straight paths and slower speed through turns. In particular, we aim to taxi as fast as possible while not exceeding FAA recommended speed limits for different types of taxiways. Accordingly, we have implemented two different controllers for this purpose:

1) For the first controller, we consider a hybrid proportional-integral (PI) controller, defined as:

$$\begin{aligned} u(t) &= -k_P(v(t) - v_{PI}^{des}(\kappa)) - k_I e(t), \\ e(t) &= \sum_{\tau=0}^{t}(v(\tau) - v_{PI}^{des}(\kappa)). \end{aligned} \tag{2}$$

where $e(t)$ is the error state and k_I is the integral coefficient. The output u of the controller is saturated to

Table 1 Parameter values for the longitudinal dynamics model

Param.	Description	Value
m	aircraft taxi weight	2182 (kg)
f_0	friction/drag term	71.28 (N)
f_1	friction/drag term	1.2567 (Ns/m)
v^{min}	minimum longitudinal velocity	0 (m/s)
v^{max}	maximum longitudinal velocity	15 (m/s)
s^{min}	minimum taxied distance	0 (m)
F_w^{max}	maximum force	10000 (N)
F_w^{min}	minimum force	−8728 (N)
$v^{des,straight}$	desired longitudinal velocity of straight taxiway	5 (m/s)
$v^{des,curve}$	desired longitudinal velocity of curved taxiway	2 (m/s)
$\tau^{min,brake}$	minimum braking horizon	1 (s)
r^{des}	desired yaw rate	0.1 (rad/s)

compute the input to the system by $F_w = sat_{F_w^{min}}^{F_w^{max}}(u)$[*]. The desired longitudinal velocity $v_{PI}^{des}(\kappa)$ is defined by:

$$v_{PI}^{des}(\kappa) = \begin{cases} v^{des,straight} & \text{if the current path is straight} \\ v^{des,curve} & \text{if the current path is curved.} \end{cases} \tag{3}$$

2) We also consider a Model-predictive controller (MPC) that incorporates specifications related to braking. The braking specifications include a minimal braking horizon $\tau^{min,brake}$, which means once the brakes are used, they should be engaged for a minimum of $\tau^{min,brake}$ seconds. This is encoded within the MPC constraints and the following optimization problem is solved online at each sampling time:

$$\begin{aligned} min \quad & \sum_{t=0}^{T} ||v(t) - v^{des,straight}|| \\ s.t. \quad & Time - discretized\ longitudinal\ dynamics \\ & v(t) \le v_{MPC}^{max}(s(t)),\ t = 0, ..., T, \\ & x_{long}(0) = x_0, \\ & F_w^{min} \le F_w(t) \le 0.1 F_w^{min}\ or\ 0 \le F_w(t) \le F_w^{max},\ t = 0, ..., T-1, \\ & F_w(t) < 0,\ t = 0, ..., bstep - 1, \\ & if\ \exists t \in \{0, 1, \dots, T-2\}\ s.t.\ F_w(t) \ge 0\ and\ F_w(t+1) < 0, \\ & \qquad then\ F_w(\tau) < 0,\ \tau = t+1, ..., min(T, t + \tau^{min,brake,step}), \\ & if\ bstep = -1\ and\ F_w(0) < 0, \\ & \qquad then\ F_w(\tau) < 0,\ \tau = 1, ..., min(T, \tau^{min,brake,step} - 1). \end{aligned} \tag{4}$$

where x_0 is the initial condition, T is the length of the prediction horizon. The sampling period is chosen to be $\Delta t = 0.1s$, and the braking steps is defined as $\tau^{min,brake,step} \doteq \lceil \tau^{min,brake}/\Delta t \rceil$. The variable $bstep$ is an input to the MPC and denotes the remaining brake steps (within the $\tau^{min,brake,step}$ limit) if the system used the brakes in the previous time instant(s) or set to -1 if braking was not used in the preceding time step. Since F_w^{min} is negative, the constraint on $F_w(t)$ implies that the braking would not be too light when it is used to avoid wear and tear. To capture the dependence of maximum longitudinal velocity to the curvature κ of the path at each control

[*]Throughout the text, we use the following notation for the saturation function: $sat_a^b(x) \doteq \max(a, \min(b, x))$.

interval t, we use a prediction of which part of the taxiways the aircraft will be at and use the curvature of that path to compute the $v_{MPC}^{max}(t)$, i.e., $v_{MPC}^{max}(t)$ is defined as:

$$v_{MPC}^{max}(s(t)) = \begin{cases} v^{des,straight} & \text{if the path at control interval } t \text{ is straight} \\ sat_{v^{des,curve}}^{v^{des,straight}}\left(\frac{r^{des}}{|\kappa(s(t))|}\right) & \text{if the path at control interval } t \text{ is curved} \end{cases}. \tag{5}$$

To avoid sharp turns, we use the inverse of curvature κ multiplied by r^{des} to set desired absolute yaw rate around r^{des} (rad/s). We also use a saturation function to set a minimum and maximum allowed velocity to make sure for extreme values of κ the aircraft will not be forced to stop or exceed the speed limit. Finally, the constraints in the MPC problem in (4) are converted to a set of mixed integer linear constraints by introducing binary variables to capture the logical constraints and if-then statements.

The variants of the longitudinal controllers we tested are shown in Table 3. Qualitatively, the best performing controller was MPC$_{long}^{30}$, which is used to generate the simulation videos. However, computation time increases roughly linearly with the MPC horizon so this controller was about 3 times slower in computing a control action compared to MPC$_{long}^{10}$. The PI controller completely ignored the specifications regarding braking duration but it did not lead to any computational overhead either. One interesting observation is that when we omitted the braking constraints, we have experienced flat tire events in our X-Plane implementation (cf. Section IV.A) when the aircraft repeatedly taxied for the purposes of falsification. Braking constraints, as per FAA guidance, significantly reduces such incidents.

In addition to longitudinal control, we design lateral controllers to follow the centerline. The following model is used for the lateral dynamics of the aircraft:

$$\frac{d}{dt}\underbrace{\begin{bmatrix} y \\ v \\ \Delta\psi \\ r \end{bmatrix}}_{x_{lat}} = \underbrace{\begin{bmatrix} 0 & 1 & v_0 & 0 \\ 0 & -\frac{C_{af}+Car}{mv_0} & 0 & \frac{bC_{ar}-aCaf}{mv_0} - v_0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{bC_{ar}-aCaf}{I_z v_0} & 0 & -\frac{a^2C_{af}+b^2Car}{I_z v_0} \end{bmatrix}}_{A_{lat}} \begin{bmatrix} y \\ v \\ \Delta\psi \\ r \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{C_{af}}{m} \\ 0 \\ a\frac{C_{af}}{I_z} \end{bmatrix}}_{B_{lat}} \delta_f + \underbrace{\begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}}_{E_{lat}} r_d \tag{6}$$

where the four states are: lateral deviation from the center of the taxiway lane (y), the lateral velocity (v), the yaw-angle deviation in road-fixed coordinates ($\Delta\psi$), and the yaw rate (r). The input δ_f is the steering angle of the front gear, which is limited to lie within θ_s^{min} and θ_s^{max}. And, the external disturbance/reference term r_d is the required yaw rate computed by $r_d = v_0\kappa$ where κ is the taxiway curvature and v_0 is the aircraft's longitudinal velocity. Other parameters include m, the total taxi mass of the aircraft, and a, b, C_{af} and C_{ar}, are the aircraft's geometry and tire parameters. These values can be found in Table 2. And accordingly, the domain of the dynamics is $X_{lat} := [-y^{max}, y^{max}] \times [-v^{max}, v^{max}] \times [-\Delta\psi^{max}, \Delta\psi^{max}] \times [-r^{max}, r^{max}]$.

For the specification regarding lateral dynamics, we aim to achieve "cockpit over centerline" taxiing.

We explore two types of tracking controllers:

1) A hybrid proportional (P) controller defined as:

$$u = K_P^\top x_{lat}. \tag{7}$$

The proportional gain $K_P^\top$ is tuned to give good performance for the complex dynamics in X-Plane. The control input δ_f is obtained by saturating u to account for the physical limits of the actuators, i.e., $\delta_f = sat_{\theta_s^{min}}^{\theta_s^{max}}(u)$.

2) An MPC controller with a linearized discrete-time model and a sampling period of 0.1s using the following formulation:

$$\begin{aligned} min \quad & \sum_{t=0}^{T} x_{lat}(t)^\top Q x_{lat}(t) + 2u^2(t) \\ s.t. \quad & Time - discretized\ lateral\ dynamics \\ & \theta_s^{min} \le u(t) \le \theta_s^{max},\ t = 0, ..., T-1, \\ & x_{lat}(0) = x_0. \end{aligned} \tag{8}$$

Table 2 Parameter values for the lateral dynamics model

Param.	Description	Value
m	aircraft taxi weight	2182 (kg)
I_z	aircraft moment of inertia	15000 (kgm^2)
a	aircraft geometry parameter	2.44 (m)
b	aircraft geometry parameter	0.44 (m)
C_{af}	tire parameter	59128 (N/rad)
C_{ar}	tire parameter	347409 (N/rad)
y^{max}	maximum lateral deviation	1 (m)
v^{max}	maximum lateral velocity	1 (m/s)
$\Delta\psi^{max}$	maximum yaw-angle deviation	0.2 (rad)
r^{max}	maximum yaw rate	0.2 (rad/s)
θ_s^{min}	minimum steering angle	-0.6981 (rad)
θ_s^{max}	maximum steering angle	0.6981 (rad)

where T is the length of the prediction horizon, x_0 is the initial condition and $Q = \text{diag}([0.5\ 0\ 4\ 0])$. The dynamics in (6) has the required yaw rate $r_d(t)$ as an input. While implementing the time-discretized version of the dynamics within MPC, $r_d(t)$ over the horizon $t = 0, 1, ..., T-1$ is estimated using the current longitudinal velocity and the curvature information about the path ahead from the waypoint generator. Since the constraints in the MPC formulation already include the input saturation, control input is set as $\delta_f = u(0)$.

We use a Baron 58 aircraft model (in X-Plane) in our initial tests, and the longitudinal and lateral models used by MPC are tuned to be compatible with this aircraft. Those tuned model parameters can be found in the upper parts of Table 1 and Table 2. Specifically, the aircraft's taxi weight, friction/drag terms, moment of inertia, geometry parameters and tire parameters are computed according to the aircraft's parameters in X-Plane.

Table 3 Controllers used in our tests

	Controller (parameters)	Notation	Parameter
Longitudinal dynamics	PI controller (P/I gains)	PI$_{long}$	$k_P = 1$, $k_I = 0.1$
	MPC (horizon)	MPC$^{10}_{long}$	10
		MPC$^{20}_{long}$	20
		MPC$^{30}_{long}$	30
Lateral dynamics	Proportional controller (P gain)	P$_{lat}$	$K_P = [0.1; 0; 5; 0]$
	MPC (horizon)	MPC$^{2}_{lat}$	2
		MPC$^{5}_{lat}$	5
		MPC$^{20}_{lat}$	20

A. Simulation setup

The overall system is implemented by integrating several modules. We use X-Plane as the aircraft simulator. The low-level controllers are implemented in Matlab Simulink, while the X-Plane Connect Toolbox [11] is used for interfacing with X-Plane. In our current implementation, the low-level controllers use GPS data from X-Plane (corrupted by noise to add realism) for state feedback. A video of an X-Plane simulation where our controllers are taxiing a Baron 58 aircraft on Ann Arbor airport can be found here `https://youtu.be/NcBEo81lKG4`.

In future work, we would like to switch from a GPS-based controller to a vision-based controller, as the former does not allow to detect and avoid service vehicles or unexpected obstacles during ground operations. This will be achieved by exporting real-time camera feeds from X-Plane and adding computer vision – image processing modules for state estimation, object detection and localization. We also plan to explore the use of vision, instead of GPS, for localization on the taxiway to estimate lateral deviation from the centerline and yaw-angle.

V. Falsification-based analysis of different controllers

Our proposed architecture is designed in a way to facilitate verification of different components. In this section, we demonstrate how falsification techniques can be employed to evaluate different lateral tracking controllers we designed at the low level. Therefore we focus on the safety specification requirement - "cockpit over centerline" taxiing. Mathematically, we want the following to hold:

$$\varphi_{lat} := \forall t: \ |y(t)| \leq y^{max}. \tag{9}$$

This is an invariance property and can also be represented by the temporal operator "always" in linear temporal logic [12]. In what follows, we use two different falsification tools to count the violations of the above specification when different controllers for the lateral dynamics are employed.

Although we explore four different controllers for longitudinal dynamics, the controller PI$_{long}$ described in Table 3 is used to control the longitudinal dynamics for all of the following results. This choice was made to decrease the simulation time, as the use of the longitudinal PI controller when compared to the longitudinal MPC controllers decreases the simulation time by roughly a factor of 3. As the specification (9) that the falsification tools attempt to violate is a specification for the lateral dynamics and the simulation is run on a straight segment of the taxiway, the choice of longitudinal controller does not have a large impact on the results of this section.

1. S-TaLiRo

We use S-TaLiRo, a falsification tool that is proposed in [13], to find falsifying initial conditions and disturbance trajectories that lead to a violation of φ_{lat}. The block-diagram in Figure 6 summarizes how S-TaLiRo and our system model are interfaced.

To evaluate the performance of the different controllers, we run S-TaLiRo multiple times to see how many times it can find falsifying initial conditions and disturbance trajectories. We restrict the initial conditions that S-TaLiRo can choose by letting $y_0 \in [-y^{max}, y^{max}]$, $\Delta\psi_0 \in [-0.15, 0.15]$ and the remaining states in lateral dynamics are initialized to constants: $v_0 = 0$, $r_0 = 0$. In addition, we impose bounds on the external disturbance inputs, i.e., $d(t) \in D$ for all t where $D := [-y^{max}/2, y^{max}/2] \times [-v^{max}/2, v^{max}/2] \times [-\Delta\psi^{max}/2, \Delta\psi^{max}/2] \times [-r^{max}/2, r^{max}/2]$. We initialized the aircraft on a straight segment of the taxiway in the Ann Arbor airport near the point Ch in Fig. 2, and ran the simulation for 3 s.

While S-TaLiRo can jointly search for initial conditions and disturbance inputs, the search spaces for initial conditions and disturbance trajectories are not comparable, the latter being much larger. Given a fix number of iterations, which acts as a timeout condition, it is hard to separate the impact of disturbance from the hardness of the search using S-TaLiRo. To overcome this issue and to understand the impact of initial conditions and disturbance trajectories on the control performance separately, we consider the following evaluation strategy. We take a set of initial conditions $y_0 \in [-1, 1]$, $\Delta\psi_0 \in [-0.15, 0.15]$ and mesh it into a 21×21 grid. Then by setting disturbance values to zero, we count how many initial conditions on this grid lead to a violation of the specification φ_{lat}. In addition, we run S-TaLiRo with those non-falsifying initial conditions to see whether it can find falsifying disturbance trajectories. The number of samples S-TaLiRo can try to find a falsifying trajectory for each initial condition is set to 100. We impose two different bounds on the external disturbance input, i.e., D is the same as defined before and

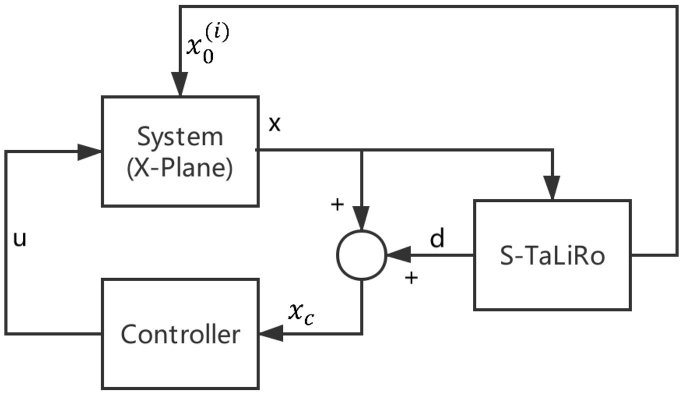


Fig. 6 Main framework. The output from system model (X-Plane flight simulator) are used by S-TaLiRo to find falsifying initial conditions ($x_0^{(i)}$) and disturbance trajectories (d). The generated initial conditions are sent to X-Plane for placing the aircraft at desired initial position. And the disturbances representing noise in sensory measurements are added to the state values from system model. Thus, $x_c = x + d$, where x_c is the state vector observed by the controller and x is the true state vector from X-Plane's data.

$D_{1/2} := [-y^{max}/4, y^{max}/4] \times [-v^{max}/4, v^{max}/4] \times [-\Delta\psi^{max}/4, \Delta\psi^{max}/4] \times [-r^{max}/4, r^{max}/4]$. Table 4 summarizes the falsifying initial condition rates using this method, while Figures 7–10 show the distribution of non-falsifying and falsifying grids. Red crosses indicate falsifying initial conditions even when there is no disturbance, black diamonds indicate initial conditions for which S-TaLiRo finds a falsifying disturbance trajectory and green dots indicate the remaining non-falsifying initial conditions.

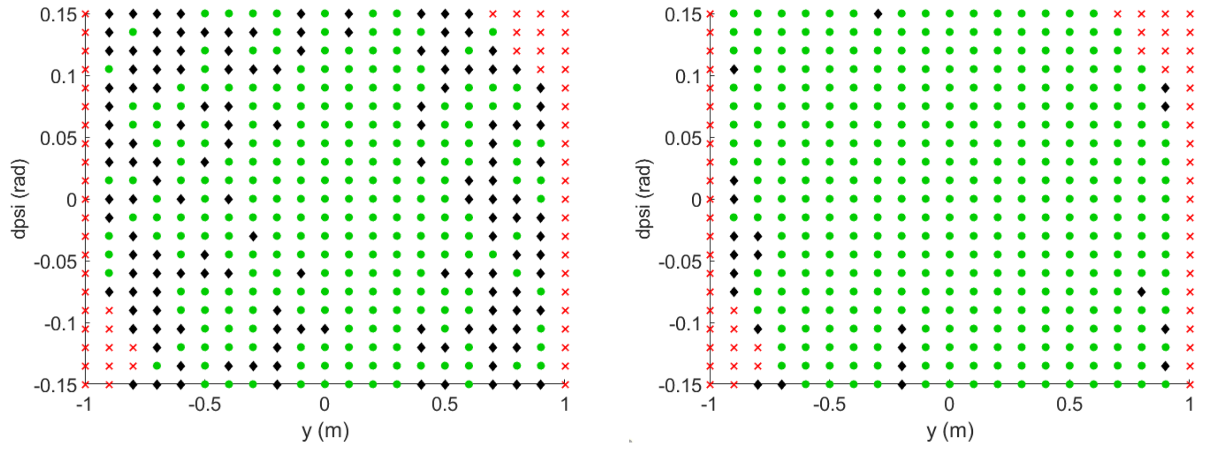
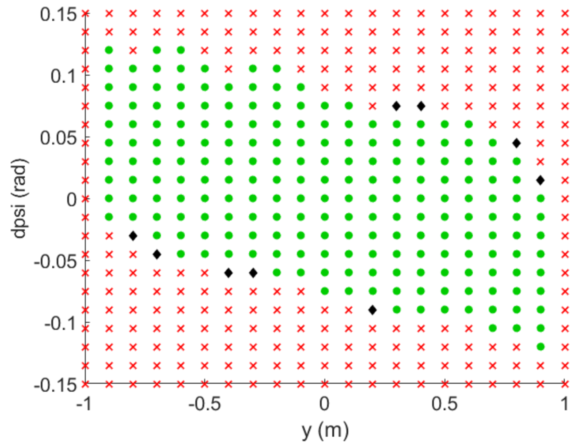


Fig. 7 S-TaLiRo: Distribution of initial condition grids for controller P$_{lat}$, with disturbance domains D (left) and $D_{1/2}$ (right).

Table 4 shows that with a smaller disturbance domain $D_{1/2}$, S-TaLiRo finds fewer falsifying disturbance trajectory as expected. Another observation from the distribution figures is the red crosses all locating on the boundaries of the plots which indicates that they are basically "hard" initial conditions for the responding controllers because they are still falsified even there is no environmental disturbance added to the system. This makes sense as the upper right and lower red corners correspond to cases where the deviation from the centerline is large and the aircraft is pointing



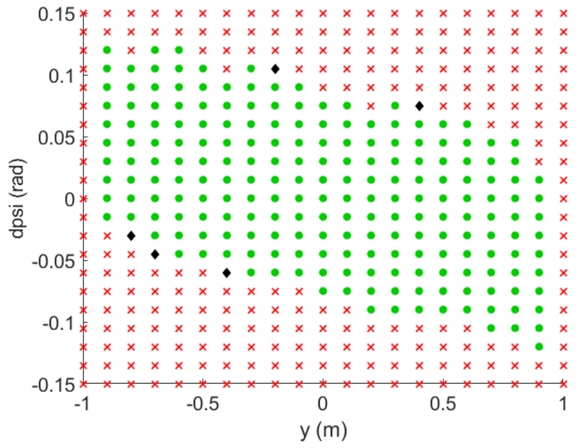


Fig. 8 S-TaLiRo: Distribution of initial condition grids for controller MPC$^2_{lat}$, with disturbance domains D (left) and $D_{1/2}$ (right).

Table 4 S-TaLiRo: Lateral dynamics falsification rates. The entire set of initial conditions N has 441 grid points. N_0 is the set of falsifying initial conditions without disturbance while N_n^1 and N_n^2 stand for the sets of initial conditions for which S-TaLiRo can find falsifying disturbance trajectories within domain D and $D_{1/2}$, respectively. When calculating N_n^1 and N_n^2, we only use non-falsifying initial conditions without disturbance, which is $N \backslash N_0$.

Controller	**size of N_0 ($\frac{N_0}{N} \times 100\%$)**	**size of $N_n^1 + N_0$ ($\frac{N_n^1+N_0}{N} \times 100\%$)**	**size of $N_n^2 + N_0$ ($\frac{N_n^2+N_0}{N} \times 100\%$)**
P$_{lat}$	61 (13.8%)	202 (45.8%)	83 (18.8%)
MPC$^2_{lat}$	229 (51.9%)	238 (54.0%)	234 (53.1%)
MPC$^5_{lat}$	58 (13.2%)	85 (19.3%)	78 (17.7%)
MPC$^{20}_{lat}$	49 (11.1%)	193 (43.8%)	112 (25.4%)

outwards from the certerline. Therefore, maneuvering without getting out of the taxiway in these scenarios is hard. The controller MPC$^2_{lat}$ has very poor performance. The MPC design MPC$^5_{lat}$ seems to be the safest among all these controllers as we can see both MPC$^{20}_{lat}$ and P$_{lat}$ have poor performance when there exists some external disturbance. The mismatch between the lateral dynamic models used in MPC design and X-Plane simulator could explain the performance degradation in MPC for larger horizons (cf. MPC$^5_{lat}$ and MPC$^{20}_{lat}$). MPC uses the dynamic model of the plant it controls to predict the future output which is later used together with the desired output trajectory and set of constraints in the calculation of the optimal controller output. So with the model plant mismatch, the errors between predicted future output and real future output become much larger for MPC$^{20}_{lat}$. Another interesting observation is that we ran the same experiments with a linear model for aircraft dynamics (instead of the X-Plane for dynamics simulator) in which case we observed that the performance of MPC$^{20}_{lat}$ is better than that of MPC$^5_{lat}$ when the dynamics matches identically to the one used in design. Finally, since the path segment considered is straight and the aircraft is (almost) symmetric, we expect the set of initial conditions that are falsified to be symmetric around the origin (i.e., being on the far left of the certerline is as hard as being on the far right with the opposite orientation). This symmetry is slightly broken in some of the figures, which might be due to the random nature of the falsification algorithm in S-TaLiro or the non-determinism in X-Plane.

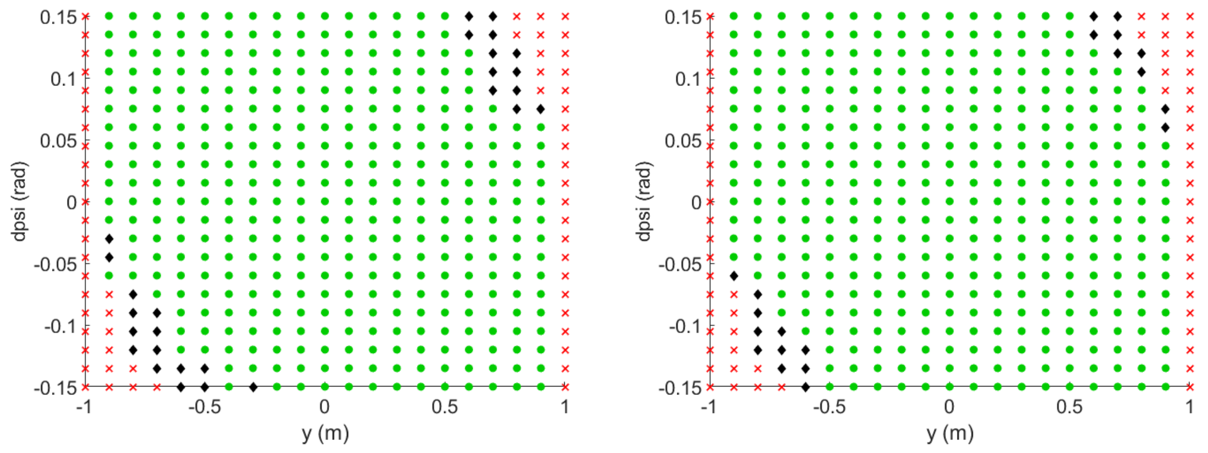


Fig. 9 S-TaLiRo: Distribution of initial condition grids for controller MPC$^5_{lat}$, with disturbance domains D (left) and $D_{1/2}$ (right).

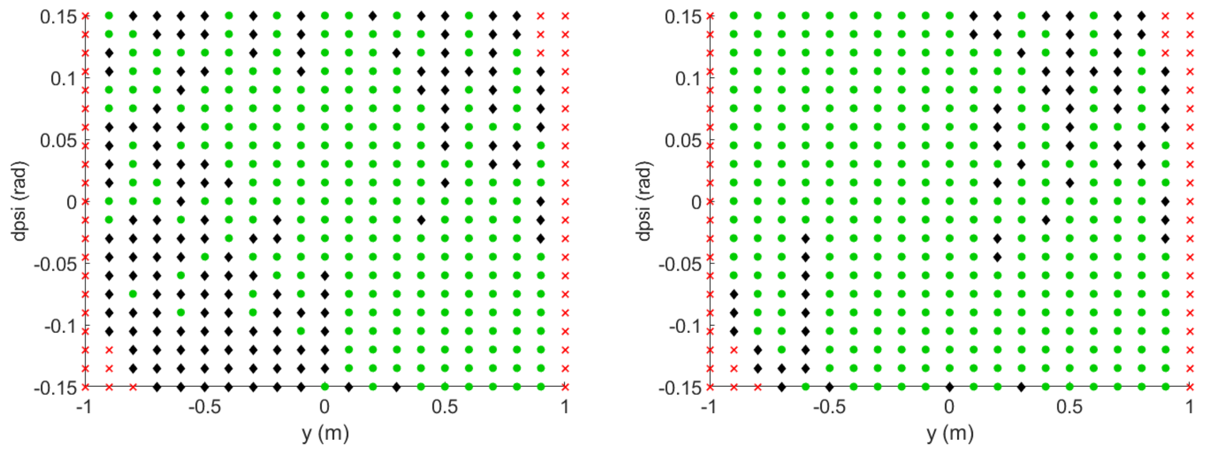


Fig. 10 S-TaLiRo: Distribution of initial condition grids for controller MPC$^{20}_{lat}$, with disturbance domains D (left) and $D_{1/2}$ (right).

2. Breach

In addition to S-TaLiRo, another falsification tool, Breach, proposed in [14], was also used to find falsifying initial conditions and disturbance trajectories to violate φ_{lat}. The framework for Breach is identical to that shown in Figure 6, except that Breach is used in place of S-TaLiRo.

For the Breach falsification runs, the same initial conditions were used: $y_0 \in [-y^{max}, y^{max}]$, $\Delta\psi_0 \in [-0.15, 0.15]$ and all other lateral dynamics state initial conditions are set to constants: $v_0 = 0$, $r_0 = 0$. In addition, the bounds on the external disturbance inputs are identical as in the analysis of S-TaLiRo, i.e., $d(t) \in D$ for all t where $D := [-y^{max}/2, y^{max}/2] \times [-v^{max}/2, v^{max}/2] \times [-\Delta\psi^{max}/2, \Delta\psi^{max}/2] \times [-r^{max}/2, r^{max}/2]$, and $d(t) \in D_{1/2}$ for all t where $D_{1/2} := [-y^{max}/4, y^{max}/4] \times [-v^{max}/4, v^{max}/4] \times [-\Delta\psi^{max}/4, \Delta\psi^{max}/4] \times [-r^{max}/4, r^{max}/4]$. We used the same set of meshed initial conditions of $y_0 \in [-1, 1]$, $\Delta\psi_0 \in [-0.15, 0.15]$ in a 21×21 grid without disturbances to determine whether these initial conditions alone lead to a violation of the specification φ_{lat}. Then, Breach was run on the remaining initial conditions to determine whether a falsifying trajectory can be found using a disturbance input. For each initial condition, Breach was also limited to 100 samples for each set of bounds on the disturbance input D and $D_{1/2}$.

Table 5 summarizes the falsifying initial condition rates using this method, while Figures 11–14 show the distribution of non-falsifying and falsifying grids. Red crosses indicate falsifying initial conditions without disturbance trajectories, black diamonds indicate initial conditions for which Breach finds a falsifying disturbance trajectory, and green dots indicate the remaining non-falsifying initial conditions.

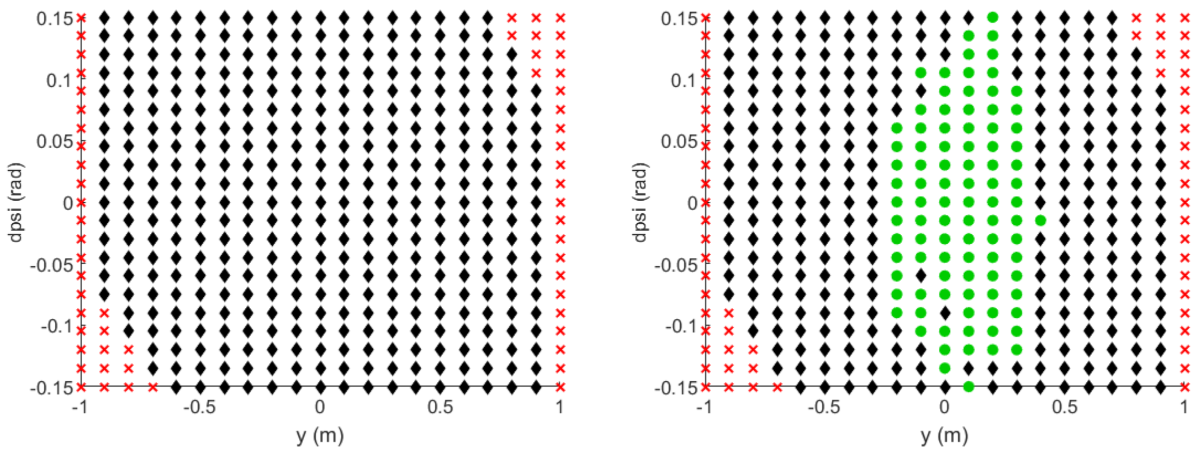


Fig. 11 Breach: Distribution of initial condition grids for controller P_{lat}, with disturbance domains D (left) and $D_{1/2}$ (right).

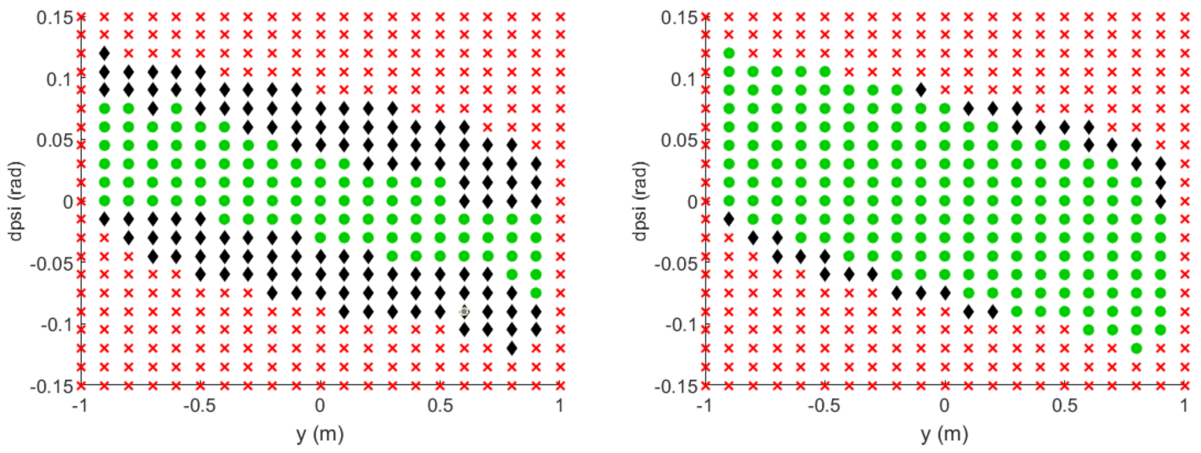


Fig. 12 Breach: Distribution of initial condition grids for controller MPC^2_{lat}, with disturbance domains D (left) and $D_{1/2}$ (right).

By comparing Tables 4 and 5, it is clear that Breach falsifies many more initial conditions than S-TaLiRo does for both disturbance bounds D and $D_{1/2}$. In fact, for disturbance bounds D, Breach falsifies every initial condition in the mesh for every controller except MPC^2_{lat}. This indicates that Breach was more successful than S-TaLiRo at finding "difficult" disturbance inputs for the controller. A possible explanation for this is the fact that Breach begins each set of falsification attempts by testing "corner-cases." These corner-cases represent setting the control points of the disturbance signal to their maximum (or minimum) values, thus resulting in a constant bias or a switching behavior between maximum positive and negative disturbance values. It appears that the controllers were very susceptible to falsification under this type of disturbance for disturbance bounds D. This is different from S-TaLiRo's random-search method. Only in the case of the MPC design controller MPC^{20}_{lat} do there appear to exist initial conditions that S-TaLiRo

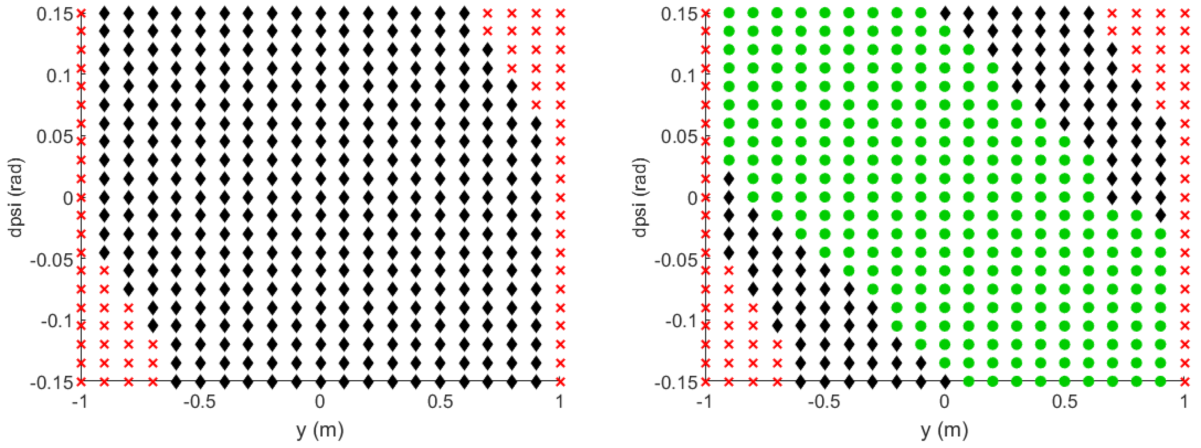


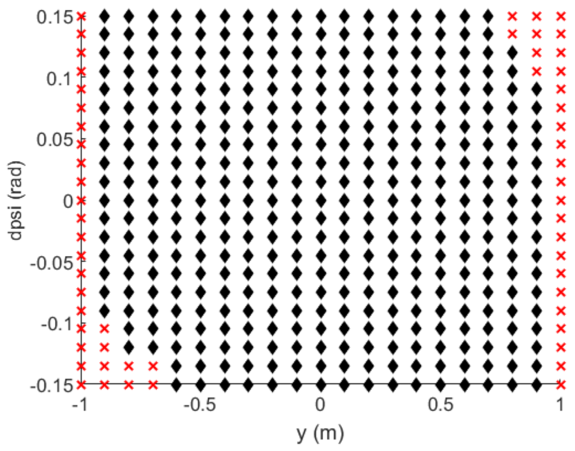
Fig. 13 Breach: Distribution of initial condition grids for controller MPC$^5_{lat}$, with disturbance domains D (left) and $D_{1/2}$ (right).

Table 5 Breach: Lateral dynamics falsification rate of initial condition grids. The entire set of initial conditions N has 441 grids. N_0 is the set of falsifying initial conditions without disturbance while N_n^1 and N_n^2 stand for the sets of initial conditions for which Breach can find falsifying disturbance trajectories within domain D and $D_{1/2}$. When calculating N_n^1 and N_n^2, we only use non-falsifying initial conditions without disturbance, which is $N \backslash N_0$.

Controller	**size of N_0 ($\frac{N_0}{N} \times 100\%$)**	**size of $N_n^1 + N_0$ ($\frac{N_n^1+N_0}{N} \times 100\%$)**	**size of $N_n^2 + N_0$ ($\frac{N_n^2+N_0}{N} \times 100\%$)**
P$_{lat}$	57 (12.9%)	441 (100.0%)	347 (78.7%)
MPC$^2_{lat}$	227 (51.5%)	349 (79.1%)	256 (58.0%)
MPC$^5_{lat}$	69 (15.6%)	441 (100.0%)	171 (38.8%)
MPC$^{20}_{lat}$	56 (12.7%)	441 (100.0%)	260 (59.0%)

was able to falsify but Breach was not. These can be seen by comparing the right-side images in Figures 10 and 14. Additionally, for disturbance bounds D, it appears that the MPC design controller MPC$^2_{lat}$ is the safest under Breach's falsification, as it is the only controller for which not all initial conditions were falsified. This contradicts the findings from S-TaLiRo mentioned previously, but could be explained due to Breach's more "extreme" method of choosing maximum or minimum disturbance inputs. This will result in larger model plant mismatch for Breach than S-TaLiRo, meaning that MPC controllers with longer prediction horizons will have greater errors between their predicted and actual future state. Thus, the safest controller found under Breach's falsification using disturbance bounds D may have a shorter prediction horizon than the safest controller found under S-TaLiRo's falsification using the same bounds D. However, for disturbance bounds $D_{1/2}$, MPC$^5_{lat}$ performs better than MPC$^2_{lat}$. This agrees with the findings from S-TaLiRo, which suggests that the MPC$^5_{lat}$ is the safest for disturbance bounds $D_{1/2}$ for both falsification tools. Another observation is that, for the smaller disturbance domain $D_{1/2}$, Breach also finds fewer falsifying disturbance trajectories than for domain D, just as S-TaLiRo did. Finally, just as with S-TaLiRo, the symmetry is somewhat broken in some of the figures, which may be due to Breach selecting different corner-cases for each point, or non-determinism in X-Plane. We also note that, in terms of the time it takes for falsification, we did not observe a significant difference between S-TaLiRo and Breach.

We note that there is a slight mismatch between the first columns of Tables 4 and 5. This is unexpected, since in this column should represent simply running the controller without any disturbance produced by the falsification tool. A



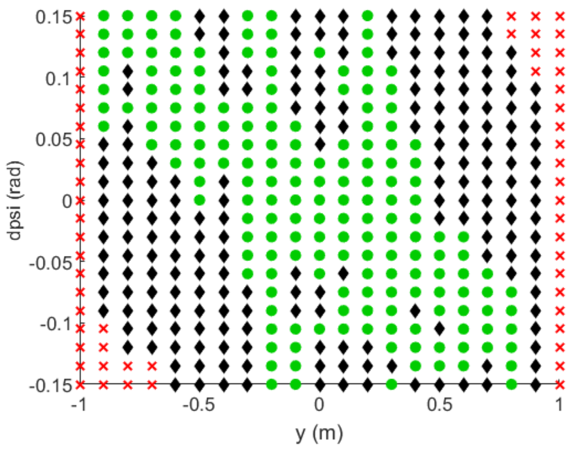


Fig. 14 Breach: Distribution of initial condition grids for controller MPC_{lat}^{20}, with disturbance domains D (left) and $D_{1/2}$ (right).

possible explanation for this that we noticed is again some non-determinism in X-Plane, i.e., running it from the exact same initial condition with the same controller does not always generate the exact same trajectory.

VI. Conclusions and future work

In this paper, we presented an architecture for autonomous taxiing operations. We divided the functionality into four parts: (i) a high-level discrete decision maker that ensures the validity of the routes provided by ATC and that notifies ATC whenever the route becomes infeasible/unsafe to proceed through; (ii) a middle-level path planner that combines the route information together with airport map information to generate waypoints that should be followed by the aircraft; (iii) a low-level controller that follows the generated path as long as feasible based on sensory information; and (iv) a sensor processing/perception layer (to be implemented as part of future work) that provides the low and high-level controllers with situational awareness necessary to safely accomplish the taxiing task. This architecture allows us to map certain specification to different parts in the architecture, which might be relevant for enabling the verification of the overall system in a modular manner. We then focused on items (i) and (iii). For (i), we developed algorithms that use graph-based abstractions of taxiway layouts of airports that are used for reasoning about route validity. For (iii), we designed controllers for longitudinal and lateral dynamics that together navigate the aircraft along the desired path. Finally, we used falsification tools S-TaLiRo and Breach to evaluate the set of initial conditions and disturbance inputs that different controllers for centerline tracking can tolerate. The larger the number of initial conditions a controller can avoid constraint violation from, the safer the controller. The disturbance inputs we consider act as a proxy for sensor imperfections. In the future, we plan to use more realistic sensor modalities, for instance, cameras with (potentially learning-based) perception components, for which new verification and falsification techniques might be required [15, 16]. One approach we are pursuing in this direction is to extend our earlier work on using game-based invariant synthesis [17] to the case with imperfect measurements. We are also interested in formalizing the complete taxiing specifications into requirements represented in signal temporal logic and developing a modular formal verification framework for ensuring that the proposed architecture provides safety guarantees for the system. The main challenge here is the scalability of the verification approaches to a system of this complexity. While sound but incomplete falsification techniques provide a means to get around the complexity issue and are shown to be efficient to find failure cases, it is not possible to get a correctness certificate based only on falsification. Therefore, developing a scalable and modular verification methodology would be desirable.

Acknowledgments

This work is supported in part by Collins Aerospace, a unit of United Technologies Corp. The authors would like to thank Timothy J. Woods from Collins Aerospace for helpful discussions. The article solely reflects the opinions and conclusions of its authors and not Collins Aerospace.

References

[1] Shepardson, D., "2017 safest year on record for commercial passenger air travel," *Reuters*, 2018.

[2] Young, S. D., and Jones, D. R., "Runway incursion prevention: A technology solution," 2001. NASA Technical Report.

[3] Schönefeld, J., and Möller, D., "Runway incursion prevention systems: A review of runway incursion avoidance and alerting system approaches," *Progress in Aerospace Sciences*, Vol. 51, 2012, pp. 31–49.

[4] Liu, C., and Ferrari, S., "Vision-guided Planning and Control for Autonomous Taxiing via Convolutional Neural Networks," *AIAA Scitech 2019 Forum*, 2019, p. 0928.

[5] Nuzzo, P., Xu, H., Ozay, N., Finn, J. B., Sangiovanni-Vincentelli, A. L., Murray, R. M., Donzé, A., and Seshia, S. A., "A contract-based methodology for aircraft electric power system design," *IEEE Access*, Vol. 2, 2013, pp. 1–25.

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

[7] Balachandran, S., Ozay, N., and Atkins, E. M., "Verification guided refinement of flight safety assessment and management system for takeoff," *Journal of Aerospace Information Systems*, 2016, pp. 357–369.

[8] Federal Aviation Administration, "AC 150/5300-13A - Airport Design," `https://www.faa.gov/airports/resources/advisory_circulars/index.cfm/go/document.current/documentNumber/150_5300-13`, 2012.

[9] Federal Aviation Administration, "Ann Arbor Municipal (KARB) Airport Diagram," `https://aeronav.faa.gov/d-tpp/1912/05506AD.PDF`, retrieved December 2019.

[10] Federal Aviation Administration, "Detroit Wayne County (KDTW) Airport Diagram," `https://aeronav.faa.gov/d-tpp/1912/00119AD.PDF`, retrieved December 2019.

[11] Teubert, C., "X-Plane Connect," `https://github.com/nasa/XPlaneConnect`, 2013–2017.

[12] Baier, C., and Katoen, J.-P., *Principles of model checking*, MIT press, 2008.

[13] Annpureddy, Y., Liu, C., Fainekos, G., and Sankaranarayanan, S., "S-taliro: A tool for temporal logic falsification for hybrid systems," *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2011, pp. 254–257.

[14] Donzé, A., "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems," *Computer Aided Verification*, edited by T. Touili, B. Cook, and P. Jackson, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 167–170.

[15] Dreossi, T., Donzé, A., and Seshia, S. A., "Compositional falsification of cyber-physical systems with machine learning components," *NASA Formal Methods Symposium*, Springer, 2017, pp. 357–372.

[16] Balakrishnan, A., Puranic, A. G., Qin, X., Dokhanchi, A., Deshmukh, J. V., Amor, H. B., and Fainekos, G., "Specifying and Evaluating Quality Metrics for Vision-based Perception Systems," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 1433–1438.

[17] Chou, G., Sahin, Y. E., Yang, L., Rutledge, K. J., Nilsson, P., and Ozay, N., "Using control synthesis to generate corner cases: A case study on autonomous driving," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 37, No. 11, 2018, pp. 2906–2917.