# Assumption Monitoring Using Runtime Verification for UAV Temporal Task Plan Executions

Sebastián Zudaire[1], Felipe Gorostiaga[2,3], César Sánchez[3], Gerardo Schneider[4] and Sebastián Uchitel[5]

*Abstract*— **Temporal task planning guarantees a robot will succeed in its task as long as certain explicit and implicit assumptions about the robot's operating environment, sensors, and capabilities hold. A robot executing a plan can *silently fail* to fulfill the task if the assumptions are violated at runtime. Monitoring assumption violations at runtime can flag silent failures and also provide mitigation and remediation opportunities. However, this requires means for describing assumptions combining temporal and quantitative data, automatic construction of correct monitors and ensuring a correct interplay between the planning execution and monitors. In this paper we propose combining temporal planning with stream runtime verification, which offers a high-level language to describe monitors together with guarantees on execution time and memory usage. We demonstrate our approach both in real and simulated flights for some typical mission scenarios.**

## I. Introduction

Temporal task planning (e.g., [1], [2]) and controller synthesis (e.g., [3], [4]) are increasingly being studied as a means to produce operational strategies for robots that are guaranteed to achieve a complex task [5]. A discrete model of the robot, its operating environment and its tasks are the input to an automated procedure that produces a strategy, often represented as an automaton, that can be directly executed by a robot using various hybrid control strategies [5], [6].

The guarantees provided by temporal task planning are subject to the satisfaction of a set of assumptions about the robot's dynamic operating environment, sensors, and capabilities. Some assumptions appear explicitly in the discrete model used for planning. These typically refer to the effects of controlled and monitored events that the planner can rely upon to produce a plan. An example of an *explicit planning assumption* is that requesting a photograph upon flying into a discrete region $i$ will be fulfilled before exiting that region.

However, other assumptions remain implicit and are crucial to ensure that a plan for the discrete representation of the planning problem will yield robotic behaviour in the real world that satisfies a continuous task specification (that is also commonly implicit). An example of an implicit assumption is that a photograph will completely cover a discrete region if the plane's roll and pitch are within certain bounds. We refer to these as *implicit assumptions* because they do not appear in the mathematical model used for planning. In this paper *we propose to monitor both explicit and implicit assumptions at runtime using runtime verification techniques.*

A robot executing a plan may fail if assumptions relied upon for temporal task planning are violated. Sometimes this failure can be observed by a user who may decide on mitigation and remediation actions. However, in some cases a mission can *fail silently* leading to users believing that a mission has been successful when it is not the case. The latter can have significant consequences. Thus the need for monitoring assumption violations

Consider a search and rescue task relying on the assumptions mentioned above. If a violation of the explicit planning assumption (i.e., the photograph is taken once the plane has left the region of interest) or the implicit assumption (e.g. the plane's roll is such that part of the region is not photographed) occurs, it might be erroneously concluded that the person searched for is not in the area.

Monitoring assumptions also provides opportunities for corrective actions. If the pitch and roll assumption violation is detected then, at the very least, searchers would be aware that the negative search result is inconclusive. Alternatively, flight-time mitigations such as re-visiting the specific area where the assumption was violated could be implemented. A monitor capable of predicting such a violation can report to a motion planner in an attempt to prevent the violation.

*In this paper we propose complementing task specifications with additional specifications of the assumptions that plans rely on, and an infrastructure to automatically synthesize monitors from specifications provided by engineers through an adequate monitor specification language to detect and predict assumption violations at runtime.* The monitor specification language should be high-level and expressive enough for typical planning assumptions, and monitor synthesis must come with guarantees of correctness and resource consumption bounds.

*Runtime verification* (RV) is a formal methods technique that studies how to synthesize monitors from high-level formal specifications of complex properties. RV borrows from static verification specification languages but focuses on the problem of observing a single trace, providing a formal framework for testing, debugging and monitoring. The formal approach in RV provides guarantees about the correctness and resources required to perform the monitoring task. We propose the use of *Stream Runtime Verification* (SRV) [7], [8], [9], a variant of RV that allows handling

Partially funded by the *STINT Initiation Grant on Runtime Verification for Autonomous Systems*, of the "The Swedish Foundation for International Cooperation in Research and Higher Education"; the ANPCYT PICT 2019-1442, 2018-3835, 2015-1718 and CONICET PIP 2014/16; by the Madrid Regional Government under project "S2018/TCS-4339 (BLOQUES-CM)" and by Spanish National Project "BOSCO (PGC2018-102210-B-100)".

[1] Instituto Balseiro - Univ. Nacional de Cuyo, Argentina.
[2] Univ. Politécnica de Madrid, Spain and CIFASIS, Argentina.
[3] IMDEA Software Institute, Spain.
[4] Univ. of Gothenburg, Sweden.
[5] Univ. de Buenos Aires, Argentina and Imperial College London, UK.

observations and verdicts richer than Boolean observations, including quantitative values and structured data.

*In this paper we combine temporal task planning with SRV to further improve the assurance of robotic behaviour. We illustrate how to enrich discrete task specifications with an explicit representation of the assumptions about the continuous world, which allows the use of SRV to monitor at runtime the validity of these assumptions.* We use SRV not only to monitor observations but also to anticipate and estimate behaviours, allowing us to detect and quantify violations, as well as to predict them. In particular, we integrate SRV and temporal task planning in a UAV hybrid controller architecture [6], and show its potential in three different mission scenarios: (1) supporting off-line post-mission decision making, (2) supporting on-line remediation actions at the hybrid-control and (3) at the discrete-event control layers.

In summary, the main contributions of this paper are (a) a novel combination of sophisticated stream runtime verification and temporal task planning to improve the effectiveness of robotic missions, (b) the practical illustration of these techniques in three different use cases, and (c) the empirical evaluation both in real and simulated UAV missions.

The rest of the paper is structured as follows. Section II presents the preliminaries of planning and runtime verification, and later Section III provides an overview of the interaction between these concepts. Section IV describes the different uses of the combination of SRV with planning, while Section V describes the practical empirical evaluation. Finally, Section VI discusses the related work and Section VII concludes.

## II. Preliminaries

**Labeled Transition Systems (LTS).** The dynamics of the interaction of the robot with its environment are modeled using LTS [10], which are automata where transitions are labeled with events that constitute the interactions of the modeled system with its environment. We partition events into controlled and uncontrolled to specify assumptions about the environment and safety requirements for a controller. Complex models can be constructed by LTS composition. We use a standard definition of *parallel composition* ($\|$) that models the asynchronous execution of LTS, interleaving non-shared actions and forcing synchronization of shared actions.

**Fluent Linear Temporal Logic (FLTL).** FLTL [11], a variant of linear-time temporal logic that uses fluents to describe states over sequences of actions, is used to describe environment assumptions and system goals.

A fluent $\mathit{fl} = \langle Set_\top, Set_\bot, v\rangle$ is defined by a set of initiating actions ($Set_\top$), a set of terminating actions ($Set_\bot$), and an initial value $v$ true ($\top$) or false ($\bot$). We may omit set notation for singletons and use an action label $\ell$ for the fluent defined as $\mathit{fl} = \langle \ell, \mathit{Act} \setminus \{\ell\}, \bot\rangle$. Thus, the fluent $\ell$ is only true just after the occurrence of the action $\ell$. FLTL is defined similarly to propositional LTL but where a fluent holds at a position $i$ in a trace $\pi$ based on the events occurring in $\pi$ up to $i$. Temporal connectives are interpreted as usual: $\Psi\varphi$, $\Phi\varphi$, and $\varphi\mathcal{W}\psi$ mean that $\varphi$ eventually holds, always holds, and (weakly) holds until $\psi$, respectively.

**Discrete Event Controller Synthesis.** Given an LTS $E$ with a set of controllable actions $L$ and a task specification $G$ expressed in FLTL, the goal of controller synthesis is to find an LTS $C$ such that $E\|C$: (1) is deadlock free, (2) $C$ does not block any non-controlled actions, and (3) every trace of $E\|C$ satisfies $G$. We say that a control problem $\langle E, G, L\rangle$ is *realizable* if such an LTS $C$ exists. The tractability of the controller synthesis depends on the size of the problem (i.e. states of $E$ and size of $G$) and also on the fragment of the logic used for $G$. When goals are restricted to GR(1) the control problem can be solved in polynomial time [12]. GR(1) formulas are of the form $\bigwedge_{i=1}^{n} \Phi\Psi\psi_i \Rightarrow \bigwedge_{i=1}^{m} \Phi\Psi\varphi_i$ where $\varphi_i$ and $\psi_i$ are Boolean combinations of fluents. In this paper we use MTSA [13], a tool for software architecture behaviour modelling, analysis [14], [15], [16] and discrete event controller synthesis [6], [17] for solving control problems.

**Implicit assumptions and Silent Mission Failures.** Planning over discrete domains requires introducing assumptions regarding the relation between the discrete abstraction and the intention in the real world that typically include continuous variables. Thus, a mission in the *real world* of the form "$A_{RW}$ implies $G_{RW}$" (where $A_{RW}$ represents the assumptions in the real world and $G_{RW}$ the mission goal). This problem is resolved using a *discrete* event controller goal $A_D \Rightarrow G_D$, which introduces two assumptions: "$A_{RW}$ implies $A_D$" and "$G_D$ implies $G_{RW}$". We refer to $A_{RW}$ and the two implications as *implicit assumptions*.

We refer to a *silent mission failure* when a system fails to achieve it goals $G_{RW}$ (because it vacuously satisfies the implication "$A_{RW}$ implies $G_{RW}$" ) but the system user cannot distinguish if it is that $G_{RW}$ holds or not.

**Hybrid Controller.** In the area of robotics, the difference in the continuous vs. discrete description of the real world, and in the interaction between *discrete event controllers* and *feedback-controllers* requires a non-trivial translation task that is implemented in a *hybrid control layer*. In [6] we present a hybrid control approach for robot missions that uses an iterator data structure to manage locations derived from the discretization of the workspace, specially suited for UAV missions involving hundreds or thousands of discrete locations, which we will use for the scenarios in this paper.

**Stream Runtime Verification (SRV).** Runtime verification (RV) studies the problem of whether a single observed trace satisfies a formal specification [18]. SRV [7], [9], [19] generalizes to richer datatypes using stream declarations (see examples in Sections III and IV). We have chosen the language HLOLA [20], [21] as the SRV language implementation to define our properties because (1) it has clean semantics, (2) its expressive yet succinct syntax results in natural specifications, (3) the language allows easy extensions (like new datatypes) and I/O of rich events, and (4) it allows the creation of predictive functions such as Kalman filters. Also, SRV allows a theoretical analysis of monitor specifications (including the ones in this paper), which lets us calculate

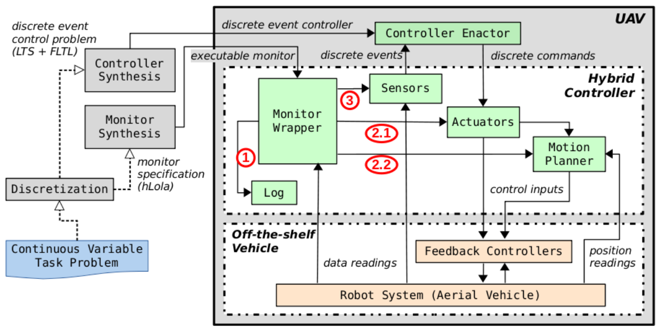


Fig. 1: Architecture for SRV + UAV Temporal Planning.

beforehand the constant amount of memory required to run the monitors throughout the entire execution independently of the length of the trace (see [8] for details).

The syntax of SRV is given by a tuple $\langle I, O, E\rangle$ where $I$ is a collection of (typed) input streams, corresponding to observations and $O$ is a collection of (typed) output streams, corresponding to output and intermediate values the monitor computes. Each output stream $o$ is associated with a defining expression $E_o$ that determines the computation of the value of $o$ at every position in time (depending on the values of inputs and outputs at the same or shifted time instants). For example, the following expresses that the altitude has always been high enough after take-off has finished:

```
output Bool ok=takeoff[now]||(ok[-1|T]&&L<alt[now])
```

where `ok` depends on the values of `take_off` and `alt` at the current instant (using `now`), and on `ok` itself at the previous instant (represented by the access offset `-1`). The value `T` in `ok[-1|T]` establishes that `ok[-1|T]` is true at the first instant.

## III. Overview of our solution

We provide a solution based on the architecture shown in Fig. 1 to monitor assumptions provided by user alongside the discrete event control problem. In particular, we use this approach in three different missions (Section IV) to (A) flag assumption violations after a mission concludes to prevent silent failures; (B) trigger recovery measures such as replanning or mission abort immediately upon an assumption violation; and (C) use quantitative streams in specifications to aid plan adjustment. We now overview our approach.

Consider the following continuous variable description of a search mission: $\exists t : r_t \in R \wedge r_t = p$, where $r_t$ represents the position of the robot at time $t$, $R \subset \mathbb{R}^2$ is a region, and $p$ is the location of the person being searched for. This mission as specified is unachievable from a practical point of view as the UAV does not have perfect precision in movement and must use a proxy (i.e., a sensor) to determine if the person is found within a range. This introduces an assumption in $A_{RW}$ stating that sensor event *yes.person* occurs if the person is close to the current UAV position (i.e., $\mathit{yes.person}_t$ if and only if $\|r_t - p\| \leq d_S$ where $d_S$ is the diameter of the area covered by the sensor). A mission goal ($G_{RW}$) amenable to discretization is $\left[\exists t : \mathit{yes.person}_t\right] \vee \left[\forall c_i \in D : \exists t : \left(|r_t - c_i| \leq d_T\right)\right]$, where $d_T$ is the location tracking error which is assumed to be less than the sensor breadth ($d_T < d_S$), and $D = \{c_1, \ldots, c_n\}$ a discretization of $R$ represented by the center points of circular regions of diameter $d$ where $d \leq d_S - d_T$ and the coverage assumption $\forall x \in R : \exists c_i \in D : \|c_i - x\| < d$ (all part of $A_{RW}$).

A discretized version of the mission goal is $G_d = \Psi(\mathsf{yes.person} \vee \bigwedge_i \mathit{visited(i)})$ where fluents *visited(i)* are initially false and change to true when at(i) occurs, at(i) is assumed to occur at time $t$ only if $\|c_i - r_t\| < d$ (also in $A_{RW}$). To achieve $G_d$, and given that yes.person and at(i) are monitored events, the following three assumptions must be explicitly included in the control problem: First, a command go(i) that guarantees eventually at(i) (i.e., $A_d^1 = \Phi \bigwedge_i \left[\mathsf{go(i)} \Rightarrow \Psi \mathsf{at(i)}\right]$; second that every command to sense for a person (person?) will be answered correctly with yes/no.person before leaving the discrete location at which the sensor was queried: $A_d^2 = \Phi \bigwedge_i \left[(\mathit{At(i)} \wedge \mathsf{person?}) \Rightarrow (\bigwedge_{i \neq j} \neg \mathsf{at(j)}) \mathcal{W} (\mathsf{yes.person} \vee \mathsf{no.person})\right]$, where *At(i)* is a fluent that is true with at(i) and false with go(i); and third that no unexpected at(i) events are received: (i.e., $A_d^3 = \Phi \bigwedge_i \left[\mathsf{go(i)} \Rightarrow \left((\bigwedge_{i \neq j} \neg \mathsf{at(j)}) \mathcal{W} \mathsf{at(i)}\right)\right]$.

Having introduced sufficient discrete event assumptions to make the discrete mission goal $\bigwedge_i A_d^i \Rightarrow G_d$ realizable, a Controller Synthesis component can be used to generate a discrete event controller. This controller is guaranteed to satisfy $G_d$ as long as the discrete event assumptions $A_d^i$ hold.

Satisfaction of $G_{RW}$ by Aerial Vehicle controlled by the discrete event controller via the Hybrid controller requires that assumptions $A_{RW}$ hold (e.g., if the breadth of the sensor when taking a picture is actually greater than $d_S$, the vehicle has the capability of reaching desired locations with certain precision). We advocate to explicitly model these assumptions and having mechanisms to automatically *monitor* them and *react* to violations. In other words, as illustrated in (see Fig. 1) the designer decomposes a continuous variable task problem into a discrete control problem and assumptions regarding discretization. The problem description is used to synthesize a discrete event controller. The assumptions (both those in the control problem and those pertaining discretization) are used to automatically generate monitors.

We specify assumptions using HLola, which are compiled into monitors that run within the Hybrid Controller, receiving data feeds from the Aerial Vehicle. Assumption $A_d^3$ can be monitored with the following specification:

```
input Position go_event, at_event
output Position going_to =
  if    go_event[0] != null then go_event[0]
  elsif at_event[0] != null then null
  else  going_to[-1|null]
output Bool unexpected_go =
  go_event[0] != null && going_to[-1|null] != null
output Bool unexpected_at =
  at_event[0] != null &&
  at_event[0] != going_to[-1|null]
```

This specification introduces an auxiliary output stream `going_to` whose value at every time point is the parameter with which go was last called, unless an at event has occurred afterwards, in which case the value of `going_to` is `null`. Recall that the value of the expression `stream[k|d]` is `d` if at time $n$ the offset $n+k$ is out of bounds.

The execution of the monitors is in parallel to the plan execution and completely unsynchronized with the enactment of the discrete event controller (c.f., CONTROLLER ENACTOR).

The MONITOR WRAPPER component executes the monitor implementations and can produce outputs to the LOG and to ACTUATORS so that the monitored data can be fed into the hybrid control, or to the SENSORS so that monitored data can be conveyed to the ENACTOR, generating events that impact the execution of the task plan.

## IV. ASSUMPTION VIOLATION HANDLING

In this section we show three scenarios in which temporal task planning and stream runtime verification are combined to monitor assumptions and react to their violations[1].

### A. *SRV for Offline Monitoring*

The first mission uses SRV for assumption monitoring and logging (see 1 in Fig. 1). The monitors process the incoming events and log assumption violations alongside the associated information, that can then be checked offline for mission post-hoc analysis. The UAV mission is a search and rescue mission similar to the one described previously in [6], [22] but with a no-fly zone and landing on finding the target.

We monitor two mission assumptions. The first checks that the UAV never enters the no-fly zone, and, if entered, measures the degree of violation as the distance inside the no-fly zone. The second ensures that the area sensed when entering a discrete location indeed covers it.

The first assumption can be easily monitored by calculating for each input position reading, the distance to the no-fly zone, returning null if the UAV is not in it.

There are several ways one could monitor the second assumption. Here we show a simplified monitor specification for a stronger property that requires the attitude `h` and position `pos` of the vehicle to be within certain bounds. Since these data measurements may carry noise, we include in the monitor specification a filtering phase with a first order low-pass filter. Note that most cameras may require several seconds to capture a high resolution image, specially on low-end hardware. For this reason, it is necessary to monitor that, for any instant between the sense command and the yes/no.person events, if a picture is taken, the filtered `h` and `pos` parameters must be within required bounds. Below we depict a simplified HLOLA specification for this assumption, setting the bounds in accordance to the mission parameters shown in Section V.

```
data Position = Position {x,y,alt :: Double}
data Attitude = Attitude {roll,pitch :: Double}
input Position pos, target
input Attitude h
input Bool capturing
output Bool near = dist(filter(pos[0]),target[0])<1
output Bool h_ok = filter(pos[0].alt)>8
output Bool roll_ok  = filter(h[0].roll)<0.0523
output Bool pitch_ok = filter(h[0].pitch)<0.0523
output Bool all_ok_capturing = capturing[0] ⇒
 (h_ok[0] && near[0] && roll_ok[0] && pitch_ok[0])
```

The input streams represent the data read from sensors or other monitors (not shown due to space restrictions). The input stream `capturing` indicates whether the person command has been issued but not yet responded. Auxiliar streams are generated from a `filter` function implemented in the specification (not shown). The output stream `all_ok_capturing` models whether the assumption holds, and it is computed based on auxiliary streams `near`, `h_ok`, `roll_ok` and `pitch_ok`. The logger will store values of all streams at the moment in which `all_ok_capturing` is `False` for a post-hoc analysis.

### B. *SRV for Online Monitoring and Adaptation*

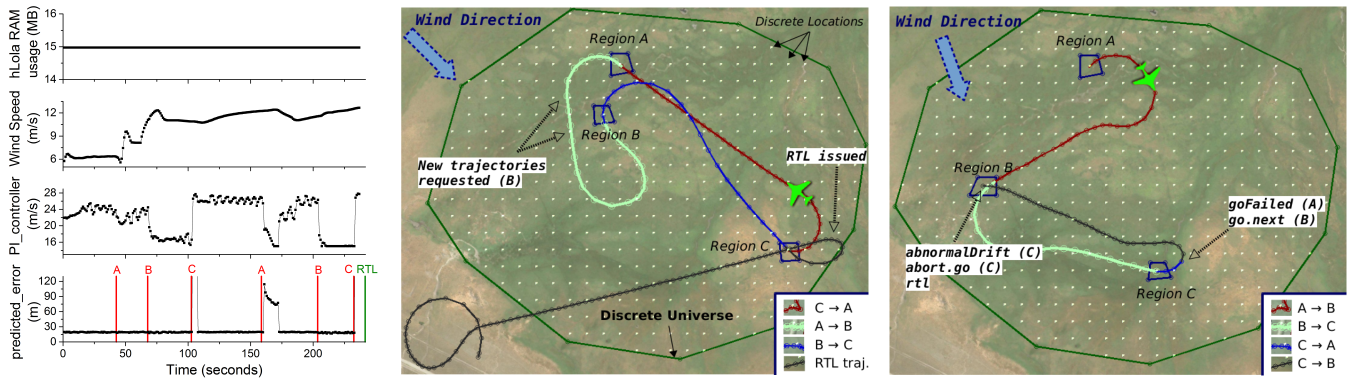
In this second mission we demonstrate how SRV can be used, beyond logging, to support remediation actions by interacting with components of the HYBRID CONTROLLER. We also showcase how more sophisticated monitoring capabilities of SRV describe the overall expected system behaviour. We ran an ordered patrol mission (as described in [23]) of three locations with no-fly zones. We tailored the monitors specifically for fixed-wing UAV.

We use two monitors that use quantitative data to *predict* the violation of assumptions, enabling the possibility of reacting to avoid the faulty behaviour *before* it occurs. One assumption monitor predicts how close the AERIAL VEHICLE will pass to the center of the discrete target location. Its output is fed into an actuator (2.1 in Fig. 1) that—based on the degree of the error predicted—can either directly abort the mission issuing a return to launch command (RTL) straight to the AERIAL VEHICLE, or request a new trajectory from the MOTION PLANNER (see 2.2).

The other monitor looks at how the AERIAL VEHICLE is following the Dubins path [24] computed by the MOTION PLANNER (see [6]) by predicting the estimated time of arrival and total flight distance. A significant difference is likely to be due to wind conditions (which are not considered by the MOTION PLANNER as the Dubins path computation assumes a constant turn radius). This last monitor also uses this arrival time and distance to compute the speed at which the vehicle is more likely to succeed in closely following the trajectory. The output of this monitor is consumed by an actuator (2.1) that changes the speed of the AERIAL VEHICLE.

Both monitors require prediction of the UAV's behaviour in following a trajectory. For this, the HLOLA monitor specifications encode a simplified non-linear 2D model of a fixed-wing UAV and the full extent of the waypoint guidance control algorithm of ArduPilot. Part of the input to these monitors includes the current state of the system (position, attitude, wind) and the list of waypoints for the current trajectory. The monitors use this input together with the non-linear model of the UAV and the guidance control

[1]The complete discrete event control problems and assumption used to synthesise plans and monitors, and the data obtained from the real and simulated flights is available at http://mtsa.dc.uba.ar.



(a) (b) (c)

Fig. 2: In (a) we show HLOLA streams and memory consumption during the ArduPlane simulation shown in (b), for the mission scenario in Section IV-B. In (c) we show an ArduPlane simulation for the mission scenario in Section IV-C. For clarity we only show in (b) and (c) the trajectory after the UAV visits for the first time region C and A, respectively.

algorithms to produce, by means of simulation into the future, a prediction of the UAV's flight path.

We add an *Extended Kalman Filter* (EKF) to estimate in-flight a parameter $tau$ from the simplified UAV models, helping to diminish the error in the predicted flight path. Other parameter identification techniques could be used instead (e.g., least squares), but we aimed to show HLOLA's ability to implement state-of-the-art estimation algorithms.

The PI controller `PIC` is used to control the UAV's airspeed set-point, taking as an input a reference velocity of $21\,\mathrm{m/s}$ and an estimation of the average UAV speed along with the predicted flight path, generated from the predicted arrival time (`estim_data.time`) and flight distance (`estim_data.dist`). The main idea is that in order to correctly follow a Dubins path one must satisfy its constant turn radius assumption, which (for the UAV) translates into constant ground-speed. When the UAV has tailwind, for a constant airspeed, it will fly at a faster ground-speed than when it experiences headwind. Controlling the ground-speed around a fixed value (by actuating on the airspeed set-point) helps to follow the desired path more accurately.

We show a portion of the specification of the second monitor focusing on the implementation of the PI controller. The function `simulate_guidance` simulates the UAV trajectory based on `tau` (provided by the EKF) and the input navigation. Note that some auxiliary functions are not shown (`is_satur` and `saturate`).

```
data Estimation = Estimation {time,dist :: Double}
input NavigationData nav_data
output Estimation estim_data =
  simulate_guidance(tau[0], nav_data[0])
output Double estimated_error =
  21 - estim_data[0].dist/estim_data[0].time
output Double err_int =
  if is_satur(PIC[-1|21],15,30) then err_int[-1|0]
  else err_int[-1|0] + estimated_error[0]*0.03
output Double PIC = saturate(21 +
           estimated_error[0]*0.5 + err_int[0],15,30)
```

## C. *SRV for Online Discrete Event Generation*

In the third mission we reuse the monitors from the second mission but allow their output to be consumed by the discrete event controller of the architecture (see (3) in Fig. 1). We instrumented the MONITOR WRAPPER component of the HYBRID CONTROLLER to produce three new events based on the `estimated_error` output stream. If the AERIAL VEHICLE is predicted to hit the target discrete location but far from its center point, then a nearMiss event is produced. On the other hand, if the target discrete location is to be completely missed, an event goFailed is produced.

Our monitor is specified to hold its failing prediction for several time instants before producing a goFailed, to allow the `PIC` described in the second mission to attempt to correct the airspeed. Additionally, the new monitors generate an auxiliary output stream `distance_to_trajectory` (not shown) that is used for the computation of `estimated_error` to produce an event abnormalDrift when the distance of the AERIAL VEHICLE to its intended location is beyond a threshold. This third scenario is a patrol mission, as before, but with the set of monitorable events extended with the new events and added to the discrete event control problem. The new requirements ensure that: (1) upon a nearMiss the UAV should reattempt to visit the location immediately, (2) upon a goFailed the UAV should skip the location and move onto the next location to patrol and (3) upon a abnormalDrift event, the UAV should abort mission and land.

## V. EMPIRICAL EVALUATION

In this section we report on the different combinations of SRV and temporal task planning described in Section IV. Our UAV hybrid control system (including SRV) was built for MAVLink [25] complying aerial vehicles, and tested extensively on the ArduPilot Software-In-The-Loop (SITL) simulator as in [26]. We have used this simulator (see [6]) to seamlessly transition from simulation to actually flying a custom made fixed-wing vehicle based on a Pixhawk. The HYBRID CONTROLLER was run on a single Raspberry

Pi. Controllers and monitors where synthesized offline in a few minutes and seconds, respectively. The attached video provides more data and insight on these flown missions.

For the *Offline Monitoring* mission in Section IV-A we performed real flights using the Parrot Ar.Drone 2.0 and an onboard mounted Raspberry Pi Zero W. Due to stringent battery limitation, we restricted the universe of locations to $116$ locations. The discretization size was of $4\,\mathrm{m} \times 4\,\mathrm{m}$ and a flight height of $8.5\,\mathrm{m}$. Even with this low-end hardware we were able to fly the mission with ease (RAM usage of HLOLA was always lower than $13.4\,\mathrm{MB}$), and detect post-flight assumption violations.

The missions described in Sections IV-B and IV-C were simulated on the ArduPlane (i.e., fixed-wing ArduPilot-based UAV) SITL simulator. Both share the same parameters of a cell-size of $50\,\mathrm{m} \times 50\,\mathrm{m}$ and a flight height of $100\,\mathrm{m}$, generating over 400 discrete locations. Dubins paths were computed selecting an arrival direction (from eight possibilities with $45^\circ$ increments) to each location to minimize the flight distance. The simulated UAV hardware was a Raspberry Pi 3B+ as in [6].

For the *Online Monitoring and Adaptation* scenario we show actions taken by the monitor during the patrol mission in Fig. 2b. As seen in Fig. 2a, we increment gradually the simulated wind speed over time. As a result, the second time the UAV travels from $A$ to $B$ (time range $160\,\mathrm{s}$ – $205\,\mathrm{s}$), the predicted arrival distance error rises over $100\,\mathrm{m}$, which triggers the calculation of a new trajectory until a trajectory that guarantees the correct arrival of the UAV to the location is found. Traveling from $B$ to $C$ and then to $A$ forces the UAV to switch between tailwind and headwind, and thus, as shown in Fig. 2a, the `PIC` stream from the monitor sets the flight speed accordingly. This is also the cause for the discontinuities (i.e., timeouts) in `estimated_error` after $C$ is visited. The first time $C$ is visited ($\sim$$100\,\mathrm{s}$), the monitor's speed controller compensates this effect quickly, but the second time ($\sim$$240\,\mathrm{s}$) the wind is too strong and a RTL command is issued. Fig. 2a illustrates HLOLA memory consumption, which remains practically constant as expected.

For the *Discrete Event Generation* scenario (Fig. 2c). we also increment the wind speed as the mission progresses. The result is a goFailed event generated when the UAV tries to go from $C$ to $A$ (with heavy headwind) and—as specified in the requirements—proceeds to visit $B$ instead. Due to the strong wind conditions at that point, once the UAV arrives at $B$ and calculates a trajectory to go to $C$, the predicted trajectory differs too much (abnormalDrift) causing the plan to abort (abort.go) and RTL (rtl). The memory consumed by HLOLA for the duration of the mission is similar to the one shown in Fig.2a. Memory consumption reports for other missions are omitted as they are always constant. See [20] for details about the memory consumption of HLOLA.

Our experiments show that HLOLA can monitor assumptions for UAV missions with negligible interference with the main navigation system, in particular in constant memory consumption and in constant time per event. The specifications and nontrivial mathematical functions are easily expressed in HLOLA[2], providing support for hypotheses that motivated the use of SRV and HLOLA (see Section III).

## VI. Related Work

The literature on temporal task planning is very extensive (e.g., [1], [2], [3], [4], [5]). We focus here only on the combination of planning with RV.

Ulus and Belta [27] introduce a multi-layered architecture where controllers interact with monitors to enforce properties obtained from LTL specifications, and propose to construct monitors from mission specifications. Their properties are limited to the discretized domain, but we use SRV, which allows much richer properties involving quantitative aspects and complex data-types, well beyond simple Boolean properties. We also monitor violations of the environment considering dynamic aspects (e.g., change of wind strength and direction) so the controller can take corrective actions. Finally, unlike [27], we consider predictive monitoring.

In [28], Liu and Baras use an ad-hoc approach to verify at runtime if a plan expressed in Metric Interval Temporal Logic is satisfied. Their motivation is similar to ours, avoid silent mission failures. However, their focus is on goal violation while we focus on checking assumption violations, which has the advantage of being able to be preventive.

Doherty et al. [29] present a temporal logic-based task planning and monitoring for UAVs based on Temporal Action Logic, integrated into a deployed rotor-based UAV. Their properties are Booleans and are not predictive. Thus, the monitors cannot play the same kind of role as described herein for mitigation of assumption violations.

In [30] Tiger and Heintz introduce Predictive MTL (P-MTL) that extends MTL to reason over stochastic states and predictions of states. While we use SRV for monitoring and predicting trajectories at runtime, they use P-MTL, but their verdicts are still Boolean, while we can compute and predict richer values. Besides, HLOLA is a very friendly and intuitive language for engineers for writing formulae in P-MTL (or any other extension of temporal logic). Additionally, HLOLA has performance and memory guarantees. This last comment also applies to [31], where Leng and Heintz use the formula progression procedure for MTL for execution monitoring. The limitations in expressivity preclude the above to implement Kalman filters in their monitor specification languages.

## VII. Conclusion and Future Work

We have presented an application of runtime verification infrastructure to support assumption monitoring and assumption violation remediation in UAVs running temporal task plans. Stream runtime verification allows describing assumptions in a high-level language that can be used to automatically synthesize monitors that can collect and process sophisticated data, generating rich verdicts while providing strong correctness and resource consumption guarantees.

We believe that the rich interplay of monitoring and planning can be integrated even further and plan to study

[2]Available at `software.imdea.org/hlola/specs.html`

how to efficiently perform dynamic discretizations and re-planning on-the-fly—depending on data provided by the monitoring/prediction—to further improve the behavior of robots running of temporal plans.

## References

[1] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[2] M. Guo, S. Andersson, and D. V. Dimarogonas, "Human-in-the-loop mixed-initiative control under temporal tasks," in *IEEE ICRA'18*, May 2018, pp. 6395–6400.

[3] S. C. Livingston and R. M. Murray, "Just-in-time synthesis for reactive motion planning with temporal logic," in *IEEE ICRA'13*, 2013, pp. 5048–5053.

[4] E. M. Wolff, U. Topcu, and R. M. Murray, "Efficient reactive controller synthesis for a fragment of linear temporal logic," in *IEEE ICRA'13*, 2013, pp. 5033–5040.

[5] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, pp. 1343–1359, 10 2008.

[6] S. A. Zudaire, M. Garrett, and S. Uchitel, "Iterator-based temporal logic task planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 11 472–11 478.

[7] B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, "LOLA: runtime monitoring of synchronous systems," in *TIME'05*. IEEE, 2005, pp. 166–174.

[8] C. Sánchez, "Online and offline stream runtime verification of synchronous systems," in *RV'18*, ser. LNCS, vol. 11237. Springer, 2018, pp. 138–163.

[9] F. Gorostiaga and C. Sánchez, "Striver: Stream runtime verification for real-time event-streams," in *Proc. of the 18th Int'l Conf. on Runtime Verification (RV'18)*, ser. LNCS, vol. 11237. Springer, 2018, pp. 282–298.

[10] R. Keller, "Formal verification of parallel programs." *Communications of the ACM*, vol. 19, pp. 371–384, 07 1976.

[11] D. Giannakopoulou and J. Magee, "Fluent model checking for event-based systems," in *ESEC/FSE'11*. ACM, 2003, pp. 257–266.

[12] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive (1) designs," in *Proc. of the 7th Intl' Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'06)*, ser. LNCS, vol. 3855. Springer, 2006, pp. 364–380.

[13] N. R. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, "Synthesis of live behaviour models," in *FSE'10*. ACM, 2010, pp. 77–86.

[14] S. Uchitel, R. Chatley, J. Kramer, and J. Magee, "Fluent-based animation: Exploiting the relation between goals and scenarios for requirements validation," in *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.* IEEE, 2004, pp. 208–217.

[15] J. Kramer, J. Magee, and S. Uchitel, "Software architecture modeling & analysis: A rigorous approach," in *SFM*, 2003, pp. 44–51.

[16] R. Chatley, S. Eisenbach, J. Kramer, J. Magee, and S. Uchitel, "Predictable Dynamic Plugin Systems," in *Proc. of the 7th Int'l Conf. on Fundamental Approaches to Software Engineering (FASE'04)*, ser. LNCS, vol. 2984. Springer, 2004, pp. 129–143.

[17] N. D'Ippolito, V. A. Braberman, N. Piterman, and S. Uchitel, "Synthesizing nonanomalous event-based controllers for liveness goals," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, pp. 9:1–9:36, 2013.

[18] K. Havelund and G. Roşu, "Synthesizing monitors for safety properties," in *TACAS'02*, ser. LNCS 2280. Springer, 2002, pp. 342–356.

[19] L. Pike, A. Goodloe, R. Morisset, and S. Niller, "Copilot: A hard real-time runtime monitor," in *RV'10*, ser. LNCS 6418. Springer, 2010.

[20] M. Ceresa, F. Gorostiaga, and C. Sánchez, "Declarative stream runtime verification (hlola)," in *Proc. of the 18th Asian Symp. on Programming Languages and Systems (APLAS'20)*, B. C. d. S. Oliveira, Ed. Springer, 2020, pp. 25–43.

[21] F. Gorostiaga and C. Sánchez, "Hlola: a very functional tool for extensible stream runtime verification," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2021.

[22] H. Choi, M. Geeves, B. Alsalam, and F. Gonzalez, "Open source computer-vision based guidance system for uavs on-board decision making," in *2016 IEEE Aerospace Conference*, 2016, pp. 1–5.

[23] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[24] X. Song and S. Hu, "2d path planning with dubins-path-based a* algorithm for a fixed-wing uav," in *ICCSSE'17*, Aug 2017, pp. 69–73.

[25] S. Atoev, K. Kwon, S. Lee, and K. Moon, "Data analysis of the mavlink communication protocol," in *ICISCT'17*, Nov 2017, pp. 1–3.

[26] J. d. S. Barros, T. Oliveira, V. Nigam, and A. V. Brito, "A framework for the analysis of uav strategies using co-simulation," in *SBESC'16*, Nov 2016, pp. 9–15.

[27] D. Ulus and C. Belta, "Reactive control meets runtime verification: A case study of navigation," in *RV'19*, ser. LNCS, vol. 11757. Springer, 2019, pp. 368–374.

[28] Z. Lin and J. S. Baras, "Planning and runtime monitoring of robotic manipulator using metric interval temporal logic," in *IEEE SysCon'19*. IEEE, 2019, pp. 1–8.

[29] P. Doherty, J. Kvarnström, and F. Heintz, "A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems," *Aut. Ag. and Multi-Ag. Sys.*, vol. 19, no. 3, pp. 332–377, 2009.

[30] M. Tiger and F. Heintz, "Stream reasoning using temporal logic and predictive probabilistic state models," in *TIME'16*. IEEE Computer Society, 2016, pp. 196–205.

[31] D. Leng and F. Heintz, "Approximate stream reasoning with metric temporal logic under uncertainty," *CAI'19*, vol. 33, pp. 2760–2767, 2019.