2018 IEEE Conference on Control Technology and Applications (CCTA)
Copenhagen, Denmark, August 21-24, 2018

Time Optimal Robust Fleet Management of micro UAV through Timed Games formulation

Henrik Schiøler[1] Luminita Totu[1] Jan Dimon[1] Kim Guldstrand Larsen [2] and Jakob Haahr Taankvist [2]

***Abstract*— We consider the case of fleet management of a moderate number of Unmanned Aerial Vehicles (UAV or UAS) in confined space, e.g. indoor. Indoor operation of UAVs requires indoor localization capabilities as well as a sufficiently precise 3D map of the facility, whereas fleet management of multiple UAV adds an extra dimension of complexity. Additional complexity is mainly introduced by constraints ensuring collision free operation. Based on earlier work on mapping, path planning and localization we suggest fleet management through a *Timed Game* formulation and *Winning Strategy* controller synthesis using the tool UPPAAL TIGA.**

I. INTRODUCTION

Route planning of mobile *robotic agents* such as *Automated Guide Vehicles* (AGV), *Intelligent Mobile Robots* (IMR) and lately UAV has been an active area of research for decades [1]. Considering a single unit, *least cost* trajectory planning in static or dynamic environments has been studied in []. For AGV and IMR almost all studies consider 2-dimensional work spaces, whereas for UAV an extra dimension of mobility and complexity is added. 3-dimensional mapping and path planning for indoor UAV operation is presented in [2], where a parallel box formulation is suggested for 3D mapping along with an *obstacle centered* way-point assignment procedure suggested in conjunction with *visibility graphs* for path planning. Fleet management of AGV has been presented in [1] employing *Genetic Algorithms* for time optimal scheduling of mobile part feeding robots for manufacturing. A similar approach is applied to UAV fleet management in [3] and [4] for the synthesis of high level fleet schedules. Although high level schedules include collision-free path plans for individual UAVs no concern is made to uncertainty and it is emphasized that high level schedules need run time refinement in terms of delays to accommodate uncertainty in the final detailed schedule. In this paper we suggest a timed game formulation [5] of the collision-free planning problem accommodating the uncertainty of flight duration and providing the synthesis of a time optimal *discrete event* controller producing real time flight commands for individual UAVs. Timed games [6] constitute a special branch of *Timed Automata* [7], where a subset of transitions are *un-controllable*. Timed automata have been suggested for scheduling in [8], [9] and for verification of flight control SW in [10], whereas controller synthesis with timed games has been suggested in [11] for a case of climatic control. In the above cases the tool UPPAAL TIGA [12], [13] for timed automata (games) modeling and verification has been applied.

[1]Section for Control and Automation, Aalborg Universty, Aalborg, Denmark (henrik,dimon,lct)@es.aau.dk

[2]Department for Computer Science, Aalborg Universty, Aalborg, Denmark kgl@cs.aau.dk, jht@at-systems.dk

II. BACKGROUND

The UAWORLD [14] project considers the application of UAVs in indoor environments for light weight logistics and attempts to establish a complete operational eco-system to facilitate the industrial adoption of UAV as a feasible, cost efficient and safe alternative to AGV and IMR. The eco-system comprises:

- Indoor 3D mapping
- Path planning
- Fleet management
- Indoor localization
- Robust wireless communication
- Robust Fault tolerant motion control of multi-rotor UAVs
- State of the art multi-rotor UAVs

III. MAPPING AND PATH PLANNING

Whereas earlier studies [2] within the UAWORLD project advocated for *Axes Parallel Boxes* (ABP) as the elementary building block for 3D mapping, industrial preferences have later called for a different paradigm; namely *Lifted Polygons*, where any obstacle O is defined in 3D space through a *non-self-intersecting* 2D polygon $P = \{(x_1, y_1), (x_2, y_2), .., (x_N, y_N)\}$ by

$$O = \{(x, y, z) \mid (x, y) \in \overline{int(P)},\ z \in [z_L, z_U]\} \quad (1)$$

defining in turn *top*

$$T = \{(x, y, z) \mid (x, y) \in \overline{int(P)},\ z = z_U\} \quad (2)$$

and *bottom*

$$B = \{(x, y, z) \mid (x, y) \in \overline{int(P)},\ z = z_L\} \quad (3)$$

as well as a number of vertical rectangular *sides* $\{S_1, .., S_{N-1}\}$ given by

$$\begin{aligned} P_i &= \{(x_i, y_i, z_U), (x_i, y_i, z_L), \\ &\quad (x_{i+1}, y_{i+1}, z_U), (x_{i+1}, y_{i+1}, z_L)\} \\ S_i &= \overline{int(P_i)} \end{aligned}$$

where $int(\cdot)$ and $\bar{\cdot}$ denote interior and closure of subsets of R^3 respectively.

A. Visibility

Euclidean shortest paths are alternating sequences of straight lines (in open space) and obstacle geodesics, which for polyhedral obstacles reduces to sequences of straight lines. Therefore we adopt a simplistic flight strategy: namely UAVs are flying in straight lines between way-points that are either origin and destination or edge/corner points on obstacle polyhedra. Shortest path routing in 3D space with polyhedral obstacles is known to introduce exponential complexity [15], which has called for a less complex approach, where way-points are assigned in pre-defined flight altitudes $\{L_1, .., L_M\}$, inspired by air-traffic, on obstacle corners. For an obstacle $P_i = \{(x_1^i, y_1^i), (x_2^i, y_2^i), .., (x_{N_i}^i, y_{N_i}^i)\}$ way points W_i are defined by the set

$$W_i = P_i \times \{L_1, .., L_M\} \subset R^3 \tag{4}$$

such that the entire set of way-points W can be defined by $W = \cup_i W_i$. For origin O and destination D a flight path is then defined as a sequence $P = \{O, w_1, .., w_L, D\}$, where $w_i \in W$. For path planning the *visibility graph* V_G is defined with vertices $V = W \cup \{O\} \cup \{D\}$ and edges $E \subseteq V \times V$ such that $e = (v_i, v_j) \in E$ if and only if the line segment connecting v_i and v_j does not cross any obstacle tops, bottoms or sides.

B. Path planning

Planning a path P for a single UAV comprises the task of selecting a sequence $\{e_1, .., e_{N_P}\}$ of edges in E such that $e_1 = (O, w)$ and $e_{N_P} = (w', D)$ and $w, w' \in W$. Continuous time reference trajectories X_R along edges $e = (w, w')$ within the time interval $[0, t_e]$ are given by

$$X_R(s(t)) = w + sw' \quad \text{for} \quad s(t) \in [0, 1] \tag{5}$$

where $s : [0, t_e] \rightarrow [0, 1]$ is sufficiently smooth to be mechanically viable and $\dot{s}(0) = \dot{s}(t_e) = 0$, i.e. any edge flight starts and ends with hover. More dynamical flight is of course possible but conflicts the visibility approach above and as such calls for trajectory/path planning of higher complexity.

The specific selection of paths may be accomplished through graph based shortest path approaches such as Dijkstra or $A*$, with either pure Euclidian cost or additional cost on climbing and number of way-points, since these are well known to increase energy/battery consumption. With the suggested approach flight time along an edge e is therefore nominally t_e but of course subject to uncertainties imposed by air flow disturbances, battery conditions etc. Uncertainties are conveniently captured through either probabilistic or non-deterministic modeling or combinations. Following a non-deterministic approach we may capture flight time uncertainty through estimates of *best* and *worst* case quantities, i.e.

$$t_e \in [\underline{t_e}, \overline{t_e}] \tag{6}$$

and further define a probability distribution P_e concentrated on $[\underline{t_e}, \overline{t_e}]$ to refine the uncertainty modeling allowing for the computation of statistics such as expected values and higher order moments.

C. Multiple path scheduling

We consider batch scheduling, i.e. a finite set of paths $B = \{P_1, .., P_K\}$ to be executed (flown) by equally many UAVs from time $t = 0$ onwards. The overall task is to infer a flight schedule of the shortest possible duration preventing UAVs to come in too close proximity during flight. We allow UAVs to rest hovering on way-points to wait for other UAVs to move further away. Finally UAVs resting inactive prior take-off or post landing in origin or destination respectively are disregarded, i.e. they present no risk to active UAVs.
Collecting all edges E_B contributing to B, i.e.

$$E_B = \{e \mid e \in P \in B\} \subseteq E \tag{7}$$

we may define the proximity graph G_P with vertices E_B and edges $E_P \subset E_B \times E_B$ defined by

$$E_P = \{e_p = (e_i, e_j) \mid d(e_i, e_j) \leq \delta\} \tag{8}$$

where $d(e_i, e_j)$ denotes the minimum Euclidean distance between line-segments e_i and e_j. The minimum distance δ is chosen from safety requirements.
With the proximity graph at hand we may ask for the collision free schedule that in shortest time executes the batch B. The resulting schedule would in general depend on edge durations $\{t_e\}$ which are pre-runtime only known up to the uncertainties expressed above. Therefore we instead ask for some causal strategy/controller issuing, at time t flight commands on the basis of flight history within $[0, t)$. The controller should be identified according to expected performance, which may be expressed in various manners; least probability of exceeding a certain upper time limit, guaranteed completion of the batch within a time limit, least expected completion time w.r.t. edge distributions $\{P_e\}$ etc. We generally adopt the term *Time Optimal Robust* for such controllers expressing robustness to uncertainties and time optimality. In this paper we restrict strategy synthesis to controllers with guaranteed completion time and subsequently assess results w.r.t. statistics such as expected completion time.

IV. Timed Automata and Games

Since timed games is an extension to timed automata we give a short introduction to timed automata leading to the definition of timed games. A timed automaton is state automaton with a state $x = (q, c)$ evolving over continuous time in a state space $X = Q \times C$, where $Q = Q_{loc} \times Q_{var}$ is a discrete space and $C = R_+^N$. The partition of Q into Q_{loc} and Q_{var} reflects a modeling paradigm where *locations*, i.e. elements in Q_{loc} define the basic model structure, whereas Q_{var} holds auxiliary discrete variables refining model behavior. The discrete state $q(t)$ is piecewise constant between transition instants $\{t_n\}$ where the continuous state (clocks) evolve according to

$$\frac{d}{dt} c_j = 1 \quad \forall j \in \{1, .., , N\} \tag{9}$$

At transition times clocks c_j may be reset, i.e. $\lim_{t \uparrow t_k} c_j(t) > 0$ and $c_j(t_k) = 0$.

Discrete state evolution happens through state transition limited to a pre-defined set of transitions $T \subset Q \times Q$ as well as sets of *guards*, one associated to each transition in T. Guards are logical propositions over $C \times Q_{var}$. Additionally state evolution is governed by *invariants* which are logical propositions over C. As an example we consider a timed automaton modeling the flight of a UAV between way points w_a and w_b estimated to last between 10 and 15 seconds, i.e $t_e \in [\underline{t_e}, \overline{t_e}] = [10, 15]$. We define Q_{loc} as

$$Q_{loc} = \{I, w_a, F, w_b\} \tag{10}$$

where I denotes an initial state, such that the transition from I to w_a indicates starting time. States w_a and w_b model the hovering state of respective way-points, whereas F models flight between way-points. Transitions T are then appropriately defined as

$$T = \{(I, w_a), (w_a, F), (F, w_b)\} \tag{11}$$

whereas $C = (c_g, c_l)$, where c_g acts as a *global* clock and c_l models flight time between way-points. A guard $c_l > \underline{t_e} = 10$ is associated to the transition (F, w_b) restricting transitions to take place no earlier than $\underline{t_e}$, whereas an invariant $c_l <= \overline{t_e} = 15$ is associated to state F forcing transitions to happen before $\overline{t_e}$. Resets $c_g := 0$ and $c_l := 0$ are associated to transitions (I, w_a) and (w_a, F) respectively to reset the the global clock initially and the flight duration clock upon departure from a way-point. A graphical impression of the defined timed automaton is shown using the graphical notion of UPPAAL depicted in Figure 1

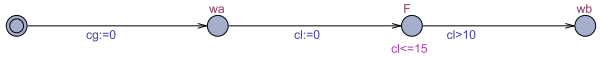

Fig. 1. Timed automaton modeling flight between way-points.

A. Parallel Composition

A notion of parallel composition between timed automata exists. Consider two timed automata $M_1 = (Q_1, C_1, T_1)$ and $M_2 = (Q_2, C_2, T_2)$, the parallel composition is (partly) defined by

$$M = (Q = Q_1 \times Q_2, C = C_1 \times C_2, T) \tag{12}$$

where $T \subseteq Q \times Q$, such that $(q_1^a, q_2^a), (q_1^b, q_2^b) \in T$ if

- $(q_1^a, q_1^b) \in T_1$ and $q_2^a = q_2^b$ or
- $(q_2^a, q_2^b) \in T_2$ and $q_2^a = q_2^b$ or
- $(q_1^a, q_1^b) \in T_1$ and $(q_2^a, q_2^b) \in T_2$ and (q_1^a, q_1^b) and (q_2^a, q_2^b) are synchronized

Synchronization in parallel compositions is not used in the following and not treated further in this exposition. Any guard associated to a transition $(q_1^a, q_1^b) \in T_1$ in M_1 is associated to $(q_1^a, q_2), (q_1^b, q_2)$ in M as is the case for resets (and vice versa for M_2). Invariants associated to states q_1 in M_1 is associated to states (q_1, q_2) in M.

With parallel compositions we may introduce the complete task-set model for an entire fleet of multiple UAVs. A timed automaton M_i is defined for each UAV by discrete state space

$$Q_{loc}^i = \{O^i, \{F_j^i, w_{j+1}^i \mid j = 1, ..N_i - 1\}, F_{N_i}^i, D^i\} \tag{13}$$

Transitions T^i are defined by

$$\begin{aligned} T^i = \{(O^i, F_1^i), \{(F_j^i, w_{j+1}^i) \quad &| \quad j = 1, ..N_i - 1\}, \\ \{(w_j^i, F_{j+1}^i) \quad &| \quad j = 2, ..N_i\}, (F_{N_i}^i, D^i)\} \end{aligned}$$

and a local clock c^i. Every flight state F_j^i is associated with an invariant $c^i <= \underline{d}_j^i$, whereas every transition $(F_j^i, \cdot)$ is associated with a guard $c^i > \overline{d}_j^i$. Bounds $\underline{d}_j^i$ and $\overline{d}_j^i$ model lower and upper bounds on flight time between way-points w_j^i and w_{j+1}^i. Additionally a time keeping automaton is defined with the sole purpose of resetting the global clock c. A graphical impression of the defined timed automaton is shown in the UPPAAL screen shot in figure 2. With multiple UAVs we augment the discrete state space, i.e. we set $Q_{var} = \{0, 1\}^{\#E_B}$ (implemented as the the Boolean array *Blocks* in UPPAAL TIGA) where $\#\cdot$ denotes cardinality. The initial state of Q_{var} is $0^{\#E_B}$ indicating that initially no edge is blocked.

For a single UAV, M, consider four consecutive transitions $t' = (w_j^i, F_{j+1}^i)$, $t'' = (F_{j+1}^i, w_{j+1}^i)$, $t = (w_{j+1}^i, F_{j+2}^i)$ and $t''' = (F_{j+2}^i, w_{j+2}^i)$ for consecutive edges $e' = (w_j^i, w_{j+1}^i)$ $e = (w_{j+1}^i, w_{j+2}^i)$ both in E_B. If $Q_{var}(e') == 1$ this can only be a block by M itself, since M has just been flying this edge. Also if $Q_{var}(e) == 1$, e must be blocked by M since e' and e are consecutive and therefore $(e', e) \in E_P$. As a result t is appropriately equipped with a guard $Q_{var}(e') \leq 1$ & $Q_{var}(e) \leq 1$ ensuring no flight on an edge blocked by other UAVs. The transition t is accompanied by a *blocking* transition $Q_{var}(\tilde{e})$+=1 for all $(\tilde{e}, e) \in E_P$ implemented in UPPAAL by the function

```
SetBlocks(e)
```

as well as an unblocking transition $Q_{var}(\tilde{e}) = \max\{0, Q_{var}(\tilde{e}) - 1\}$ for all $(\tilde{e}, e') \in E_P$ implemented in UPPAAL by the function

```
ResetBlocks(e')
```

A UPPAAL screen shot of 3 UAVs with blocking and unblocking transitions is shown in figure 2.

A run ρ of a timed automaton M constitutes the evolution of the state $x(t)$ over some time interval (finite/infinite) such that invariants on discrete states q are met for all times t where $x(t) == (q, c)$ and no transition between discrete states q and q' are done in contradiction to any guard on (q, q'). Let $\Gamma(M)$ denote set of all runs, then various properties of $\Gamma(M)$ are deducible. We shall limit the exposition to properties relevant to the UAV scheduling application, namely the existence of runs such that all UAVs reach their destination within a given time frame. In *Computation Tree Logic* (CTL) such a property my be checked with the following UPPAAL query for 2 UAVs (where "end" encodes destination and "$E <> \Phi$" queries the *existence* of

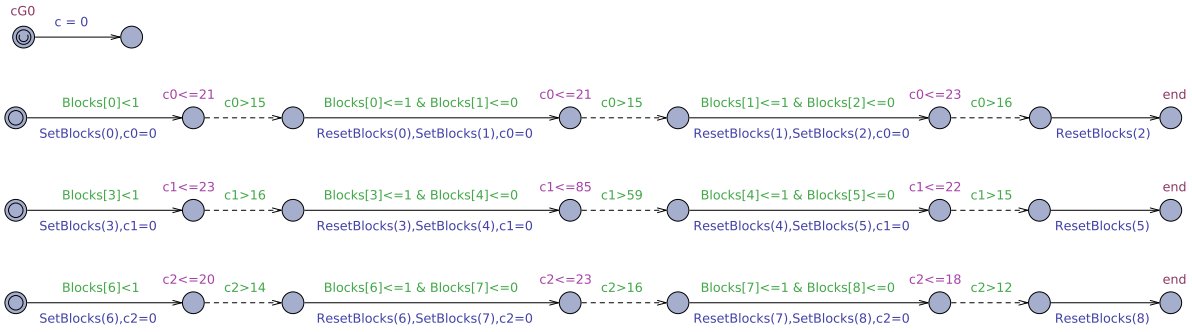

Fig. 2. Timed automaton modeling multiple flights with blocking/unblocking and controllable/uncontrollable transitions.

a path such that the subsequent proposition Φ is eventually fulfilled):

```
E<> uav0.end && uav1.end && c <= 152
```

B. Timed Games

A run ρ fulfilling the query above may then be used as a flight schedule if all transitions were under the control of some scheduling authority. This will generally **not** be the case for flight transitions $e = (F^i_{j+1}, w^i_{j+1})$, where external circumstances decide the exact time for the transitions to happen. This uncertainty is, as described above, non-deterministically modeled through the associated clock-invariant and -guard. This leads us to the definition of *Timed Games* which is readily defined as a time automaton (TGA), where some transitions are *uncontrollable*, i.e. T is partitioned into $\{T_C, T_U\}$. Uncontrollable transitions $\tau \in T_U$ model cases, where external circumstances decide the state evolution.

Closely linked to timed games is the concept of *strategy* $S : X \to T \cup \lambda$ to be interpreted as a state controller (also being representable as a timed automaton), which at every instant t emits a control signal $S(X(t))$ such that a timed game M and S in conjunction may be seen as a feedback control loop. Closed loop runs ρ_C fulfill for any controllable transition $\tau = (q_A, q_B)$

$$S(X(t_k)) = \tau \;\Rightarrow \lim_{t \uparrow t_k} q(t) = q_A \quad \text{and} \quad q(t_k) = q_B \qquad (14)$$

No transition happens outside of $\{t_k\}$. Let $\Gamma_C(M, S)$ denote the set of closed loop runs for a timed game M and a controller S, then various properties of $\Gamma_C(M, S)$ are deducible. In the UAV scheduling application we may ask whether for **all** runs $\rho_C \in \Gamma_C(M, S)$ all UAVs reach their destination within a given time frame. More ambitiously we ask whether a controller S exists such that this is achieved. Such a property may be checked with the following UPPAAL query for 2 UAVs:

```
control: A<> uav0.end && uav1.end
                      && c <= 152
```

Moreover the UPPAAL TIGA inference engine not only checks the existence of a controller S, it also constructs the controller as a piecewise constant function over X (we refer to each pair (constant, piece) as a *rule*, where a piece is a polyhedron over clocks). An example controller specification is partly given verbatim below

```
State: ( uav0._id2 uav1._id1 uav2._id0 )
While you are in
    (15<c && c<=23 && c==uav1.c1),
   wait.

State: ( uav0.end uav1.end uav2._id4 )
When you are in
     (136<c && c<=152),
          take transition
              uav2._id4->uav2._id5
```

where two rules are shown of which the former dictates a *wait*, i.e. no transition and the latter a discrete transition. Generally any strategy may be described using a finite collection of such rules.

C. Time optimality

We adopt a *robust time-optimal* design paradigm, i.e. we ask for a TGA batch model including N UAVs the following sequence of queries $\{S^p = K(M, C^p)\}$:

```
control: A<> uav0.end && uav1.end
          .. && uav<N>.end && c <= C^p
```

where $\{C^p\}$ is a sequence of time bounds found through a *bisection* procedure as follows

---

$A := C_L$
$B := C_U$
$p := 1$
while $A > B + 2$ **do**
  $C^p := \text{ceil}\left(\frac{A+B}{2}\right)$
  if a controller $S^k = K(M, C^p)$ exists **then**
    $A := C^p$
  else
    $B := C^p$
  $p := p + 1$
$S := K(M, A)$

Where C_L is found as the maximum over sums of upper time duration bounds for all UAVs and C_U as the sum over sums of upper time duration bounds for all UAVs. Thus, C_L becomes the smallest guaranteed duration if no blocking occurs and UAVs can be scheduled separately, whereas C_U becomes the smallest guaranteed duration if UAVs need to be scheduled consecutively, such that only on UAV is air borne at a time. Finally the obtained timed game controller is given as S. Importantly, robustness with respect to flight durations is guaranteed by construction.

V. Controller Architecture

The timed game controller S is obtained through a series of operations and need some transformation (interpretation) to act as a real-time scheduler for fleets of UAV. The solutions *life cycle* includes

1) Building 3D map
2) Selecting origins and destinations for batch of UAV flights
3) Path finding through pre-defined way-points.
4) Find robust time optimal controller through consecutive applications of UPPAAL TIGA inference engine
5) Translate verbatim UPPAAL TIGA controller S into executable format.
6) Real time execution of executable format.

We omit the exposition items 1)-3), whereas more details for items 5)-6) are given below.

The real time controller can be seen as part of a more extensive situational awareness command-and-control center responsible for indoor micro-UAV fleet management, including other software components as well as human operators. Scrutiny should be given in evaluating the accuracy of the behavioral model. Strategies for handling the un-modeled aspects of the system need to be developed in a satisfactory manner. At the same time, the interfaces and information exchange between the physically different systems (the UAVs and the control center) needs to be given careful consideration. For the implementation of the UPPAAL controller S into a real-time executor, the following behaviors are left un-modeled:

- Flight durations outside the $[\underline{t_e}, \overline{t_e}]$ bounds.
- Proximity alert, causing the UAV to immediately stop. If the sensor clears before another command is received, the UAV continues to the last received way point
- Low battery alert, causing the UAV to stop and wait for a new command

Un-modeled behavior is generally handled as an exception leading to *emergency landing* commands issued to all UAVs. Later developments aim to include *on-the-fly* re-scheduling for increased efficiency.

A. Controller translation

To facilitate an executable software representation of the controller S, a rule parser has been designed to convert the verbatim UPPAAL TIGA output into an appropriate object class structure as shown in Figure 3 directly manageable using object-oriented language such as C# or Java. Upon instantiation, `UAVDispatcher` takes a controller file output from UPPAAL TIGA as argument. When the file is parsed, the `UAVDispatcher` first tokenizes the file into substrings of the format shown in Section IV-B and then generates `Rule` objects for each substring.

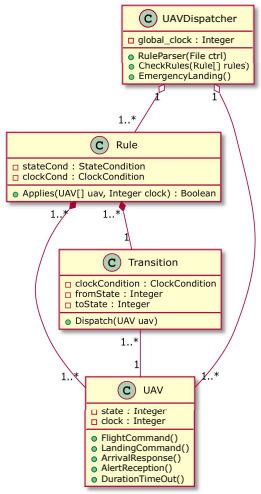

Fig. 3. Class diagram for rule parser; only the most important attributes and methods are shown

`Rule` objects store discrete state and clock conditions as well as associated conclusions in term of an allowed UAV state transition (`Transition` object), which in all cases encode a transition from *hovering at a way-point* to *flight to subsequent way-point*. Clock conditions are encoded as Boolean expressions based on the verbatim UPPAAL syntax for TGA clock zones captured in

$$\begin{aligned} g, g' \quad &::= \quad x \bowtie n \mid n \bowtie x \mid x - y \bowtie n \mid \\ &\qquad\quad n \bowtie x - y \mid g \wedge g' \mid (g) \vee (g') \\ \bowtie \quad &::= \quad \leq \mid \geq \mid == \mid < \mid > \end{aligned} \tag{15}$$

where x, y are clocks and n is a natural number. `Transition` objects associate each one `UAV` and holds origin and destination states as attributes. Each `UAV` object holds UAV state and local UAV clock.

B. Controller execution

Real time controller strategy execution is managed through making the `UAVDispatcher` object active (running as thread). Each `UAV` object is additionally made active to manage communication to its associated physical UAV through wireless network communication. `UAV` objects proxy *dispatch* and *emergency landing* commands from the `UAVDispatcher` to physical UAVs as well as *way-point reached* responses and alerts in the opposite direction. Simultaneously `UAV` objects track UAV state and clock evolution. The `UAVDispatcher` continuously examines every `Rule` with current UAV state and clock information and issues (through specific `Rule` and `Transition` objects) valid dispatch commands at the earliest valid opportunity. Figure

4 shows a component diagram for the proposed executor design.

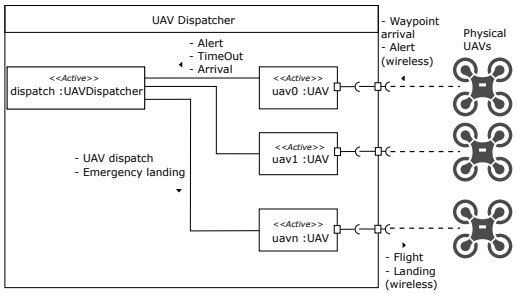

Fig. 4. Component diagram for the designed executer design.

VI. Tests and Results

A performance evaluation of the proposed methodology is performed through stochastic simulation for a moderate amount of test samples, where each sample comprises a batch of flight routes generated for the 3D model of the test flight space of which an excerpt is shown in figure 5 along with a sample path.

Fig. 5. Excerpt of flight space model with sample path.

For every sample a time optimal robust control strategy S is generated with a more tangible result being the smallest guaranteed completion time $T_G(S,s)$, which is used for comparison. Comparison values for each sample will be the best and worst case completion times $T_W(s)$ and $T_B(s)$ defined as follows. Let $\underline{t_e}_{i,j,s}$ be the lower bound flight duration of edge i for the jth UAV in sample s. Then

$$T_B(s) = \max_j \sum_i \underline{t_e}_{i,j,s} \tag{16}$$

corresponding to the case where all UAVs can execute paths independently with lowest flight durations. On the other hand let $\overline{t_e}_{i,j,s}$ be the corresponding upper bound flight duration, then

$$T_W(s) = \sum_j \sum_i \overline{t_e}_{i,j,s} \tag{17}$$

corresponding to a case with largest flight durations where blocking forces execution to be entirely sequential. Obviously $T_G(S,s) \in [T_B(s), T_W(s)]$ so a suitable comparison parameter $\gamma(s)$ measuring the gain of planning could be defined as

$$\gamma(s) = \frac{T_W(s) - T_G(S,s)}{T_W(s) - T_B(s)} \in [0,1] \tag{18}$$

A. Statistical Performance Evaluation

Since neither $T_G(S,s)$, $T_B(s)$ nor $T_W(s)$ will frequently be observed in practice, a statistical comparison seems in place. This however requires a probabilistic uncertainty model in terms of a distribution P_e of flight duration for each edge e as well as a dependency model for different flight durations. If uncertainty in flight duration is mainly caused by air flow fluctuations, statistical independence seem appropriate. If on the other hand battery conditions are the main cause, flight durations for a single UAV are more likely highly correlated. We adopt here an independence model and postpone more elaborate models to future work. With a probabilistic duration model available we may define $T_E(S,s)$ to be the expected completion time for sample s under strategy S control. Obviously $T_E(S,s) \in [T_B(s), T_G(S,s)]$ and as above we define a comparison parameter γ_E measuring the expected gain of planning by

$$\gamma_E(s) = \frac{T_W(s) - T_E(S,s)}{T_W(s) - T_B(s)} \in [0,1] \tag{19}$$

$T_E(S,s)$ may be estimated through stochastic simulation and is available from the Uppaal*Stratego* branch estimating non-parametrical statistics (histograms) for e.g. completion times as shown in figure 6

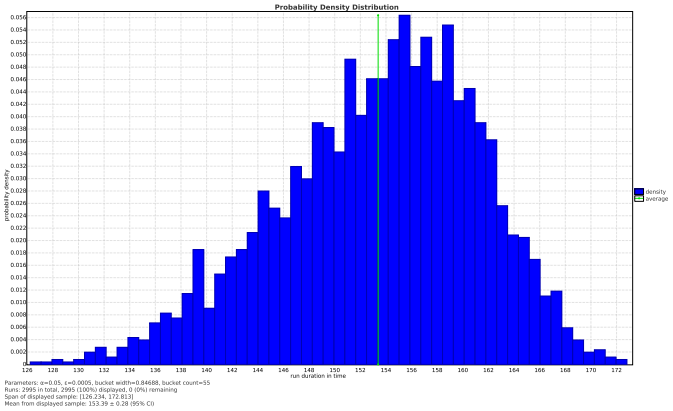

Fig. 6. Estimated completion time distribution from stochastic simulation.

where $T_G(S,s) = 172$ and $T_E(S,s) = 153$ are identified. Results in figure 6 are obtained assuming P_e to be uniform, which seems appropriate when uncertainty is relatively small, i.e.

$$(\overline{t_e} - \underline{t_e}) << (\overline{t_e} + \underline{t_e}) \tag{20}$$

or from a *maximum entropy* approach in absence of further knowledge. Higher uncertainty may call for more precise

flight duration modeling likely to produce uni-modal distributions such as *truncated* normal- or beta-distributions. For and edge $e = (w, w')$ we set a nominal flight $t_e^{nom} = \frac{|w-w'|}{v_{nom}}$, where v_{nom} denotes nominal UAV speed. Duration bounds are then set to meet $\overline{t_e} + \underline{t_e} = 2t_e^{nom}$ and $(\overline{t_e} - \underline{t_e}) = 0.2(\overline{t_e} + \underline{t_e})$, which is found to be sufficiently tight to justify uniform distribution. Results for γ and γ_E are found in figure 7 for10 randomly generated sample batches for the 3D test flight space model shown in figure 5.

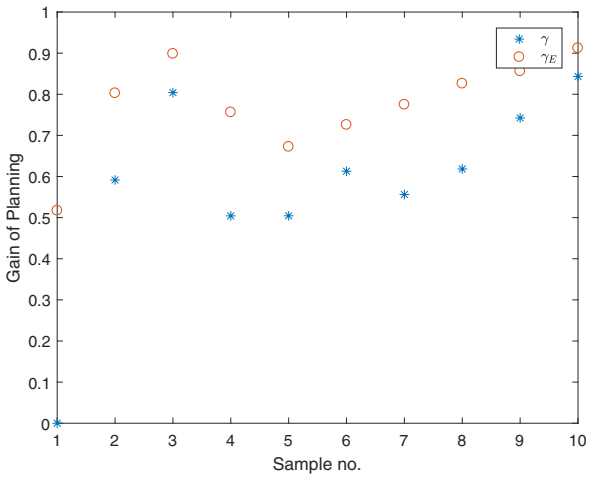

Fig. 7. Estimated gains of planning.

Results in figure 7 consistently show planning gains around 50%, whereas expected planning gains tend to be higher $70 - 80\%$. Despite a low number of samples results indicate a significant potential gain to be obtained through careful machine assisted planning. Results may of course vary among sample batches according to the geometric blocking of paths, i.e. if all paths are in close proximity, paths need to be flown sequentially and only a small gain is to be expected, whereas higher geometric spread allows for parallel path execution and higher planning gains.

VII. Conclusion

Fleet management of Unmanned Aerial Vehicles (UAV or UAS) in confined space, e.g. indoor has been considered as a planning task. A flight topology has been defined allowing for a Timed Game formulation of multiple simultaneous flight requests. A robust time-optimal *Winning Strategy* controller is then synthesized using the tool Uppaal Tiga. An architecture for implementing the synthesized controller as a real time executive has been formulated. Numerical results from stochastic simulation indicates that significant completion time gains are to be expected both in terms of worst-case quantities and statistical expectation.

We find that the proposed methodology is suitable as a paradigm for the realization of real-time fleet management, not only for the presented UAV case but for various mobile robot applications. Future research is focussed on the elaborate inclusion of un-modelled behavior and *on-the-fly* re-scheduling.

References

[1] Q.-V. Dang, I. E. Nielsen, and G. Bocewicz, *A Genetic Algorithm-Based Heuristic for Part-Feeding Mobile Robot Scheduling Problem*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 85–92.

[2] H. Schioler, L. Totu, A. la Cour-Harbo, J. J. Leth, and J. Larsen, "Easy 3d mapping for indoor navigation of micro uavs," in *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics - Volume 2: ICINCO,*, INSTICC. SciTePress, 2017, pp. 300–308.

[3] Y. Khosiawan, I. Nielsen, N. Do, and B. Yahya, *Concept of Indoor 3D-Route UAV Scheduling System*. Germany: Springer, 2, vol. 429, pp. 29–40.

[4] Y. Khosiawan and I. Nielsen, "A system of uav application in indoor environment," *Production & Manufacturing Research*, vol. 4, no. 1, pp. 2–22, 2016.

[5] A. David, K. Fleury, K. Larsen, and D. Lime, "Efficient on-the-fly algorithms for the analysis of timed games," *Lecture Notes in Computer Science*, vol. 3653, 2005.

[6] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *STACS 95*, E. W. Mayr and C. Puech, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 229–242.

[7] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994. [Online]. Available: https://doi.org/10.1016/0304-3975(94)90010-8

[8] Y. Abdeddaim, E. Asarin, and O. Maler, "Scheduling with timed automata," *Theor. Comput. Sci.*, vol. 354, no. 2, pp. 272–300, Mar. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2005.11.018

[9] M. Bisgaard, D. Gerhardt, H. Hermanns, J. Krcal, N. Gilles, and M. Stenger, "Battery-aware scheduling in low orbit: The gomx–3 case," in *FM 2016: Formal Methods*, F. John, H. Constance, G. Stefania, and P. Anna, Eds. Cham: Springer International Publishing, 2016, pp. 559–576.

[10] Y. Zhang, Y. Dong, Y. Zhang, and W. Zhou, "A study of the aadl mode based on timed automata," in *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, July 2011, pp. 224–227.

[11] J. Jessen, J. Rasmussen, K. Larsen, and A. David, "Guided controller synthesis for climate controller using uppaal tiga," *Lecture Notes in Computer Science*, vol. 4763, 2007.

[12] "Uppaal homepage," http://www.uppaal.org/, accessed: 2018-01-26.

[13] "Uppaal tiga homepage," http://people.cs.aau.dk/ adavid/tiga/, accessed: 2018-01-26.

[14] "Uaworld project description," http://uaworld.dk/, accessed: 2018-01-25.

[15] C. H. Papadimitriou, "An algorithm for shortest-path motion in three dimensions," *Information Processing Letters*, vol. 20, no. 5, pp. 259–263, 1985.