Receding Horizon Synthesis and Dynamic Allocation of UAVs to Fight Fires

Joshua Shaffer, Estefany Carrillo, and Huan Xu, Member, IEEE

***Abstract*—This paper explores the design of a high-level mission planner and controller for managing UAVs fighting a wildfire through the utilization of reactive synthesis and dynamic allocation of the UAVs as resources to the fires. Reactive synthesis provides a formal means of guaranteeing the UAVs transition to areas of fire, refill on water, and emergency land as defined by linear temporal logic specifications. Dynamic allocation coordinates the behavior of multiple UAVs through assignments to regions of fire based on a cost function that takes into effect the fire locations, fire intensities and other UAV locations. For six fire scenarios, this paper determines the minimum number of UAVs required to eliminate all fires and prevent the burning of more than a set amount of fuel (i.e. forestry or shrubbery). Lastly, our results and successful application expand discussion on the utilization of reactive synthesis in larger task spaces and the implications of abstracting UAV transitions for use in formal methods.**

I. Introduction

The economic burden of wildfires on the United States (in 2016 $US) exceeds $63.5 billion annually due to damages; more than $7.6 billion is spent annually to fight said fires [1]. Current methods of fighting fires involve numerous man hours and piloted vehicles capable of dropping large amounts of suppressant over regions of fire. As discussed in [2], unmanned aerial vehicles (UAVs) pose a huge benefit to traditional firefighting methods by providing additional "eyes in the sky" and supply drops, which has motivated numerous agencies to explore such options in the last decade. Furthermore, UAVs could potentially extinguish fires on their own as was mechanically demonstrated on a small, single-UAV scale in [4]. Fighting wildfires primarily through a fleet of autonomous drones, in turn, could limit the number of required personnel and produce more efficient results. For the purposes of research in UAV autonomy, a design problem such as this allows for the exploration and further assessment of multiple methods in autonomy.

One such method, reactive synthesis, provides a means for generating correct-by-construction controllers for complex systems. Synthesized controllers take into account system and environment variables with varying dynamics, progress and safety specifications, and initial conditions. The primary benefit of this method is the correct-by-construction attribute; the realized controller will satisfy the specifications designed by the user as long as the environment behaves according to the assumptions made. Programmers are not required to "handcraft" individual behaviors of a system under specific conditions (of which many are often error prone) and can instead focus on defining the system and specifications. The major pitfall of reactive synthesis, however, is the computational complexity in dealing with dynamic environments as discussed in [3]. For example, consider a scenario involving the path planning of a robot that must always avoid any number of arbitrarily placed obstacles on a 10x10 grid. The synthesized controller must account for every possible obstacle location (2^{100} permutations), resulting in an inordinate amount of computation time. Full models of real problems will easily involve this scenario's scale of permutations in the environment. As a result, reactive synthesis is typically restricted to "smaller" problems that utilize low-granular finite abstractions of system state spaces alongside limited dynamic environments.

When the high-level design of a system does involve the previous example's scale of environmental permutations and state space, solutions typically seek to discretize the synthesis problem or approach the problem from a different perspective. Discretization appears in the use of receding horizon control in [5] and the use of decentralized controllers for multiple agents in [6]. Both examples break down the top-level synthesis problem into smaller, discrete pieces for the computational benefits. On the other hand, [7] approached their synthesis problem with a focus on resolving deadlock under specific environment conditions instead of directly avoiding dynamic obstacles. In each of the presented cases, the problem description focuses on a limited task space and how the solution can handle larger sets of actions from an environment. In [5], [6], and [7], the task space encompasses up to 4 progress statements for each scenario.

For the purposes of fighting large, dynamic fires with a single autonomous UAV, a high-level controller created with just reactive synthesis would quickly present an impractical solution if the design should accommodate a fairly granular state space with a large number of tasks. Further expanding this concept to a whole fleet of UAVs, the problem becomes worse due to an increase in the number of system variables. Even with decentralized controllers for each UAV, more system variables would need to be introduced to describe coordination and behaviors between the decentralized UAV controllers. To alleviate the computational complexity on reactive synthesis in this regard, the coordination of UAVs can be handled by a dynamic allocation process.

Dynamic allocation presents an automated method for assigning resources in an ever-changing environment. Recent approaches include clustering techniques and formulating the allocation problem in the form of a multi-objective problem. In [8], a partitioning-based task allocation strategy is proposed to decide on the assignment of agents in dynamic disaster environments by applying the K-means clustering algorithm. In [9] and [10], the resource allocation problem is framed as a multi-objective optimization problem of minimizing the

J. Shaffer, E. Carrillo, and H. Xu are with the Department of Aerospace Engineering, University of Maryland, College Park, MD 20740, USA. e-mail: {jshaffe9, ecarril2} @terpmail.umd.edu, mumu@umd.edu.

This work was partially funded by Lockheed Martin.

extinguishing time and resource utilization cost, solved by the use of evolutionary algorithms in [9] and by mathematical programming techniques in [10].

We propose an allocation strategy to address the assignments of UAVs for our fire fighting scenario. If the fleet of UAVs are treated as resources to manage with respect to a changing fire landscape, dynamic allocation would serve well in assigning the UAVs to specific fires. Assignments would depend upon factors in the behavior of the fires such as density, intensity, ability to spread, and more. Perhaps readily apparent, though, is that an allocation algorithm would not necessarily be constructed to control the UAVs within the state space defined, nor would the algorithm manage other high-level system aspects associated with the UAVs, such as suppressant control and decisions on emergency landings. Building these high-level behaviors into an allocation algorithm requires "handcrafting" the behaviors for all scenarios, an approach that is avoided through the use of synthesis.

To integrate the two discussed methods, reactive synthesis and dynamic allocation, we utilize the receding horizon framework for reactive synthesis as discussed in [3]. As far as we have found, [11] first touched upon the idea of manipulating the receding horizon framework for decomposing the synthesized problem and decentralizing the planning procedure. For their purposes, this idea resulted in the ability of multiple agents to satisfy high-level specifications through only considering other agents that entered their local horizon. For our purposes, decentralizing the planning procedure allows for dynamic allocation to arrange the order of progress goals specified in the synthesized controller in real-time. Through this, we seek to reconcile the strengths and weaknesses of the two discussed methods to create a high-level mission planner and controller for implementation in a fire fighting scenario.

The rest of the paper follows as such. In *Section II*, we present subjects pertinent to the exploration of our topic, primarily in relation to reactive synthesis. In *Section III*, we present the fire fighting scenario, including environment and system definitions. *Section IV* discusses our proposed solution, followed by the implementation of our solution in *Section V*. In *Section VI*, we present the outcomes to our tested cases, followed by our conclusions in *Section VII*.

II. Preliminaries

Linear temporal logic (LTL) is utilized for describing specifications within the reactive synthesis framework. LTL makes use of boolean system variables that serve as atomic propositions (AP), and LTL formulas are built through APs with logic connections and temporal modal operators. Logic connections include $\neg$ (negation), $\vee$ (or), $\wedge$ (and), and $\implies$ (implication). Temporal modal operators include $\bigcirc$ (next), $\Box$ (always), $\Diamond$ (eventually), and $\mathscr{U}$ (until). LTL formulas that only utilize logic connections are referred to with φ. Through the use of LTL, a broad range of specifications can be written to describe the behaviors of a system (or environment). We point the reader to the preliminaries section of [12] for an expanded description of LTL as it pertains to our purposes.

Reactive synthesis provides a method for generating controllers within the context of a defined environment and system

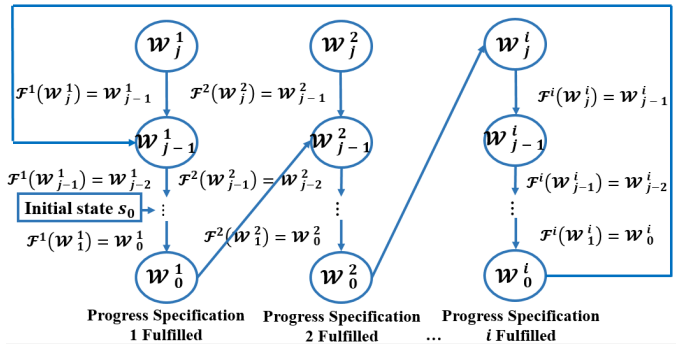

Fig. 1: Segmentation of state space and example of ordered set flow-down performed in receding horizon framework, as described in [3].

as specified through LTL. One of the most commonly used forms for these LTL formulas is general reactivity(1) (GR(1)), the assume-guarantee form shown in Eq. (1),

$$\begin{aligned}(\varphi_{init}^e \wedge \bigwedge_{i\in I_r} \Box\varphi_{s,i}^e \wedge \bigwedge_{i\in I_f} \Box\Diamond\varphi_{p,i}^e) \implies \\ (\varphi_{init}^s \wedge \bigwedge_{i\in I_s} \Box\varphi_{s,i}^s \wedge \bigwedge_{i\in I_g} \Box\Diamond\varphi_{p,i}^s).\end{aligned} \tag{1}$$

From the above equation, the propositional formula φ_{init} describes the initial condition of the environment or system (denoted by superscript e or s, respectively), $\varphi_{s,i}$ describes safety specifications, and $\varphi_{p,i}$ describes progress specifications. [12] proved that when the system and environment specifications are written in the framework shown in Eq. (1), the synthesis of a system controller that fulfills the specifications while subject to the environment will occur in polynomial time. The proof of such, though, provides no guarantee that the polynomial time required will fall within a practical time-limit, due to the formulation's limiting dependency on the total amount of permutations between the system and environmental states.

[13], [14], and [3] have explored reactive synthesis within a receding horizon (RH) framework. The primary benefit of utilizing RH is the segmentation of the state space for both the environment and system into separate horizons. Each horizon provides a smaller problem to synthesize a controller for, and the combination of these controllers form a single controller that obeys the specifications written for the total system and environment. The primary disadvantage of RH is that while each horizon itself can be optimized, the total sum is not. Other sources have explored methods of optimizing control with respect to time-based rewards on each horizon, such as in [14], but such optimization is forgone in this paper.

Implementation of RH revolves around, for each progress specification, segmenting the total system state space into regions $\mathscr{W}_j^i$ so that, when placed into properly constructed ordered sets represented by $\mathscr{F}^i(\mathscr{W}_j^i)$, the system variables will converge to meeting each progress statement for the system, i.e. $\mathscr{W}_0^i$. Here, i represents the system progress statement $i \in I_g$, and j indexes the ordered regions $\mathscr{W}$ about the progress specification i. This process is displayed in Fig. 1. Following the basis laid out by [3], each region consists of its own GR(1) specification (shown in Eq. (2)), constructed so that the synthesized controller will move the system states towards

the next region within the ordered set (eventually leading to $j = 0$) and fulfill the top level GR(1) specification,

$$\Psi^i_j = ((s \in \mathscr{W}^i_j) \wedge \Phi \wedge \bigwedge_{i \in I_r} \Box \varphi^e_{s,i} \wedge \bigwedge_{i \in I_f} \Box \Diamond \varphi^e_{p,i}) \longrightarrow (\bigwedge_{i \in I_s} \Box \varphi^s_{s,i} \wedge \Box \Diamond (s \in \mathscr{F}^i(\mathscr{W}^i_j)) \wedge \Box \Phi). \tag{2}$$

In Eq. (2), s refers to the system state. The formula Φ consists of all limitations on the states of system, preventing the system from making transitions to or initializing within states that are not allowed. This tautology prevents individual synthesized controllers from creating transitions to states that are infeasible for longer horizons.

III. Problem Formulation

The high-level problem scenario this paper explores is presented as such. A region of forest and/or grassland, segmented by various large-scale obstacles, is experiencing a wildfire. The fire spreads from starting points with a fixed behavior, and the starting fire conditions are arbitrary, i.e. any number of fires can occupy any number of subdivisions in the region that contain fuel to burn. A base of operations exists near the edge of the region and contains a fleet of N UAVs for fighting the fire. Each UAV holds a varying level of water for dumping on the fire, from High (100%), Medium (60%), Low (20%), to Empty (0%), associated with a total water volume of $W_v = 50$ liters. Each individual UAV contains a radio for communicating with base, GPS for determining position, and any other sensors required for lower-level controllers. Each UAV's average flight speed, v, is approximately 15 m/s.

Definition 1. *The system is defined as $S = s_p \times s_o \times w$, where $s_p = \{1,2,...x\} \times \{1,2,...y\}$, $s_o = \{1,2,3,4\}$, and $w = \{0,1,2,3,4\}$. x and y are integer values for the position in 2D space.*

Definition 2. *The transition relation for the state set S is defined as $T \subseteq S \times S$, where elements $s \rightarrow s' \in T$ are defined for each of the following s' definitions for each $s \in S$. Note that $s' \in S$.*

If $s_o \in s = 1$, $s' = \{s_{x,y} + \{0,1\},1\}$, $s' = \{s_{x,y} + \{1,1\},2\}$, and $s' = \{s_{x,y} + \{-1,1\},4\}$.

If $s_o \in s = 2$, $s' = \{s_{x,y} + \{1,0\},2\}$, $s' = \{s_{x,y} + \{1,1\},1\}$, and $s' = \{s_{x,y} + \{1,-1\},3\}$.

If $s_o \in s = 3$, $s' = \{s_{x,y} + \{0,-1\},3\}$, $s' = \{s_{x,y} + \{1,-1\},2\}$, and $s' = \{s_{x,y} + \{-1,-1\},4\}$.

If $s_o \in s = 4$, $s' = \{s_{x,y} + \{-1,0\},4\}$, $s' = \{s_{x,y} + \{-1,-1\},3\}$, and $s' = \{s_{x,y} + \{-1,1\},1\}$.

This definition is geometrically described by the transitions in Fig. 3

For the purposes of this paper, suppose that the overall 2D region is partitioned into a granular 10x10 grid, as shown in Fig. 2, with each partition side measuring at $l = 45$ meters. Each partition that is not an edge of the region or does not contain an obstacle is capable of experiencing fire within it. UAVs can transition throughout these partitions in the manner

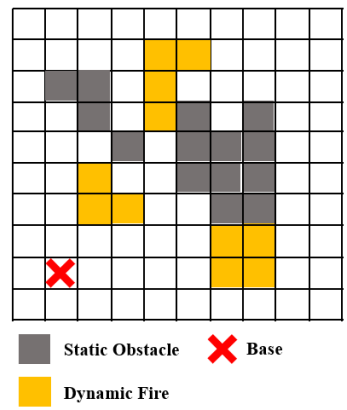

Fig. 2: 2D grid partition of problem location with environmental indicators

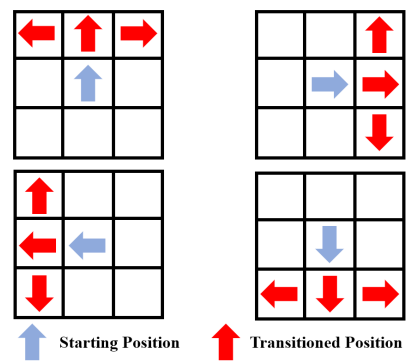

Fig. 3: Possible transitions for UAV within grid given starting orientation and location

displayed in Fig. 3 and have the ability to stop within a partition, indicating emergency landing. Every transition represents $l/v = 3$ seconds in real-time. From the above descriptions, the set of states of the UAVs, S, consist of their position in the grid, s_p, and orientation, $s_o \in \{1,2,3,4\}$. Additionally, the UAVs' states include the water level, $w \in \{0,1,2,3,4\}$ (each level tied to the percentages described before), and therefore $S = s_p \times s_o \times w$.

The fire behaves as follows. Each partition can contain $f \in \{0,1,2,3,4,5,6\}$ amount of fuel (i.e. forestry or shrubbery), starting with $f = 6$ for all partitions capable of holding fire. This results in an initial total fuel amount of $F = 300$ (subtracting region edges and obstacles). The fuel amount corresponds to how long a fire can burn within a partition, and as long as it is greater than 0, the partition can hold a fire. Each fire varies in intensity, with levels of Low, Medium, and High, that are each tied to a specific fire spread rate, f_r. These rates, shown in Table I, were pulled from the results data gathered in [15], which tested the amount of suppressant volume required to extinguish various spread rates of fires in a 1.5 by 3 meter test area. From the same test data, each rate requires a specific volume of water per square meter, v_r, to extinguish. A fire will grow in intensity and spread to adjacent partitions with time, t_f, determined by the partition length, l, divided by the spread rate of the fire. Each time a fire grows in intensity, all adjacent partitions that do not currently hold fire and do contain fuel will start holding fire, matching the intensity of the originating fire. Additionally, the fire will consume fuel in the partition,

an amount described by f_c, proportional to its intensity level. Each of the above described properties are listed in Table I.

TABLE I: Fire growth behavior

Fire intensity level	Low	Medium	High
Fire spread rate f_r (m/s)	0.035	0.040	0.045
Required suppressant volume per square meter v_r (L/m^2)	0.156	0.211	0.267
Time before fire spreads t_f (s)	1287	1125	999
Fuel consumption at intensity increase f_c	1	2	3

Each time one or multiple UAVs drop a combined amount of water, labeled v_{ud}, on a partition with fire, the total volume of water dropped on that partition, V_d, grows by v_{ud}. Furthermore, if V_d reaches the necessary amount v_r (provided in Table I) for the given fire intensity *before* the fire grows and spreads, the partition is considered extinguished with the current fuel amount, and fire can no longer spread to that partition. If the fire spreads before this volume of dropped water is reached, then V_d resets to 0.

Desired design specifications for each individual UAV are as follows. Each UAV must only drop a fraction of its water supply (transitioning from the currently held amount to next lowest) when flying over designated fire regions, and each UAV must return to base for replenishing water supplies when $w = 0$. Additionally, the UAVs may experience engine problems and must land for prolonged periods of time in the next available region not consumed by fire. Lastly, the UAVs must coordinate when asked, either prioritizing the same fires with multiple UAVs or different fires for each UAV.

As mentioned before, the design goal for this problem scenario is to create a high-level autonomous controller to dictate the overall behavior of the fleet for tackling any generic fire situation while maintaining the desired design specifications per UAV. For a given fire scenario, we seek to determine the required minimum number of UAVs. Given any number of UAVs, success is achieved when the fleet can permanently extinguish all fires while maintaining a minimum level of total fuel.

IV. Proposed Solution Method

We propose a solution to the formulated problem scenario that combines reactive synthesis with a dynamic allocation algorithm. These two methods form a high-level planner and controller that fulfills the design constraints imposed on each UAV and dictates the behavior of each one as well as their collective maneuvers. Fig. 4 depicts a conceptual view of the process of creating our solution and the duties that each method fulfills. Fig. 5 depicts the direct relationship between the allocation process and a synthesized controller.

As shown in Fig 4, a common synthesized controller is created for each of the UAVs through the RH framework with the duties presented by the indexed arrows. Given any arbitrary initial condition, the controller progresses to each partitioned space, these goals represented within Fig. 4 by the "nodes" protruding from each rectangle. The order that the controller meets these progress statements for a single UAV, shown previously through the ordering of $\mathscr{W}^i$ in Fig. 1, is not dictated by the synthesized controller as typically

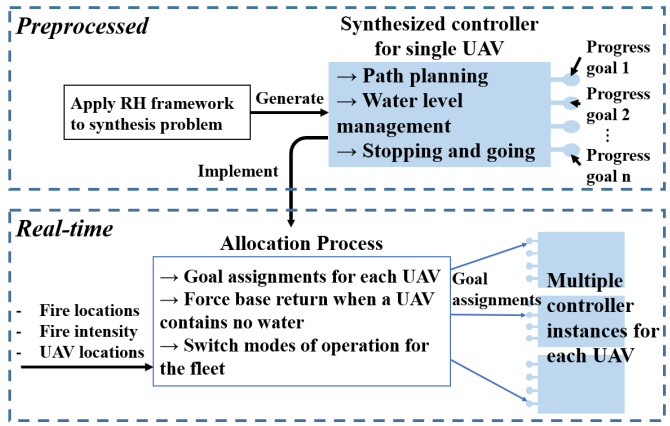

Fig. 4: Diagram of creating the solution method and the responsibilities and roles for both the synthesized controllers and the allocation process during real-time implementation

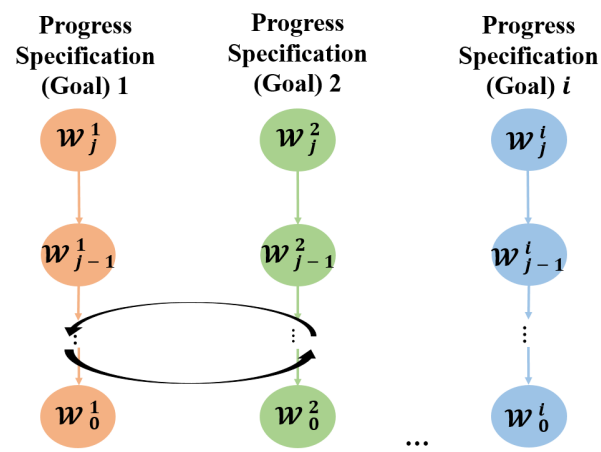

Fig. 5: Diagram of allocation process rearranging the progress goal ordering (as depicted in Fig. 1) for a single UAV controller instance in real-time

performed within the RH framework. Instead, the allocation process decides which progress specification any single UAV should pursue, represented within Fig. 5 by the switching of the "nodes'" order for any given moment in time. Hence, the allocation process prioritizes and assigns which goals a single controller should meet next in real-time. The combination of these two methods in the described manner works to highlight the strengths of each method. The synthesized controllers manage various system oriented aspects of the design and path planning while allocation governs the fleet behavior through assignments of goals for each controller.

Previously mentioned in [4], a physical scenario was constructed that dealt solely with one UAV gathering water, moving to another location, and dumping said water on the destination. [4] demonstrates the existence of lower level controllers that could manage the actions necessary to achieve the high-level planner and controller this paper proposes (at least for a smaller scale UAV). So, as often expressed in other sources dealing with high-level synthesized controllers, we assume there exists low-level controllers to dictate the motion of individual agents in real-time.

Remark: One caveat with regards to the solution explored in this paper is the, perhaps obvious, alternative solution.

Suppose the design was created through a more "handcrafted" approach, one without the use of formal methods for verification. Utilization of a path planning algorithm (such as any of the on-line algorithms discussed in [16]) for sending UAVs to any given destination (i.e. a fire or base) from any starting condition, could be combined with the described top-level allocation process that managed the destinations for all UAVs together. A huge benefit of such an approach is that the dedicated path-planning algorithm could handle a greater variety of dynamic environments than a synthesized controller. However, the described approach could easily be susceptible to programming errors due to the necessity of creating the conditions that dictate other task-oriented operations of the UAV (such as those involving control of the water level, knowing when to land under what conditions, etc.), precisely the issue that reactive synthesis aims to resolve. This paper's proposed solution method seeks to explore and expand the capabilities of reactive synthesis as a formal method within a more granular environment and complex task space.

V. Implementation

This section describes the construction and operation of the synthesized controllers and allocation algorithm used for the high-level controller. Additionally, the simulation used to test such cases is also outlined.

A. Synthesis of Controllers in Receding Horizon framework

For the common synthesized controller, environmental variables as perceived by a single UAV were created. These Boolean variables consist of: *StopSignal*, representing whether or not the UAV needs to stop due to periodic engine failures; *Fire*, representing the presence of fire directly beneath the UAV; and *DropWater*, representing allocation signaling a UAV to drop water on the assigned goal location. Hence, the environment is $E = StopSignal \times Fire \times DropWater$. The system consists of S as described in *Section III*. Additional APs $GoalPos_i$ and *Base* were created for when the UAV enters any specified goal location (indexed with $i \in I_g$) and the base location, respectively.

The overall environment specifications for each UAV are listed:

$$\varphi^e_{init} = \neg StopSignal \wedge \neg Fire, \tag{3}$$

$$\varphi^e_s = \{\}, \tag{4}$$

$$\varphi^e_p = \Box\Diamond DropWater \wedge \Box\Diamond\neg StopSignal \wedge \Box\Diamond\neg Fire, \tag{5}$$

Eq. (3) states that no UAV starts with engine failure or fire beneath it. Eq. (4) shows no guarantees on the environments behavior, and Eq. (5) states that always eventually: allocation instructs the UAV to drop water; the UAV will experience engine failure; and the UAV will not fly over fire.

The overall system specifications for each UAV are listed:

$$\varphi^s_{init} = \Phi \wedge (w = 4). \tag{6}$$

Eq. (6) reflects that a UAV will start in the position and orientation states allowed by Φ (tautology that governs feasible states) and will start with the full water supply.

$$\varphi^s_{s,1} = \Box(StopSignal \wedge \neg Fire \implies sys_actions = \text{"}Stop\text{"}), \tag{7}$$

Eq. (7) states that a UAV experiencing the engine fail signal and is not over fire will stop in the current partition.

$$\varphi^s_{s,2} = \Box(\neg(StopSignal \wedge \neg Fire) \implies sys_actions = \text{"}Go\text{"}), \tag{8}$$

Eq. (8) indicates that in the negation of the previous condition, the UAV must transition to new partitions.

$$\varphi^s_{s,3,i} = \Box((\neg(w = 0) \wedge DropWater \wedge \bigcirc GoalPos_i \wedge \neg(\bigcirc Base)) \implies ((\bigcirc w) = (w - 1)), \tag{9}$$

Eq. (9), which exists for every goal AP, states that the next water level decreases by 1 if: the water level isn't empty; the UAV is told to drop water into its goal; the UAV will transition into the goal next move; and the next move doesn't result in *True* for *Base*.

$$\varphi^s_{s,5,i} = \Box(((w = 0) \wedge DropWater \wedge \bigcirc GoalPos_i \wedge \neg(\bigcirc Base)) \implies ((\bigcirc w) = w), \tag{10}$$

Eq. (10) indicates that if the same conditions as Eq. (9) hold except the water level is at empty, the water level must stay the same.

$$\varphi^s_{s,4,i} = \Box((\neg(DropWater \wedge \bigcirc GoalPos_i \wedge \bigcirc Base \implies ((\bigcirc w) = w), \tag{11}$$

Eq. (11) states that the water level will remain the same as before if the UAV: is not told to drop water into its goal; is not transitioning into the goal next move; and/or is not transitioning to the base location.

$$\varphi^s_{p,i} = \Box\Diamond DropWater \implies GoalPos_i, \tag{12}$$

Lastly, Eq. (12) states that for every goal i, the UAV will transition into the goal location if the UAV is told to drop water. The resulting total reactive synthesis specification is displayed in Eq. (13).

$$\varphi^e_{init} \wedge \varphi^e_s \wedge \varphi^e_p \implies \varphi^s_{init} \wedge \varphi^s_{s,1} \wedge \varphi^s_{s,2} \wedge \bigwedge_{i=1} (\varphi^s_{s,3,i} \wedge \varphi^s_{s,4,i} \wedge \varphi^s_{s,5,i}) \wedge \bigwedge_{i=1} \varphi^s_{p,i} \tag{13}$$

Given any excessively large values for $i \in I_g$, the total specification would yield a synthesized controller that would first, take an excessive amount of time to generate (longer than the 12 hours taken for an attempt with a 3.2 GHz processor), and second, create no practical performance with regards to extinguishing fires. From such, though, the RH framework is applied to the synthesized controller, as followed from [3]. For every progress specification, the partitioned grid is further segmented into subregions $\mathscr{W}^i_j$, an example of which is shown in Fig. 6. As displayed, the regions are ordered by j based on their distance from the position that fulfills the particular progress statement (the i index is not shown).

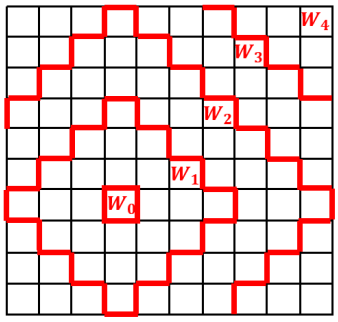

Fig. 6: RH partitions for progress statement centered on position (4,4)

Definition 3. *For all $s_{x,y} \in s_p$, $g_{x,y} = \{s \in S \mid s_{x,y} \in s\}$. In other words, $g_{x,y}$ is the set of states that contain the positional set $s_{x,y}$ that it was defined for.*

Definition 4. *Horizons are defined for each $s_{x,y} \in s_p$ as $\mathscr{W}_j^{x,y} = \{s \in S \mid |a[1]|+|a[2]| \leq 3j$ and $s \notin \mathscr{W}_{j-1}^{x,y}$, where $a = s_2 - s_{x,y}$ $(\forall s_2 \in s_p)$ and $j > 0\}$. Furthermore, $\mathscr{W}_0^{x,y} = g_{x,y}$. This definition is geometrically described by the horizons in Fig. 6*

For each set of partitions $\mathscr{W}_j^i$, GR(1) specifications (of the form in Eq. (2)) were constructed to enable correct transitions for the synthesized controllers. These specifications were differentiated between $j > 1$ and $j = 1$ through the progress statements and sets of initial conditions. For $j > 1$, the progress statements were constructed simply to force the UAV to transition into partitions contained within the next $\mathscr{W}$, an example of such shown in Fig. **??** through the highlighted partitions. For $j = 1$, the progress statements were constructed as discussed earlier with Eq. (12).

1) Receding Horizon Modifications and Proof:

Definition 5. *The existence of a walk of states, composed of a finite number of subsequent transitions in $s \to s' \in T \subseteq S \times S$, such that all $s \in S$ can be eventually reached from any other initial state $s \in S$ defines the system as reachable.*

Algorithm 1 $\mathscr{W}_j^{x,y}$ Modification during Synthesis

```
procedure SYNTHESIS_GOAL(x, y)                ▷
  Synthesis controllers from all initial conditions for the
  goal associated with location s_{x,y}
    for 0 ≤ j ≤ N do
        for s ∈ W_j^{x,y} do
            Synthesize controller given g_{x,y} and current s
            if Controller == None then
                Remove s from W_j^{x,y}
                Add s to W_{j+1}^{x,y}
```

Theorem 1. *Given a modified version of the system in Definition 1 which has removed some accessible states and transitions through the addition of static obstacles but still maintains the reachability property described in Definition 5, and by using the initial horizons $\mathscr{W}_j^{x,y}$ described in Definition 4 applied with the horizon modification algorithm described by Algorithm 1, the specification for each horizon surrounding each progress goal, Ψ_j^i, will remain realizable, preserving the RH framework guarantees on the overall specification.*

Proof. First, we maintain that the overall specification (Eq. 3-12) is realizable for the modified system description given no horizons and any allowable initial condition. Because of such, a horizon-based synthesized solution exists that fulfills the framework and definitions provided in [3].

Given the Definition 4 description of the receding horizons $\mathscr{W}_j^{x,y}$ for any individual goal $g_{x,y}$, reachability (Definition 5) implies that for any state sequence π that starts and leads from $s \in \mathscr{W}_j^{x,y}$ to a state $s_f \in g_{x,y}$, said sequence π must contain at least one $s \in \mathscr{W}_k^{x,y}$ for all $0 < k \leq j$. Under the modified system definition, all available π sequences for some $s \in \mathscr{W}_j^{x,y}$ may also need to include $s \in \mathscr{W}_r^{x,y}$ for some $r > j$, i.e. the only available path to the goal may require the state to move up horizons before moving back down due to the presence of obstacles. The possibility of such a sequence that includes paths with $s \in \mathscr{W}_{r>j}^{x,y}$ immediately violates the conditions for the receding horizon specification Ψ_j^i (defined by Eq. 2). To revert this violation, modifications to the horizons are made during synthesis in order to maintain the condition that a path π does not contain any $s \in \mathscr{W}_{r>j}^{x,y}$. This process is displayed in Algorithm 1.

As controllers are synthesized around each goal and for each initial condition s_i within each set $\mathscr{W}_j^{x,y}$, starting with $j = 0$ and incrementing, realizability failures are direct results of the lack of a system path to the horizon $\mathscr{W}_{j-1}^{x,y}$ that remains only in $\mathscr{W}_j^{x,y}$ under the assumed environment conditions. This is a result of the failure to satisfy Ψ_j^i since the overall global specification is realizable. Because a path starting at the initial condition s_i that fulfills the global specification must exist on the global scale and none of the sets $\mathscr{W}_j^{x,y}$ overlap per index (x,y), the path must enter into $\mathscr{W}_{j+1}^{x,y}$ due to the reachability property stated before. Through the algorithm, this state s_i is removed from $\mathscr{W}_j^{x,y}$ and added to $\mathscr{W}_{j+1}^{x,y}$. All intermediate states between the initial condition and horizon $\mathscr{W}_{j+1}^i$ are also moved to the next horizon since each state is tested as an initial condition in Algorithm 1, and these states cannot serve as viable initial conditions themselves. Therefore, the revised $\mathscr{W}_{j+1}^{x,y}$ contains the original set $\mathscr{W}_{j+1}^{x,y}$ plus all states from $\mathscr{W}_j^{x,y}$ that could not serve as initial conditions to reach the next horizon of $\mathscr{W}_{j-1}^{x,y}$ (or goal if $j-1=0$). This statement serves as a recursive assignment for each horizon j, shifting states back horizons until a new horizon set for a goal is defined such that each $s \in \mathscr{W}_j^{x,y}$ starts a path π contained solely in $\mathscr{W}_j^{x,y}$ that reaches $\mathscr{W}_{j-1}^{x,y}$. Because of this, Ψ_j^i is realizable for all goals, all horizons, and all initial conditions, maintaining the guarantees provided by the RH framework used from [3]. □

The benefit of the approach used by Algorithm 1 is that the viability test for an initial condition is made during synthesis and construction of controllers, saving on computation time since the horizons aren't evaluated and modified before synthesis. Hence the original horizon definition serves as a starting point that may or may not be preserved for each goal and is only modified when necessary.

B. Dynamic allocation

In this paper, we propose an efficient method for managing the dynamic allocation of UAVs to fire locations that spread with time. The allocation process considers two paramaters, fire density and proximity of the UAVs to fire locations. Regions with higher fire densities correspond to areas with a higher number of fires concentrated in a region. To partition the map into regions according to their fire densities, the first step in our allocation process is to carry out the K-means *clustering algorithm* via the Matlab built-in function, *kmeans*. We set this function to use the Manhattan distance as the evaluation index of similarity in order to group fires with similar locations into the same cluster. The *kmeans* function requires a desired number of clusters, k, as input.

To find the optimal k, k_{opt}, the elbow method is used. [Similar work on utilizing the elbow method and other clustering techniques is shown in [17].] In the elbow method, the process iterates through the possible number of clusters, k, from 2 to some maximum number of clusters, K_{max}, and the sum of square error (SEE) for each k is computed. Ideally, we are looking for a value of k that results in clusters that have a low SEE. This would lead to fires with high location similarity being assigned to the same cluster. The optimal number of clusters is the value of k at which the second derivative of the average of $SEE(k)$ is maximized. The idea is that the marginal drop in the average SEE as k increases will decrease dramatically for some value of k before it reaches a plateau, hence the "elbow criterion". The partition algorithm is summarized in Algorithm 2.

Algorithm 2 Partition algorithm

procedure CLUSTERING($K_{max}, fireLocs$)
  Set $k = 2$
  while $k < K_{max}$ **do**
    Compute clusters set $C = kmeans(fireLocs, k)$
    Compute mean sum of square error with k clusters, $SEE_{avg}(k)$
  Set optimal k, $k_{opt} = k$ that maximizes the second derivative of $SEE_{avg}(k)$
  Generate initial cluster centroid positions C_{init}
  Compute clusters set $C = kmeans(fireLocs, k_{opt}, C_{init})$
  Assign $fireLocs$ to their corresponding cluster $c \in C$,
  return C, fire location assignments

Once the map is divided into clusters, the number of UAVs that will be allocated to each cluster is defined by Eq. (14),

$$N_{alloc}(c) = \lceil \rho_c / \rho_{tot} \times (N_{tot} - N_{free}) \rceil, \tag{14}$$

where ρ_c is the cluster priority calculated as the sum of fire intensity values that belong to cluster c, ρ_{tot} is the sum of intensity values of all fires. N_{tot} is the total number of UAVs and N_{free} is the number of UAVs that have not been assigned to a cluster. For each cluster c, we assign a subset of UAVs chosen from all available UAVs. This subset of size N_{alloc} is comprised of those UAVs with locations closest to c. Once each cluster c has an assigned subset of UAVs, the next step is to decide on which fire location in c each UAV should go to.

For each cluster c, we pick the UAV in the subset of UAVs assigned to c that is the closest to c and determine the fire location f_{new} with the minimum cost relative to this UAV. This cost is calculated as the weighted sum of the distance between the fire and the UAV, the fire intensity and its distance from the centroid of c, and the switching fires penalty for the UAV. The rest of the UAVs in the subset are assigned to the same fire or their own lowest cost fire, explained in the next paragraph. We refer to this cost fuction as g as shown in Eq. (15).

$$\begin{aligned} \min_{f_{new} \in F} g(f_{new}, x) = d_f * \|x - f_{new}\| + s_f * \\ \left\|f_{prev} - f_{new}\right\| - n_f * (f_{int,new} + \|c_{centroid} - f_{new}\|), \end{aligned} \tag{15}$$

where d_f (assigned as 0.3 in our simulation) is a coefficient between 0.0 and 1.0 and is used as an importance weight for the distance between UAV location x and fire $f_{new} \in F$. F represents the set of all fire locations in cluster c. The coefficient n_f represents an importance weight for the fire intensity level, $f_{int,new}$, and the distance between the fire location and the centroid of cluster c. This coefficient is set as 1.0 - d_f. The first term in Eq. (15) seeks to drive UAVs towards their closest fires while the third term seeks to prioritize outer fires with higher intensities. Finally, coefficient s_f corresponds to the switching penalty for when a fire assignment, f_{new}, switches to a different fire than the previous assignment, f_{prev}, before the UAV reaches such (s_f assigned as 10 in our simulation). This prevents UAVs from going to different fires if they have not yet reached their previously assigned fire. If the UAV reaches its assigned fire, then the minimization is done over all other fires. The provided coefficient values were chosen heuristically during simulations with regards to performance.

We have two modes in which UAVs are assigned to fires, *Sync* mode and *NonSync* mode. In *Sync* mode, all UAVs can be assigned to any individual fire, allowing multiple UAVs to fight the same fire if said fire minimizes each of the UAVs associated cost function. In *NonSync*, no two UAVs are allowed to be assigned to the same fire at any given moment.

C. Simulation

To implement the solution and test its ability to meet the problem scenario, the synthesized controllers and allocation algorithm were constructed in MATLAB with a step-based simulation in mind, i.e. the simulation's main loop updated upon each move performed by the synthesized controllers. The time scale of the loop was adjusted to match that of a UAV's transition from one partition to the next, since most controller moves related to time revolved around such. From this, longer time-based actions, such as the fires' described spreading behaviors and the UAVs' wait times, were only updated when the amount of loops corresponding to the desired amount of time passed.

Furthermore, various fire scenarios' initial conditions were constructed for the purposes of testing the controllers. These fire scenarios were aimed at testing the capabilities of the UAVs to extinguish all fires without the fuel decreasing below

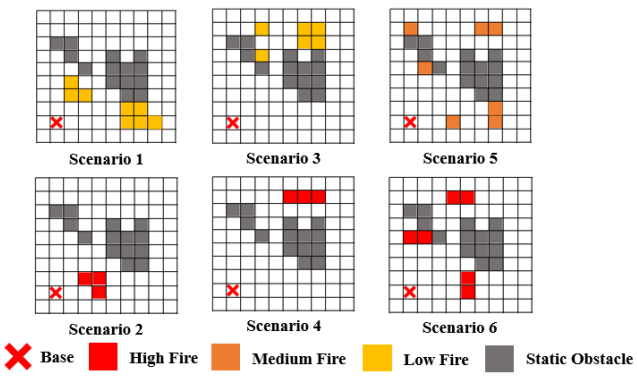

Fig. 7: Fire scenario visualizations, initial conditions

the minimum amount, with no exact focus on performance. The simulation cycled through iterations on each scenario, increasing the number of UAVs until the goal was met. Additionally, the scenarios aimed to capture a wide range of initial conditions and assess the differences between the allocation's *Sync* and *NonSync* modes. The exact initial fire placements are depicted in Fig. 7. For all scenarios, the UAVs started at the base location.

VI. Results

TuLiP [18] was utilized to realize and synthesize the controllers associated with each region $\mathscr{W}_j^i$. On an Intel i5-6500 CPU @ 3.20 GHz processor, this total process, approximately 250 regions $\mathscr{W}$, took on the order of 8 hours. In addition to the large amount of time to synthesize all of the individual controllers, numerous memory issues came up throughout the process, even with a system limit of 16 GB of RAM. The total size of the synthesized controllers was approximately 2 GB.

For each scenario tested, simulations were conducted 100 times to assess the fleet of UAVs' ability to meet the fire elimination and fuel sustainment goals described previously, provided each UAV experiences a 1% chance of engine failure for every transition time (3 seconds in all scenarios). The minimum fuel amount was chosen as 84, about one fourth the starting fuel amount. The results were compiled into Table II.

TABLE II: Simulation results

Scenario number (S - *Sync*, NS - *NonSync*)	Mean minimum number of UAVs required		Mean simulated real-time (s) for completion		Mean fuel remaining	
	S	NS	S	NS	S	NS
1	2.95	2.96	3871.2	4344.8	274.6	273.3
2	2.04	2.10	1910.6	1518.0	291.1	293.6
3	2.04	2.08	11932.7	11170.0	156.3	169.0
4	2.31	2.30	10224.8	10221.4	166.5	165.3
5	4.00	4.05	8641.8	8431.2	200.8	200.0
6	4.72	4.68	7500.1	7463.0	120.4	122.1

As shown by the mean values in Table II, each case required no more than 5 UAVs on average to meet the prescribed objectives. Marginal differences existed between the *Sync* and *NonSync* modes for each scenario, with the largest differences displayed between the time for completion and the remaining fuel. It would appear that the number of UAVs used by the allocation algorithm had a greater impact on the results than the mode used. This outcome is highlighted by the marginal differences between the modes for each scenario compared to

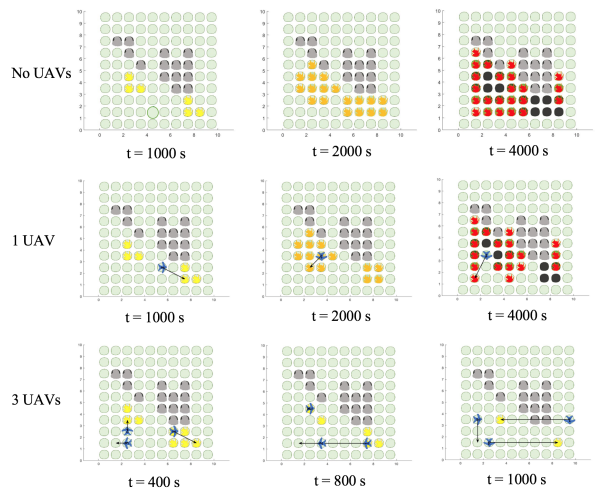

Fig. 8: Example of Scenario 1 (NonSync) results for no UAVs, 1 UAV, and 3 UAVs, with success shown in the last case

the differences between each scenario in general. Otherwise, in general, the longer the UAVs spent fighting the fires and the more intense the fires were resulted in greater fuel loss, an expected result.

A. Scenario 1

B. Scenario 2

C. Scenario 3

D. Receding Horizon Modification

An example time lapse for Scenario 1 is presented in Fig. 8, showcasing how the fire grew in response to a varying number of UAVs. The figure shows the current goal assignments for each UAV through the use of arrows pointing to such. When observing the simulations in real-time, the arrowheads switch to new locations anytime a UAV reaches the goal or a new assignment is made. These arrows help illustrate the assignments made during the simulations, such as UAVs prioritizing the outer edges of large fire clusters first or returning to the base location after running out of water.

VII. Conclusion and Future Work

In this paper, we constructed a high-level planner and controller to control a fleet of UAVs for various fire fighting scenarios. We simulated this method's ability to meet the objectives of eliminating all fires without losing more than a set amount of forestry through 6 varied scenarios, demonstrating success with no more than 5 UAVs needed on average.

Expanding upon the receding horizon framework for reactive synthesis allowed us to expand the scope of this problem while integrating the method with dynamic allocation for assigning UAVs. Even with such an approach, numerous issues arose throughout the process that help highlight key difficulties moving forward when using reactive synthesis in the control of UAVs. First, applying the $\mathscr{W}$ regions based primarily on position and ignoring orientations yields regions that prohibit specific initial conditions, especially with larger scale obstacles within the region. Our method of modified transition spaces circumvented the issues of synthesizing a controller at the

loss of our guarantees for meeting the top level specification when such spaces are used. Second, the RH framework, when considering all initial conditions, still yields an excessively large controller (about 2 GB) after 8 hours of runtime. Lastly, a simplified transition system was utilized which limited the total orientation space and interpreted UAV movement in only 2 dimensions, still far more restrictive than UAV movement in reality. Each of these points combine to exemplify the need for smarter partitioning of possible transitions a UAV can take in 3D space (easily dependent on at least 3 full degrees of freedom), should reactive synthesize be used for UAV control.

For improvements on this problem as it was explored, further evaluation of the effectiveness of our solution needs to be assessed with regards to more scenarios. Furthermore, performance metrics with regards to time should be built into the specifications to ensure timely reactions to the environment, improving efficiency at fighting the fires. Lastly, a greater detail simulation, one that models the reactions and behaviors of the fire and UAVs in real-time (instead of move-based increments), would aid in better assessment of our solution.

References

[1] D. Thomas, D. Butry, S. Gilbert, D. Webb, and J. Fung, "The costs and losses of wildfires: a literature survey," *Special Publication (NIST SP)*, Nov. 2017.

[2] D. Werner, "Fire Drones," *Aerospace America*, pp. 2831, 2015.

[3] T. Wongpiromsarn, U. Topcu and R. M. Murray, "Receding Horizon Temporal Logic Planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817-2830, Nov. 2012.

[4] H. Qin et al., "Design and implementation of an unmanned aerial vehicle for autonomous firefighting missions," *2016 12th IEEE International Conference on Control and Automation (ICCA)*, Kathmandu, pp. 62-67, 2016.

[5] B. Johnson, F. Havlak, H. Kress-Gazit, and M. Campbell, "Experimental Evaluation and Formal Analysis of High-Level Tasks with Dynamic Obstacle Anticipation on a Full-Sized Autonomous Vehicle," *Journal of Field Robotics*, 2017.

[6] K. W. Wong and H. Kress-Gazit, "Need-based coordination for decentralized high-level robot control," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, pp. 2209-2216, 2016.

[7] J. Alonso-Mora, J. A. DeCastro, V. Raman, D. Rus, and H. Kress-Gazit, "Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles," *Autonomous Robots*, 2017.

[8] Z. Liang, X. Yang and Y. Deng, "A partitioning-based task allocation strategy for Police Multi-Agents," *The 26th Chinese Control and Decision Conference (2014 CCDC)*, Changsha, pp. 2124-2128, 2014.

[9] B. Mirzapour and H. S. Aghdasi, "Modified multi-objective hybrid DE and PSO algorithms for resource allocation in forest fires," *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)*, Mashhad, pp. 187-192, 2017.

[10] P. Fiorucci, F. Gaetani, R. Minciardi, R. Sacil and E. Trasforini, "Dynamic resource allocation for forest fire risk management," *Proceedings. 15th International Workshop on Database and Expert Systems Applications, 2004*, pp. 603-607, 2004.

[11] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239-248, 2016.

[12] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive (1) designs, in Verification, Model Checking, and Abstract Interpretation." Springer, pp. 364-380, 2006.

[13] A. Ulusoy and C. Belta, "Receding horizon temporal logic control in dynamic environments," *The International Journal of Robotics Research*, vol. 33 , no. 12, pp. 1593-1607, Jul. 2014.

[14] X. C. Ding, M. Lazar and C. Belta, "Receding horizon temporal logic control for finite deterministic systems," *2012 American Control Conference (ACC)*, Montreal, QC, pp. 715-720, 2012.

[15] M. P. Plucinski, A. L. Sullivan, and R. J. Hurley, "A methodology for comparing the relative effectiveness of suppressant enhancers designed for the direct attack of wildfires," *Fire Safety Journal*, vol. 87, pp. 71-79, 2017.

[16] Liang Yang, Juntong Qi, J. Xiao and Xia Yong, "A literature review of UAV 3D path planning," *Proceeding of the 11th World Congress on Intelligent Control and Automation*, Shenyang, pp. 2376-2381, 2014.

[17] P. Bholowalia and A. Kumar, "EBK-Means: A Clustering Technique based on Elbow Method and K-Means in WSN," *International Journal of Computer Applications*, vol. 15, no. 9, pp. 17-24, 2014.

[18] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "TuLip: A software toolbox for receding horizon temporal logic planning," *Proc. Int. Conf. Hybrid Syst.: Computat. Control*, pp. 313-314, 2011.