Fire Fightin' UAVs! (Not actual title...)

Joshua Shaffer and Estefany Carrillo

***Abstract*—Abstract will focus on the motivations of this paper, implementation and results with a brief conclusion.**

***Index Terms*—Controller synthesis, Allocation processes, Receding Horizon control, Linear Temporal Logic.**

I. Introduction

REACTIVE synthesis provides a means for generating correct-by-construction controllers for complex systems. Synthesized controllers take into account varying types of dynamics for system and environment variables, progress and safety specifications, and initial conditions. The primary benefit of this method is the correct-by-construction attribute; the realized controller will always follow the specifications designed by the user as long as the environment falls within the assumptions made. Programmers are not required to handcraft individual behaviors of a system under specific conditions (of which are often error prone) and can instead focus on defining the system and specifications. The major pitfall of reactive synthesis, though, is the possibility of state explosion for highly granular state and environment definitions (even when the states are written in the standard GR(1) form).

Despite the aforementioned benefits, reactive synthesis is currently limited to a high-level view of systems with low granular abstractions and restrictive system and environment assumptions. Without this type of framework, most problems become infeasible to handle only with reactive synthesis. If, for example, an open environment (such as arbitrary obstacle placement) and a large state space (such as a 10x10 grid) were utilized to create a scenario involving the path planning of a robot that must always avoid any obstacle, synthesis of a solution would require an impractical amount of time. This stems from the fact that the synthesized controller must account for every possible environmental move (2^{100} permutations for arbitrary obstacles in a 10x10 grid). Unfortunately, real problems can typically involve this scale of permutations in the environment, hence the restrictions enforced on reactive synthesis.

When the high-level design of a system does involve this 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 [CITE] and the use of decentralized controllers for multiple agents in [Need-based Coordination for Decentralized High-level Robot Control]. Both examples break down the top-level synthesis problem into smaller, discrete pieces for the computation benefits. On the other hand, [DeCastros] approached their synthesis problem with a focus on resolving deadlock under specific environment conditions instead of directly avoiding dynamic obstacles, effectively limiting the number of permutations the reactive synthesis problem experienced and allowing lower level controllers to handle the real-time presence of 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. Specifically, in [CITE], the task space encompasses n progress statements, in [Need-based], the task space for a single robot encompasses n progress statements, and in [DeCastros], the task space for the 2nd example encompasses 4 progress statements.

This paper is motivated by the concept of combining reactive synthesis with other methods to work in tandem and provide a solution to a high-level problem with a large number of tasks to achieve. We seek to build upon the work described above by exploring a multi-agent system in a complex dynamic environment while meeting a task-space on an order of magnitude greater than the aforementioned task spaces.

Furthermore, we aim to explore a scenario involving unmanned aerial vehicles (UAVs) and their potential benefits to fighting wildfires. Current methods of fighting fires involve numerous man hours and piloted vehicles capable of dropping large amounts of suppressant over regions of fire. As the June 2015 issue of Aerospace America discussed in the article Firedrones [cite], UAVs of varying sizes currently pose a huge benefit to traditional firefighting methods through additional eyes in the sky and supply drops, and numerous agencies have spent the last decade exploring such options. While not a readily viable option, UAVs could potentially help extinguish fires on their own, as was mechanically demonstrated in [Design and implementation]. The scenario presented in [Design and implementation] dealt solely with one UAV gathering water, moving to another location, and dumping said water on the destination. These independent actions were performed through low-level controllers that were initiated by a high-level mission planner. Said mission planner, though, was not constructed to interpret the behaviors of multiple UAVs and fulfill complex specifications through its own volition, and as such, it was limited to only coordinating and enacting the lower-level controllers between two way-points. As often assumed in other sources dealing with high-level synthesized controllers, the availability of low-level controllers to dictate the motion of individual agents in real-time must exist to achieve the higher-level controller, of which are demonstrated though [Design].

The rest of the paper follows as such. In *Preliminaries*, we discuss subjects pertinent to the exploration of our topic.

J. Shaffer is with the Department of Aerospace Engineering, University of Maryland, College Park, MD 20740, USA e-mail: jshaffe9@terpmail.umd.edu.

E. Carrillo is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20740, USA e-mail: ecarril2@terpmail.umd.edu.

In *Problem Scenario*, we present the firefighting scenario, including environment and system definitions. Additionally, we discuss the exploration of possible solutions to such a problem. *Problem Solution* is dedicated to presenting our proposed solution, followed by *Implementation* which details how our solution is enacted, along with the construction of our simulation for testing such. *Results* present the outcomes to our tested cases, followed by the *Conclusions* section.

II. Preliminaries

In this section, we provide details about the commonly used terminology, notation, and equations surrounding reactive synthesis, particularly with respect to the application in this paper.

A. Linear Temporal Logic

To begin, linear temporal logic (LTL) is utilized for describing specifications within the reactive synthesis framework. LTL consists of infinite logic sequences upon a finite set of variables, of which make up the environment and system for this paper. Through these sequences, the behavior of a system can be described with respect to the actions of an environment.

LTL itself is built through atomic propositions, logic connections and temporal modal operators. Logic connections include $\neg$ (negation), $\vee$ (or), $\wedge$ (and), and $\longrightarrow$ (Not sure, results?). Temporal modal operators include $\bigcirc$ (next), $\Box$ (always), $\Diamond$ (eventually), and $\mathscr{U}$ (until). An atomic proposition (AP), p, is an LTL formula formed through the finite variable set, and propositional formulas (denoted with φ) consist of only logic connections on APs. Propositional formulas are also LTL formulas.

B. Reactive Synthesis

Reactive synthesis provides a method for generating controllers within the context of a defined environment and system. Specifications written in LTL are utilized to describe the desired behavior of the system and environment. One of the most commonly used forms for these LTL formulas is general reactivity(1) (GR(1)), the assume-guarantee form:

$$\begin{aligned}(\varphi^e_{init} \wedge \bigwedge_{i\in I_r} \Box\varphi^e_{s,i} \wedge \bigwedge_{i\in I_f} \Box\Diamond\varphi^e_{p,i}) \longrightarrow \\ (\varphi^s_{init} \wedge \bigwedge_{i\in I_s} \Box\varphi^s_{s,i} \wedge \bigwedge_{i\in I_g} \Box\Diamond\varphi^s_{p,i})\end{aligned} \tag{1}$$

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. Safety specifications describe properties that should always be fulfilled, while progress statements describe properties that should always eventually be fulfilled.

Piterman, et al. (CITE) proved that as long as the system and environment specifications are written in the framework shown in equation BLANK, the synthesis of a system controller that fulfills the system specifications while subject to the environment specifications will occur in polynomial time N^3. The proof of such, though, provides no guarantee that the polynomial time required will fall within a practical time-limit, due to the formulations limiting dependency on the total amount of permutations between the system and environmental states. (Correct way to state this?).

C. Receding Horizon

Various other sources have explored reactive synthesis within a receding horizon (RH) framework (X.C.,.. Ulusay,... T.). The basic premise revolves around, for each progress specification, segmenting the total system state space into regions W^i_j so that, when placed into properly constructed ordered sets represented by $F^i(W^i_j)$, the system variables will converge to meeting each progress statement for the system. Here, i represents the system progress statement $i \in I_g$, and j indexes the ordered regions W about the progress specification i. Following the basis laid out by (CITE), each region consists of its own GR(1) specification (shown in ()), constructed so that the synthesized controller will move the system variable towards the next region within the ordered set (eventually leading to $j = 0$) and fulfill the top level GR(1) specification.

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

In (), s refers to the system state. The formula Φ consists of all limitations on the states of system s, preventing the system from making transitions to or initializing within states that are not allowed.

The primary benefit of utilizing RH is the segmentation of the state space for both the environment and system. 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 (CITE), but any time-based optimization is forgone in this paper.

III. Problem Scenario

The problem scenario this paper explores is presented as such. A region of forested land, segmented by various large-scale obstacles, is experiencing a forest fire. The fire spreads from starting points in a flexible manner, and the starting fire conditions are arbitrary, i.e. any number of fires can occupy any number of subdivisions in the region. A base of operations exists at the edge of the region containing 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%), to Medium (60%), to Low (20%), and to Empty (0%). Each individual UAV contains a radio for communicating with base, GPS for determining position, and short-range radar for detecting nearby UAVs and their short-term trajectory.

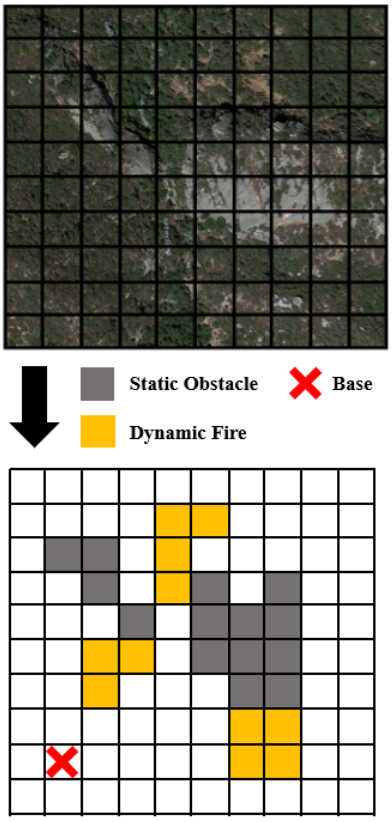

Fig. 1: 2D grid partition of problem location with environmental indicators

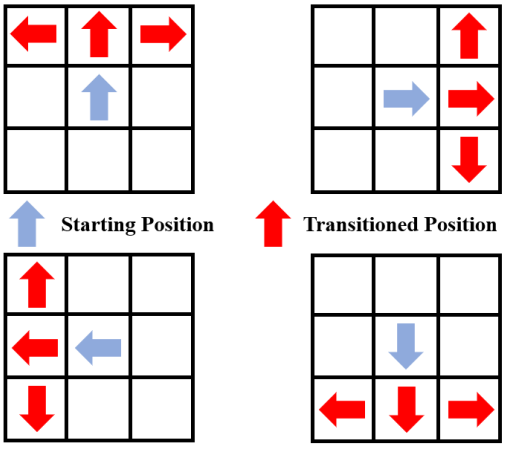

Fig. 2: Possible transitions for UAV within grid given starting orientation and location

For the purposes of this paper, suppose that the overall 2D region is partitioned into a granular 10x10 grid (units ignored, density 1), as shown in 1, with each forested partition (shown as empty) capable of experiencing fire within it. UAVs can transition throughout these partitions in the manner displayed in 2 and have the ability to stop within a partition, indicating landing. As such, the 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 ($w \in \{4,3,2,1,0\}$), and therefore $S = s_p \cup s_o \cup w$

The fire behaves as described. A partitioned subdivision contains f amount of fuel. As long as the fuel amount sits above 0, the subdivision can hold a fire. The fire itself varies in intensity, with levels of *Low*, *Medium*, and *High*. A fire will grow in intensity with time, the duration of growth diminishing at each higher level of intensity. Every time a fire grows in intensity, all adjacent partitions that do not currently hold fire and do contain fuel will begin holding fire, starting at the lowest intensity. These rules are detailed further in the *Implementation* section.

Design specifications for each individual UAV are as follows. Each UAV must only drop a fraction of its water supply (transitioning from the current amount to next lowest) when flying over designated fire regions (the amount dropped will result in varying degrees of effectiveness), and each UAV must return to base for replenishing water supplies when $w = 0$. Additionally, the UAVs 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, dropping water on the same location just as another UAV does so also.

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 and achieving the overarching goal of permanently extinguishing all fires. The open-ended nature of this design goal leaves plenty of flexibility in the remaining environmental design and allows for the exploration of the effectiveness of this papers proposed solution.

A. *Possible Solution Methods*

To begin, suppose a singular design approach is utilized for tackling this scenario, i.e. one methodology is utilized for achieving the high-level goal and design specifications.

First, attempting to meet all specifications with strong performance towards fighting fires through synthesizing a single controller to dictate the behavior of all the UAVs together would be impractical. As mentioned in the *Introduction*, the arbitrary nature of the fires combined with extensive system variables within the large state space provided creates an infeasible amount of possible permutations. Additionally, further system variables and specifications would need to be added for describing performance metrics and coordination. As discussed later on in this paper (in the *Implementation* section), breaking down the synthesis problem still provided excessive state spaces to explore, and the futility of attempting a singular synthesized solution became readily apparent.

Second, tackling design through an allocation process that manages which UAVs should go to which fires would provide a reasonable approach to managing the control of the fires through what is essentially an allocation of resources. An allocation process based on the one shown in [CITE] could incorporate a performance function for each UAV that weighs the distance of each fire to the UAV and the fires' intensities and proximity to other fires. The process would allocate each UAV to the fire that maximizes said performance function. On the flip side, though, such an allocation process would not manage information regarding the dynamics of the UAVs, control of their transitions and water levels, and determining when the UAVs should emergency land. An allocation process is well suited for a piece of the problem, but not necessarily the whole scenario.

Lacking on their own, the two methods presented above could combine to meet the desired design specifications for the presented problem scenario. While the combination of these two particular methods is plainly the explored solution for this paper, it is worthwhile to quickly discuss advantages and disadvantages of what is perhaps the more obvious solution method.

Suppose the design was created through a handcrafted approach, akin to many designs methods utilized in autonomous systems today (Need examples??). Utilization of a path planning algorithm (such as the one used in [d*]) 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. On the flip side, depending on the exact path planning algorithm, the path planning may not anticipate and handle obstacles much larger than its scale of operation. Additionally, 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, synchronizing with other UAVs, etc.). Furthermore, the syncing behavior for the UAVs might require another method to force such an outcome, or perhaps the allocation process would need to grow to accommodate such a behavior, which would require an understanding of the locations of the UAVs and how their possible transitions could enable synchronization. Evidently, a handcrafted approach to the problem would require isolation of all possible behaviors and encoding methods to resolve each, precisely the issue that reactive synthesis aims to avoid.

IV. Proposed Solution Method

In this paper, we propose a solution method that combines reactive synthesis with allocation. 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 individual UAV as well as their collective maneuvers. 3 depicts a conceptual view of the duties that each method fulfills and how each interacts with the other.

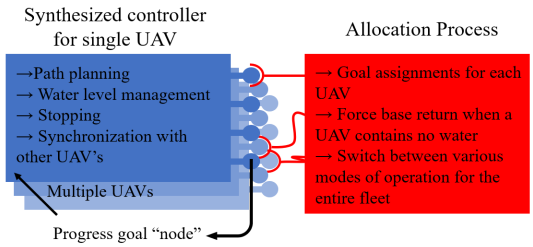

Fig. 3: Diagram of responsibilities and roles for both the synthesized controllers and the allocation process

As shown in 3, a common synthesized controller exists for each of the UAVs. This single controller dictates: high-level path planning through the partitioned space; the control of the water level; stopping during perceived engine failure; and synchronization with other UAVs. Each controller also aims to progress to each partitioned region given any arbitrary initial condition. The order of these progress statements, though, are not dictated by the synthesized controller. Instead, the allocation process decides which progress specification any single UAV should pursue. Hence, the allocation process is in charge of: assigning goals for the UAVs; forcing each UAV to return to home when water is depleted; and switch its own internal mode to prioritize different types of firefighting (synchronized fighting vs. non-synchronized fighting).

V. Implementation

A. Step I. Synthesis of Controllers in Receding Horizon framework

For the synthesized controller, the following environmental and system variables for a single UAV were created. The environment consisted of: *StopSignal*, representing whether or not the UAV needs to stop; *Fire*, representing the presence of fire directly beneath the UAV; and *SyncSignal*, representing the UAVs detection of whether or not a nearby UAV can enter the desired goal location. Hence, the environment E equals $StopSignal \cup Fire \cup SyncSignal$. The system consists of S as described in the *Problem Scenario* section. Additional APs $GoalPos_i$ and *Base* are used for when the UAV enters any specified goal location (indexed with i) and the Base location, respectively.

The overall environment specifications for a single UAV are listed as follows.

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

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

$$\varphi^e_p = \square\diamond SyncSignal \wedge \square\diamond \neg StopSignal \wedge \square\diamond \neg Fire \tag{5}$$

The overall system specifications for a single UAV are listed as follows.

$$\varphi^s_{init} = \neg\Phi \wedge w = 100 \tag{6}$$

$$\varphi^s_{s,1} = \square(StopSignal \wedge \neg Fire \rightarrow sys\_actions = \text{"}Stop\text{"}) \tag{7}$$

$$\varphi^s_{s,2} = \square(\neg(StopSignal \wedge \neg Fire) \rightarrow sys\_actions = \text{"}Go\text{"}) \tag{8}$$

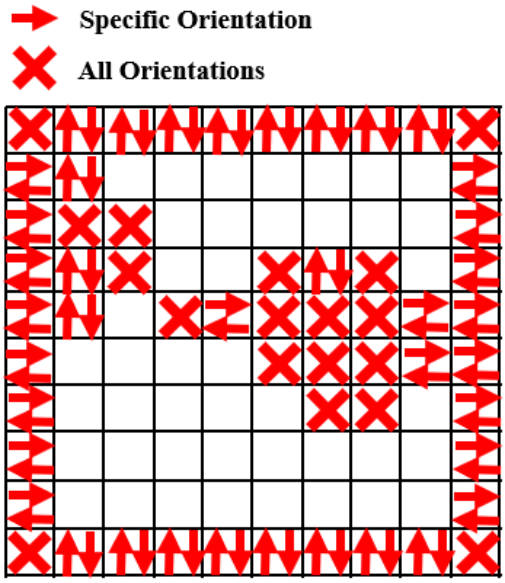

Fig. 4: All positions and orientations included in Φ

$$\varphi^s_{s,3,i} = \square((\neg(w=0) \wedge SyncSignal \wedge GoalPos_i \wedge \neg Base) \rightarrow (w \rightarrow w-1)) \quad (9)$$

$$\varphi^s_{s,4,i} = \square((\neg(SyncSignal \wedge GoalPos_i \wedge Base) \rightarrow (w \rightarrow w) \quad (10)$$

$$\varphi^s_{s,5,i} = \square(w=0 \wedge SyncSignal \wedge GoalPos_i \wedge \neg Base) \rightarrow (w \rightarrow w) \quad (11)$$

$$\varphi^s_{p,i} = \square\diamond SyncSignal \rightarrow GoalPos_i \quad (12)$$

As presented for the system, the initial conditions consist of all possible states except for those consisting of Φ, which consists of the states shown in . Safety specifications $\varphi^s_{s,1}$ and $\varphi^s_{s,2}$ dictate the motion of the UAV, with *sys_actions* representing whether the UAV should move or stay in the same state. The rest of the safety specifications pertain to how the UAV should control its water levels, such as dropping water over any goal location and refilling when it arrives at base. The set of progress specifications $\varphi^s_{p,k}$ (with k indexed through the total sum of possible fire locations) indicate that the UAV should visit all specified locations infinitely often *when the SyncSignal is true*.

Clearly, given any excessively large value of i, the total specification would yield a synthesized controller that would first, take an impractical amount of time to generate, 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 (CITE). For every progress specification, the partitioned grid is further segmented into subregions W^i_j, and example of which is shown in . As displayed, the regions are ordered based on their distance from the position that fulfills the particular progress statement.

For each set of regions W^i_j, a function $F^i(W^i_j)$ is defined as $F^i(W^i_j) = W^i_{j-1}$, leading the state down the ordered set towards W^i_0 and eventually fulfilling the desired progress statement. For each subregion W^i_j, an individual GR(1) form is defined as shown previously in ().

TuLiP (CITE) was utilized to realize and synthesize the controllers associated with each region W^i_j. On an Intel i5-6500 CPU @ 3.20 GHz processor, this total process, approximately 250 regions W, took on the order of 8 hours. On top of 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.

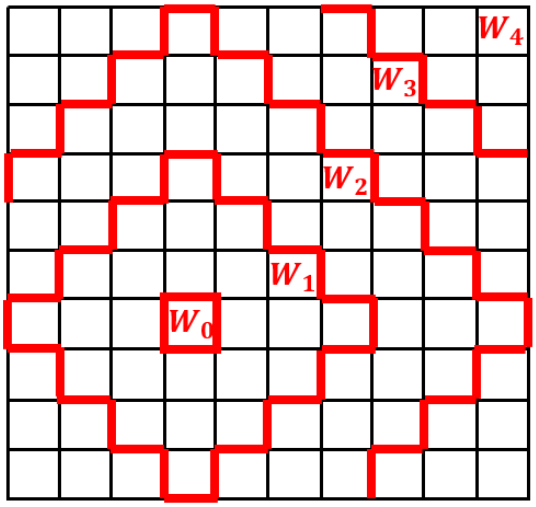

Fig. 5: RH Partitions for Progress Statement Centered on Position (4,4)

B. Step II. 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_{opt}, the elbow method is used. Similar work on utilizing the elbow method and other clustering techniques is shown in [X]. 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 I.

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

$$N_{alloc}(c) = \lceil \rho_c / \rho_{tot} \times (N_{tot} - N_{free}) \rceil, \quad (13)$$

Algorithm 1 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

where ρ_c is the cluster priority calculated as the sum of intensity values of fires 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 are not 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 the fire location belonging to c for each UAV to set as its goal. We have two modes in which UAVs can be assigned to fires, *Sync* mode and *NonSync* mode.

In *Sync* mode, for each cluster c, we loop through each UAV from the assigned subset to c and determine the fire with the minimum cost, proportional to its distance from the UAV, and highest priority, proportional to the fire intensity, according to function g as shown in (). The rest of the UAVs in the loop are assigned to the same fire as the first UAV in the loop.

$$\min_{f \in F} g(f,x) = d_f * \|x - f\| + s_f * \|g - f\| - n_f * intensity_f, \tag{14}$$

where d_f 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 $f \in F$. F represents the set of all fire locations in cluster c. The coefficient n_f represents an importance weight for the fire intensity level, $intensity_f$. This coefficient is set as 1.0 - d_f. Finally, coefficient s_f corresponds to the switching penalty of changing an already assigned fire to a different fire, if the assigned fire has not been reached yet. If the UAV reaches its assigned fire, then the minimization is done over all other fires.

In *NonSync* mode, for each cluster c, we also loop through each UAV from the assigned subset, but only one UAV is assigned to a single fire. The fire is chosen from the set of unassigned fires for each UAV according to (). Each time a fire is assigned, it is removed from the set F.

C. Step III. Simulation

To test the combination of the allocation process and the synthesized controllers, a simulation was constructed within MATLAB. The primary loop of the simulation incremented per move-set instead of with time, i.e. every action of the environment and reaction of the system occur within what is considered the same move. The environment consisted of the fires in each partition, the intensity of each fire, the number of moves before each fire intensity increased, the amount of fuel left in each partition, and a random engine failure signal that forced UAVs to land.

Variables pertaining to the fire behavior at each level of intensity are displayed in ()

TABLE I: Fire growth behavior

Fire Intensity Level	Required Number of Moves before Intensity Increase	Fuel Consumption at Intensity Increase
1	10	1
2	7	1
3	5	1

When a fire grew in intensity, it spread to adjacent partitions that contain fuel and no fire, starting new fires at intensity levels of 1. Additionally, if a fire was at intensity level of 3, it would continue to remain at 3.

A feedback loop for enacting the impact of water dropped on fires was also implemented. The amount of water dropped from UAVs into each partition was recorded at each move, and a rule set was followed for changing a partition's fire status from the dropped water. This rule set is displayed in ().

TABLE II: Fire response behavior

Fire Intensity Level	Amount of Water Dropped (% of UAV Tank)	Resulting Fire Intensity
1	$\geq 80\%$	0
1	$< 80\%$ and $\geq 40\%$	0
1	$< 40\%$	0
2	$\geq 80\%$	0
2	$< 80\%$ and $\geq 40\%$	1
2	$< 40\%$	1
3	$\geq 80\%$	0
3	$< 80\%$ and $\geq 40\%$	2
3	$< 40\%$	3

The response of any fire to water dropped was constructed around the possible drop amounts (corresponding to the allowed water level transitions with and without UAV synchronization). Additionally, when water dropped on a fire, the fire update counter restarted at 0 and the fuel level remained as it was before.

The simulation loop structure followed as such:

Algorithm 2 Simulation Loop

procedure SIMULATIONRUN
  Initialize UAVs locations and fire locations
  while Simulation Runs **do**
    Update fire growth
    Allocate UAVs based on fire state
    Move UAVs forward through synthesized controllers
    Implement feedback changes on fires due to water dropped

VI. RESULTS

Here go the results!

VII. Conclusion and Future Work

Here's a conclusion!

References

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

[2] T.Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. Tulip: a software toolbox for receding horizon temporal logic planning, in Proceedings of the 14th international conference on Hybrid systems: computation and control. HSCC, 2011, pp. 313-314.

[3] 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, 2016, pp. 62-67.

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

[5] B. Johnson, Havlak, F., Kress-Gazit, H., and Campbell, M. 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, 2016, pp. 2209-2216.

[7] Aerospace America, June 2015

[8] Ulusoy, Alphan and Marrazzo, Michael and Belta, Calin. (2013). Receding Horizon Control in Dynamic Environments from Temporal Logic Specifications.

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

[10] Multi-agent planning under local LTL specifications and event-based synchronization

Estefany Carrillo is an graduate student at the University of Maryland, College Park, in the Department of Electrical Engineering. She received her BS degree and M.S. in electrical engineering from University of Maryland in 2012 and 2017, respectively.