1. **Introduction and Motivation**

Reactive synthesis provides a means for generating correct-by-construction controllers for complex systems. Synthesized controllers take into account varying types of dynamics for the 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 benefits, reactive synthesis is currently limited to a high-level view of systems with low granularity in conjunction with restrictive system and environment assumptions. 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 that involves the path planning of a robot, 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 (2100 possibilities for arbitrary obstacles in a 10x10 grid). Real-life problems, though, typically involve this scale of permutations in the environment, hence the restrictions enforced on reactive synthesis.

When the high-level design of a system involves this scale of environmental permutations and state space, solutions typically seek to discretize the synthesis problem. This appears as the use of receding horizon in [CITE] or MBS in [CITE]. Clearly a gap still exists between designing the high-level controllers for robust autonomous systems that reside within reality and utilizing reactive synthesis to fully realize such designs. In truth, this gap will probably never close.

This paper is motivated by the concept of combining reactive synthesis with other methods to work in tandem for providing a solution to a high-level problem. Specifically, a complex scenario is constructed such that reactive synthesis alone could not realize and synthesize a controller for within a limited timeframe. From this scenario, a novel solution is explored which combines an allocation-based method with finite automata generated from reactive synthesis operating within a receding horizon framework to fully achieve the desired behavior for said scenario.

1. **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. 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, the environment and system for this paper. Through these sequences, the behavior of a system can be described.

LTL itself is built through atomic propositions, logic connections and temporal modal operators. Logic connections include ¬ (negation), (or), (and), and (Not sure, results?). Temporal modal operators include next, always, eventually, and until (symbols need added for each). An atomic proposition (AP), *p*, is an LTL formula formed through the finite variable set, and propositional formulas (denoted with ) consist of only logic connections on AP’s. Propositional formulas are also LTL formulas. (CITE), (CITE), and (CITE) each provide more in-depth overviews of LTL, while (CITE) provides a fully comprehensive look on the subject for readers unfamiliar with LTL.

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 GR(1), the assume-guarantee form, shown in equation *BLANK*.

From the above equation, the propositional formula describes the initial condition of the environment or system (denote by superscript *e* or *s*, respectively), describes safety specifications, and 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 ***N3*** (CITE). 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 (Correct way to state this?).

Various other sources have explored reactive synthesis within a receding horizon (RH) framework (2 CITES). The basic premise revolves around segmenting the total system state space into regions *W* that, when these regions are placed into properly constructed ordered sets, the system variables will converge to meeting each progress statement for the system. Each region consists of its own GR(1) specification, constructed so that the synthesized controller will move the system variable towards the next region within the ordered set. There exists a set of regions and ordered sets for each individual system progress statement.

The primary benefit of utilizing RH is the segmenting 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 is optimized, the total sum is not. Other sources have explored methods of working around this issue, primarily through some form of a performance index combined with iteration (CITE), but such a method is forgone in this paper.

1. **Problem Scenario**

The scenario 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 unmanned aerial vehicles (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.

For the purposes of this paper, suppose that the overall region is partitioned into a fairly granular 10x10 grid (denoted with ), as shown in Figure **Blank**, with each forested partition (shown as empty) capable of experiencing fire within it. UAVs can transition throughout these partitions in the manner displayed in Figure **Blank** and have the ability to stop within a partition, indicating landing. As such, the states of the UAVs () consist of their position in the grid () and orientation (). Additionally, the UAVs’ states include the water level (), and therefore .

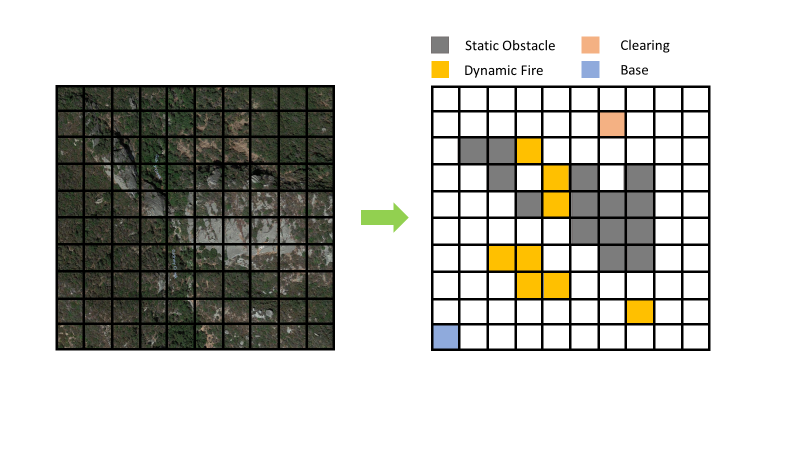


Figure **Blank**

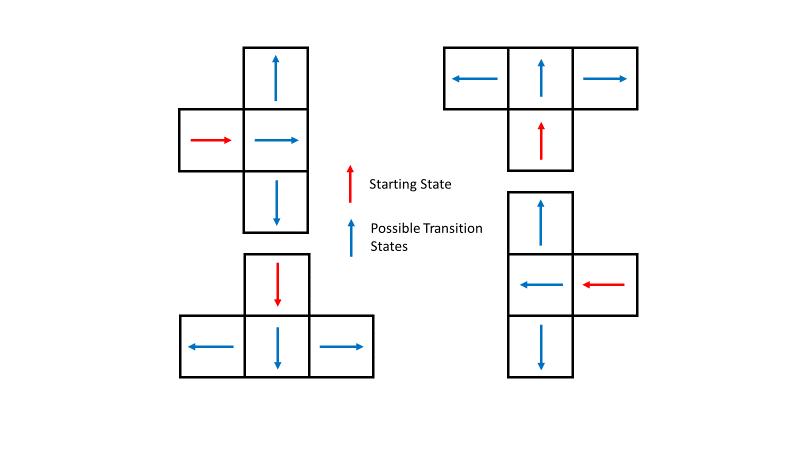


Figure **Blank**

Design specifications for each individual UAV are as follows: each UAV must only drop a fraction of its water supply (transitioning from 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 . 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 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 paper’s proposed solution.

1. **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 (provide 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. *Further describe an allocation process and what information it might have access to, such as UAV positions, # UAVs, fire locations.* 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. An allocation process is well suited for a piece of the problem, but not necessarily the whole scenario.

Lastly, 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 CITE) for sending UAVs to any given destination (i.e. a fire or base) from any starting condition, could be combined with a top-level allocation process that managed the destinations for all UAVs together. This approach, though, would easily be susceptible to errors due to the necessity of creating the conditions that dictated the operations of the UAV (such as those involving control of the water level and knowing when to land). Additionally, 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 might include understanding 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.

\*\*To note, the aforementioned method is not truly a singular design approach since it combines multiple methods together (allocation with common position-based UAV path planning), but this idea is introduced to highlight the possible strength of combining allocation with some form of path planning that managed other system variables (i.e. exactly what reactive synthesis could provide).

1. **Proposed Solution Method**

This paper proposes 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. Figure BLANK depicts a conceptual view of the duties that each method fulfills and how each interacts with the other.

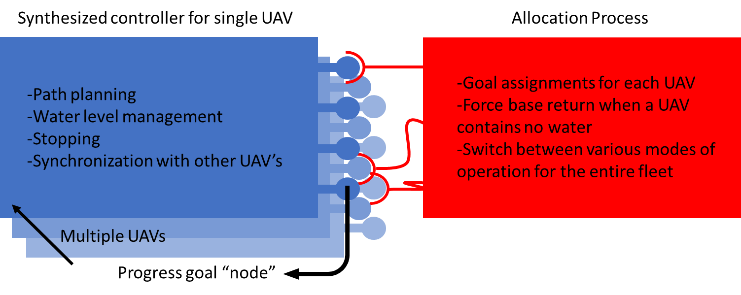


Figure Blank.

As shown in Figure Blank, a common synthesized controller exists for each of the UAVs. This single controller dictates 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 destination 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).

1. **Implementation**

For the synthesized controller, the following environmental and system variables were created. The environment consisted of: *Goal*, representing the Boolean desire of the allocation process for the UAV to dump water; *StopSignal*, representing whether or not the UAV needs to stop; *Fire*, representing the presence of fire directly beneath the UAV; and *SyncSignal*, representing the UAV’s detection of whether or not a nearby UAV can enter the desired goal location. Hence, the environment equals . The system consists of as described in the **Problem Scenario** section.

The relevant specifications for each UAV controller are listed. The environmental initial conditions are the entire possible set. The environmental safety conditions are none. The environmental progress conditions consist of each variable (i.e. each environmental variable will always eventually turn true). The system initial conditions consist of every possible starting location and orientation that does not exist on an obstacle or would lead directly into an obstacle or region edge. Additionally, all values (system water levels) are included in the possible initial conditions. Transitions are split between action sets titled “Go” (must transition to new position) and sets titled “Stop” (must stay within previous position). The safety specifications of the system are listed in Equations Blank.

Not sure how to best show these

Note that in the above equations, *Base* is an AP corresponding to the defined base location, and *GoalPos* is an AP corresponding to the defined goal location (i.e. any goal position). The progress statements are defined as shown in Equations Blank.

Not sure how to best show these

Again, these are defined for any goal location.

The reason why the specifications were written as shown is to highlight how the RH framework is applied to the synthesized controller, as followed from (CITE). For every feasible system position possible within the partitioned region, a progress specification drives the system to meet such. RH provides a methodology for fulfilling each of these progress specifications, and the overall partitioned region is further segmented into sets , an example of such shown in Figure Blank. As displayed, the regions are ordered based on their distance from the position that fulfills the particular progress statement.

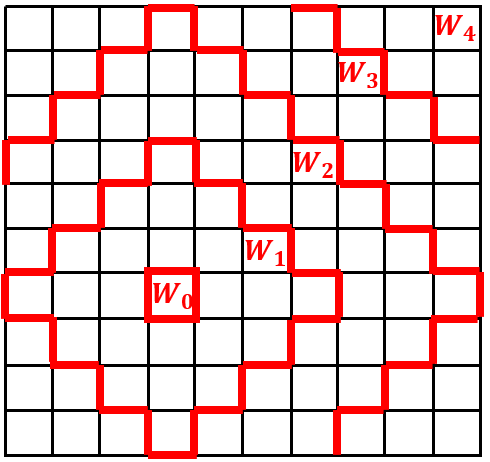


Figure **Blank**. RH Partitions for Progress Statement Centered on Position (4,4)

For each region *W*, an ordered set is defined such that *F(W)* will always lead down the ordered set, towards the *W* that fulfills the desired progress statement. For each region *W*, an individual GR(1) form is defined, as shown in Equation Blank.

RH GR(1) form

As shown in the equation above, an invariant *Phi* specification was also defined and added to the safety specifications. This invariant *Phi* details all system states that should never be entered for any partitioned region. In this case, *Phi* restricted the system from entering positions and orientations for which initial conditions were not allowed (cannot enter a position and orientation for which no transitions are possible or one that will only lead to such a scenario).

TuLiP (CITE) was utilized to realize and synthesize the controllers associated with each region *W* surrounding each progress statement. On an Intel i5-6500 CPU @ 3.20 GHz processor, this total process, approximately 250 regions *W*, took on the order of BLANK. 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 16 GB of RAM.

Detail the construction of the allocation process

To test the combination of the allocation process and the synthesized controllers, a simulation was constructed within MATLAB. The primary loop of this simulation incremented per move-set instead of 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 cell along with their defined behaviors and a random engine failure signal that forced UAVs to land.

Need to add description…

1. **Results**

Include different types of fires, i.e. list the imposed fire behaviors and changes to each. (would like around 5 types)

For each scenario, list the utilized allocation process, weights for it, and minimum number of UAVs needed to satisfy in a given amount of moves for the simulation

1. **Conclusions**

Discuss the issues with the constructed approach, but highlight what this combination of methods entails.

1. **References**

“Receding Horizon Control for Temporal Logic Specifications” Murray

“Multi-agent planning under local LTL specifications and event-based synchronization” Tumova

“Receding Horizon Temporal Logic Control for Finite Deterministic Systems” Ding

“Experimental Evaluation and Formal Analysis of High-Level Tasks with Dynamic Obstacle Anticipation on a Full-Sized Autonomous Vehicle” Johnson

“Synthesis of Reactive(1) Designs” Piterman