Fire Fightin' UAVs!

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 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 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 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 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 approach appears in the use of receding horizon control in [CITE] or MBS (need to change here) in [CITE]. Even with the problem space discretized, both examples presented rely on other methods to achieve the desired results. In [CITE],... and 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, attempting to close the gap presents a futile challenge, one that ....

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. Specifically, a complex scenario is constructed such that reactive synthesis alone could not realize and synthesize a controller for within a limited time-frame. From this scenario, a novel solution is explored which combines an allocation process with the finite automata generated from reactive synthesis operating within a receding horizon framework. The assembly of these two methods work to fully achieve the desired behavior for the constructed scenario.

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.

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 (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 APs. 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.

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 GR(1), the assume-guarantee form:

$$(\varphi^{e}_{init} \wedge) \tag{1}$$

From the above equation, the propositional formula ϕ_{init} describes the initial condition of the environment or system (denote by superscript e or s, respectively), $\phi_{s,i}$ describes safety specifications, and $\phi_{p,i}$ 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 N3. 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 (3 CITES). The basic premise revolves around, for each progress specification, segmenting the total system state space into regions W so that, when 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 and fulfill the top level GR(1) specification.

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 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.

III. 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 M 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 V), as shown in 1, with each forested partition

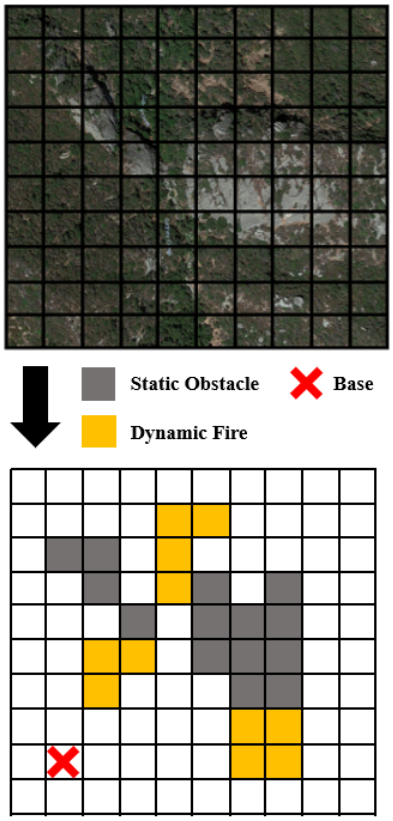

Fig. 1: 2D grid partition of problem location with environmental indicators

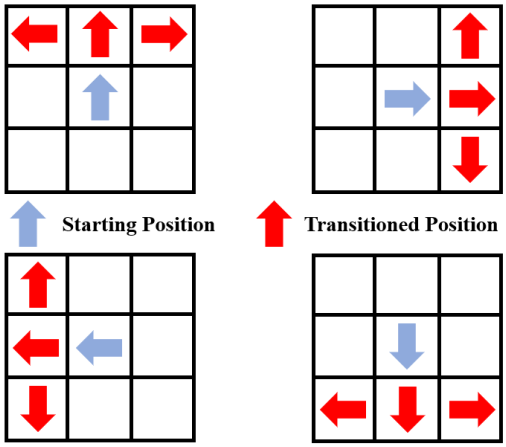

Fig. 2: Possible transitions for UAV within grid given starting orientation and location

(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_pV) and orientation ($s_o(1,2,3,4)$). Additionally, the UAVs states include the water level ($w(100,60,40,20,0)$), and therefore $S = s_p s_o w$

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 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 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 (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, number of 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

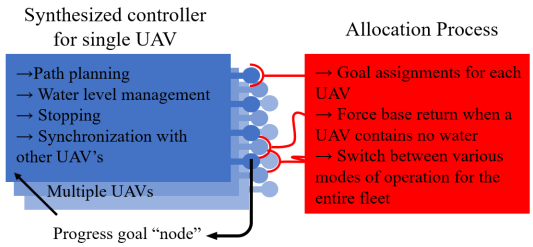

Fig. 3: Diagram of responsibilities and roles for both the synthesized controllers and the allocation process

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).

B. *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.

As shown in 3, 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).

IV. 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 UAVs detection of whether or not a nearby UAV can enter the desired goal location. Hence, the environment E equals (GoalłStopSignalł FirełSyncSignal). The system consists of (S) as described in the Problem Scenario section.

The relevant specifications for each UAV controller are listed. The environmental initial conditions ˙initˆe are the entire possible set. The environmental safety conditions ˙(s,i)ˆe are none. The environmental progress conditions ˙(p,i)ˆe consist of each variable (i.e. each environmental variable will always eventually turn true). The system initial conditions ˙initˆs 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 w 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 ˙(s,i)ˆs 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 ˙(p,i)ˆs 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 (excluding changes in orientation), a progress specification drives the system to meet such a location. RH provides a methodology for fulfilling each of these progress specifications by segmenting overall partitioned region into smaller sets W, 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.

For each set of regions W, a function is defined such that $F(W_i) = W_{i-1}$, leading the state down the ordered set towards W_0 and eventually fulfilling the desired progress statement. For each subregion W_i, an individual GR(1) form is defined as:

RH GR(1) form

As shown in the equation above, an invariant Φ specification was also defined and added to the safety specifications. This invariant Φ details all system states that should never be entered for any partitioned region. In this case, Φ restricted the system from entering positions and orientations for which

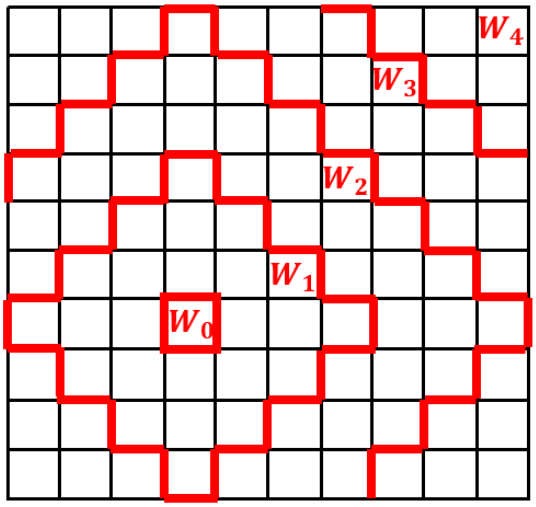

Fig. 4: RH Partitions for Progress Statement Centered on Position (4,4)

initial conditions were not allowed (i.e. 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 5 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.

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 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 cell along with their defined behaviors and a random engine failure signal that forced UAVs to land.

Need to add description

In the estimation process, the state of the system is expressed as a valuation of the state of individual components and uncontrollable contactors. Controllable and uncontrollable contactors are user-defined and remain unchanged throughout the estimation process. Let X be the state of the circuit, modeled as a random variable. First, the circuit is assumed to be in an unknown state $x_0 \in \Omega$, where Ω represents the set of all possible states. Let $V = \{(v_i)_{i\in\{0,\dots,t\}}\}$, represent the set of actions that can be performed on the controllable contactors. Let Y be the set of sensor measurements, with $y = \mu(v,x) \in Y$ representing the unique outcome of performing action v for a system state x. For actions $\{v_0,\dots,v_t\}$ and outcomes $\{y_0,\dots,y_t\}$ observed up until step t are represented by the partial realization $\psi_t = \{(v_i,y_i)_{i\in\{0,\dots,t\}}\}$. The estimation process adaptively gets to the actual state x_0. Let $D(y,v)$ with $y = \mu(v,x_0)$ be the set of states that are indistinguishable from x_0 under action v. Let $S_t = h(v_{0:t},x_0)$ be an extension of this set containing states that produce the same set of outcomes as

x_0 under the same set of actions $\{v_0,...,v_t\}$. At each time step t that a new action $v' \notin \psi_t$ is taken, there exists a recursive relation between the two sets of states:

$$h(v_{0:t} \cup \{v'\}, x_0) = h(v_{0:t}, x_0) \cap D(\mu(v', x_0), v'), \tag{2}$$

and S_t becomes:

$$S_t = \cap_{i \in \{0,...,t\}} D(\mu(v_i, x_0), v_i). \tag{3}$$

A strategy π is defined as a function of partial realizations to actions where $\pi(\psi_t) = v_{t+1}$. For an initial configuration with x_0, v_0 and y_0, the sequence of decision, measurement and update operations of the estimation process are expressed in Equations (3a)-(3c) respectively:

$$v_i = \pi(\psi_{i-1}) \tag{4a}$$
$$y_i = \mu(v_i, x_0) \tag{4b}$$
$$\psi_i = \psi_{i-1} \cup (v_i, y_i) \tag{4c}$$

The goal is to reduce the uncertainty in the state estimate, thus minimizing the number of indistiguishable states, by performing $k \in \{0,...,t\}$ actions that maximize the objective function f. This function maps the set of actions $A \subseteq V$ under state x_0 to reward $f(A, x_0)$, which measures the reduction in uncertainty of X represented by the probability distribution $\mathbb{P}[x]$ through performing k actions:

$$f(v_{0:k}, x_0) = -\mathbb{P}[S_k] = -\sum_{x \in S_k} \mathbb{P}[x]. \tag{5}$$

Thus, the estimation will find the strategy π that allows the best expected estimate for the state as shown in (4). We denote $|\tilde{V}(\pi, x_0)| \subseteq V$ the set of all the actions performed under the strategy π, the state of the system being x_0.

$$\pi^* \in \arg\max_{\pi} \mathbb{E}[f(\tilde{V}(\pi, X), X)], \tag{6}$$

subject to $|\tilde{V}(\pi, x)| \leq k$ for all x, and with expectation taken with respect to $\mathbb{P}[x]$. A greedy strategy is then used to select, at each step t, the action v_{t+1} that maximizes the expected one-step gain in uncertainty reduction with respect to all previous actions:

$$v_{t+1} \in \arg\max_{v \in \tilde{V}} \mathbb{E}[f(v_{0:t} \cup \{v\}, X) - f(v_{0:t}, X) | \psi_t]. \tag{7}$$

The overall goal is to design a strategy the fault detection controller runs to adaptively estimate the discrete state of the circuit by taking actions (i.e., closing and opening controllable contactors), and then reading voltage sensor measurements.

By changing the number and locations of sensors, it may be possible to improve state estimation performance. In the next section, a sensor placement algorithm is proposed to determine the number of sensors and locations with the goal of maximizing the performance of the state estimation method.

A. Sensor placement

Two algorithms explored in [11]-[12] for different applications are adapted to address our sensor placement problem. One proposes a combinatorial approach and the other a greedy approach, as shown in [11] and [12] respectively. Both approaches seek to maximize the state estimation ratio. This ratio is computed for a given sensor placement strategy P as follows,

$$r(P) = \frac{n_{succ}}{n_{tot}}, \tag{8}$$

where n_{succ} is the number of state configurations for which the state estimation algorithm returns a unique state (i.e. number of successful runs) and n_{tot} is the total number of state configurations.

The combinatorial algorithm is used as a reference since it considers all possible combinations of the desired number of sensors out of the total available sensors and outputs the combination of sensors that results in the maximum state estimation ratio. Unlike the combinatorial algorithm, the greedy algorithm does not need to consider all possible sensor combinations to achieve the same results, making it a much better candidate for our methodology. We use the ideas in [12] to develop our greedy algorithm and provide a theoretical description of this algorithm below.

1) Background: In [12], the problem of sensors and actuators placement is analyzed as a set function optimization problem of the form

$$\underset{P \subseteq L,\ |P|=k}{\text{maximize}} \quad r(P), \tag{9}$$

where $L = \{1,...,M\}$ is a finite set and $r : 2^L \to \mathbb{R}$ is a set function. L represents the set of potential locations for the placement of sensors or actuators and r is a metric for how controllable the system is for a given set of placements. After proving mathematically that r is a monotone increasing submodular function, a greedy heuristic is used to obtain a sub-optimal solution with guaranteed worst-performance bounds [19], [20]. For r to be submodular and monontone increasing, the following equations need to be satisfied respectively:

$$r(A \cup \{p\}) - r(A) \geq r(B \cup \{p\}) - r(B), \tag{10}$$

for all subsets $A \subseteq B \subseteq V$ and all elements $p \notin B$,

$$A \subseteq B \Rightarrow r(A) \leq r(B), \tag{11}$$

for all subsets A, $B \subseteq V$.

The greedy algorithm starts with an empty set, $P_0 \leftarrow \emptyset$, computes the gain $\Delta(a|P_i) = r(P_i \cup \{a\}) - r(P_i)$ for all elements $a \in V \setminus P_i$ and adds the element with the highest gain

$$P_{i+1} \leftarrow P_i \cup \left\{ \arg\max_a \Delta(a|P_i) | a \in L \setminus P_i \right\}. \tag{12}$$

2) Implementation: In our implementation of the greedy algorithm, we set the function r in (9) to be the state estimation ratio. Since this function is submodular monotone increasing, the greedy algorithm will provide sub-optimality guarantees as shown in [12]. Thus, the greedy algorithm obtains a sensor placement strategy by making a sequence of choices. At each step i, it assumes that i sensor locations are fixed and makes a

greedy choice where to place the $(i+1)$-th sensor. The sensor location added at each step is the one that maximizes the gain in the estimation ratio. The pseudocode for the greedy estimation algorithm is shown in Algorithm 1.

Algorithm 1 Greedy Sensor Placement Algorithm

```
procedure GREEDYSELECTION(r, p, e, k, n)
    Initialize the maximum estimation ratio r to 0.0
    Initialize the desired ratio p to 0.8
    Initialize the number of chosen sensors n to 0
    Initialize the sensor placement strategy S to {}
    while r < p and n < k do
        for each sensor location Li in Fig. 2 do
            if Li has not been selected already then
                Load the measurements database for S ∪ Li
                Calculate the estimation ratio e
                if r < e then
                    r = e
                    Lchosen = Li
        S = S ∪ Lchosen
        n = n + 1
    return sensor placement strategy
```

Once the sensors are connected in the locations determined by the strategy, the off-line step of the methodology is completed. Using this sensor placement, state estimation can be periodically run to monitor for faults. The state estimates obtained are passed to the controller synthesis in order to proceed with fault restoration if the state estimate reveals a fault in any component of the system. Fault restoration will generate actions to reroute power to ensure safe operation in the system. These actions are dictated by a control protocol built by verification and synthesis techniques described below.

B. Controller Synthesis

As shown in [13], [14] and [15], we can translate system requirements written in English to a temporal logic specification language (e.g., Linear Temporal Logic). These specifications are used to define correct behaviors of a system. Using this approach, a correct-by-construction controller can be designed to reconfigure the electrical power system to address different system failure scenarios. Whenever a system component becomes faulty, this would characterize a specific environment scenario and the controller is designed to react to it according to some high-level specifications. The output of the controller will be a different sequence of actions in each execution since the environment may be different. This captures the reactive nature of the controller. In [13], two approaches for designing the controller are the centralized and the distributed approach as described below.

1) Centralized synthesis: In order to design the controller, we first define the system model. The system variables are classified into sets of environment variables E and controlled variables P. Let $s = (e, p) \in dom(E) \times dom(P)$ be the state of the system. Consider an LTL specification φ of assume-guarantee form,

$$\varphi = \varphi_e \rightarrow \varphi_s, \tag{13}$$

where φ_e characterizes the assumptions on the environment and φ_s characterizes the system requirements. The synthesis problem is then concerned with constructing a controller strategy which chooses the action on the controlled variables based on the state sequence so far and the behavior of the environment so that the system satisfies the system requirements as long as the environment assumptions are satisfied.

The synthesis problem can be viewed as a two player game between an environment that attempts to falsify the specification and a controlled plant that tries to satisfy it. The formula φ is true if φ_s is true, and the system specifications are satisfied. When φ_e is false, i.e. the environment is inadmissible, the synthesis problem becomes unrealizable and there is no guarantee about the behavior of the system. To obtain LTL formulas, a set of atomic propositions needs to be defined. Any atomic proposition that belongs to this set is an LTL formula. The LTL formulas can express a wide range of system specifications regarding safety, performance, and reliability of the system. The LTL specifications are then transformed into General Reactivity of Rank 1, GR(1), specifications. It has been shown that realizability and synthesis problems for GR(1) specifications can be solved efficiently in polynomial time and GR(1) is expressive enough to provide complete specifications of many designs [17]. GR(1) is explored in depth in [16]. Thus, both φ_e and φ_s have the following structure:

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e;\ \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s, \tag{14}$$

where φ_i^e, φ_i^s are the initial values for the environment and controlled variables, φ_t^e, φ_t^s represent the evolution of the state of the system, and φ_g^e, φ_g^s represent goal assumptions for the environment and desired goal specifications for the system.

The GR(1) specifications are then passed to the synthesizer available in the temporal logic planning (TuLiP) toolbox. For a thorough description of this tool, see [18]. The synthesizer returns a finite-state machine in which states are valuations of environment and controlled variables and transitions are actions that the controller can take to reach a desired state.

When a controller has access to all environment and controlled variables, it functions as a centralized controller. As the number of components increases, the computational complexity can increase significantly. In order to address this and other issues such as system robustness and resilience to failure, a distributed controller strategy is also considered.

2) Distributed synthesis: By using a distributed controller, the system is divided into subsystems and a local controller is synthesized for each one of them. Thus, the synthesis task is divided into smaller subproblems and the computational complexity of the problem is reduced. Also, the distribution of power can be controlled and reestablished for each subcomponent of the system without affecting other subcomponents, leading to a more robust and resilient system.

Synthesizing a local controller requires the decomposition of global specifications into local specifications. For instance, let a system SYS be decomposed into SYS_1 and SYS_2. For i = 1,2, let E_i and P_i be the environment variables and controlled variables for SYS_i such that $P_1 \cup P_2 = P$ and $P_1 \cap P_2 = \emptyset$. The overall environment assumptions φ_e and system guarantees φ_s are distributed into the two systems SYS_1 and SYS_2. Let φ_{e_1}

and φ_{e_2} be LTL formulas containing variables in E_1 and E_2, respectively. Let φ_{s_1} and φ_{s_2} be formulas in terms of $E_1 \cup P_1$ and $E_2 \cup P_2$, respectively. Then, as long as the following conditions are satisfied, the distributed control protocol will satisfy the global specifications:

1) any sequence of actions from the environment that satisfies φ_e also satisfies $(\varphi_{e_1} \wedge \varphi_{e_2})$,
2) any sequence of actions of the system that satisfies $(\varphi_{s_1} \wedge \varphi_{s_2})$ also satisfies φ_s, and,
3) there exists two control protocols that realize the local specifications $(\varphi_{e_1} \longrightarrow \varphi_{s_1})$ and $(\varphi_{e_2} \longrightarrow \varphi_{s_2})$.

Additional information exchange between subsystems may be needed to make sure these conditions are satisfied. This can be done through an extra set of local guarantees that interact with other subsystems. For example, let S_1 have an extra set of local guarantees ϕ_1 that interact with S_2 as environment assumptions denoted as ϕ_1', while ϕ_2 are the local guarantees provided by S_2 that interact with S_1 as environment assumptions denoted as ϕ_2'. Thus, if the local specifications hold, then the global specification $\varphi_e \longrightarrow \varphi_s$ is realizable as shown in .

$$\phi_2' \wedge \varphi_{e_1} \longrightarrow \varphi_{s_1} \wedge \phi_1, \tag{15}$$

$$\phi_1' \wedge \varphi_{e_2} \longrightarrow \varphi_{s_2} \wedge \phi_2. \tag{16}$$

V. Test on a simple circuit

In this section, we give implementation details on applying the system methodology to the circuit topology shown in Fig. 2. This circuit topology consists of basic high-voltage AC components: two generators G0-G1 and two AC buses B1-B2, DC components: two rectifier units RU1-RU2 and two DC buses B3-B4, and contactors C1-C8. A discrete model for the circuit is chosen in order to make the integration with the controller synthesis method easier. In this model, generators and rectifier units can be in three states: offline (0), healthy (1) and unhealthy (2). Contactors can be in two states: open (0) and closed (1). Buses can be in two states: healthy (1) and unhealthy (0). Possible values for sensors healthy (1) and unhealthy (0) as well. Component variables and status variables are distinguished by upper and lower cases, e.g., the first generator is represented by G0, while its status is represented by g_0.

A. Sensor Placement Results

The first step of the implementation is to generate the databases of the inverse mappings from sensor measurements to compatible states of the circuit for all possible sensor placement configurations. For example, for a given sensor placement configuration, if a sensor were placed at L0, then the database reads a 0 for a fault at G0. Up to 8 voltage sensors can be placed in the circuit. Components (G1;G2;RU1;RU2;C2;C8) are set as uncontrollable and (C1;C3;C4;C5;C6;C7) as controllable. Using the databases obtained for each sensor placement configuration, we can execute the sensor placement algorithms which require the estimation ratio. For performance comparison, we use two versions of the state estimation algorithm to compute this ratio. In one version, the algorithm uses the full greedy strategy in which all possible configurations of the circuit components are simulated systematically as shown in Tables I and III. In the other version, the algorithm uses the reduced greedy strategy in which the initial configuration for the controlled components is fixed as shown in Tables II and IV. From the results obtained, we can conclude that the performance of the reduced version is better without compromising the accuracy of state estimates.

For the state estimation step, the greedy strategy is run with a horizon length of k = 6, the number of actions allowed. The status of all sensors is set to healthy. Both sensor placement algorithms were run in Python 2.7 on a 2.3 GHz Intel Core i7 64 bits CPU. The results are shown in Tables I-IV for various numbers of sensors. From these results, we can observe that the sensors selected by each sensor placement algorithm vary, but the same ratio is achieved. For both sensor placement algorithms, the maximum estimation ratio reached was 0.80. This result is the same as the one obtained after placing all available sensors in the system and running the state estimation by brute force method (i.e. where all possible configurations and actions are tested). Thus, for the given circuit set up, the state estimation algorithm can only uniquely identify the state in 80 percent of all possible states.

To improve this percentage, we consider changing the topology of the circuit. For instance, some uncontrollable contactors are set to be controllable. When contactor C2 is set to be controllable and sensor locations L5 and L6 are chosen, an estimation ratio of 1 is obtained. Allowing more contactors to be controllable improves the estimation ratio as shown in Table V. This happens because the dynamic estimation process has access to a wider range of possible actions that it can take and consequently can gather more information about the state.

TABLE I: Sensor Placement Strategy obtained by the Combinatorial Method with the full version of the adaptive greedy strategy

No of Sensors	Sensor Locations Selected	Estimation Ratio	Execution Time (s)
1	L7	0.14	190.17
2	L6, L7	0.39	871.35
3	L3, L6, L7	0.76	2203.25
4	L2, L3, L6, L7	0.80	3134.83
5	L5, L2, L3, L6, L7	0.80	2657.65
6	L4, L5, L2, L3, L6, L7	0.80	1396.82

TABLE II: Sensor Placement Strategy obtained by the Combinatorial Method with the reduced version of the adaptive greedy strategy

No of Sensors	Sensor Locations Selected	Estimation Ratio	Execution Time (s)
1	L7	0.14	30.25
2	L6, L7	0.39	177.64
3	L3, L6, L7	0.76	499.19
4	L2, L3, L6, L7	0.80	804.99
5	L5, L2, L3, L6, L7	0.80	787.45
6	L4, L5, L2, L3, L6, L7	0.80	432.88

TABLE III: Sensor Placement Strategy obtained by the Greedy Method with the full version of the adaptive greedy strategy

No of Sensors	Sensor Locations Selected	Estimation Ratio	Execution Time (s)
1	L6	0.14	173.68
2	L6, L1	0.39	393.92
3	L6, L1, L5	0.76	616.69
4	L6, L1, L5, L2	0.80	773.29
5	L6, L1, L5, L2, L7	0.80	908.26
6	L6, L1, L5, L2, L7, L3	0.80	432.88

TABLE IV: Sensor Placement Strategy obtained by the Greedy Method with the reduced version of the adaptive greedy strategy

No of Sensors	Sensor Locations Selected	Estimation Ratio	Execution Time (s)
1	L6	0.14	17.88
2	L6, L1	0.39	51.59
3	L6, L1, L5	0.76	89.72
4	L6, L1, L5, L2	0.80	131.48
5	L6, L1, L5, L2, L7	0.80	177.47
6	L6, L1, L5, L2, L7, L3	0.80	251.35

TABLE V: Sensor Placement Strategy obtained by the Greedy Method for various contactor configurations

Controllable Contactors	Uncontrollable Contactors	Sensors Selected	Estimation Ratio
C1	C2, C3, C4, C7, C8	L7, L6, L1, L0	0.0425
C1, C2	C3, C4, C7, C8	L7, L6, L1, L0	0.0825
C1, C2, C3	C4, C7, C8	L7, L6, L1, L0	0.205
C1, C2, C3, C7	C4, C8	L7, L6, L1, L0	0.800
C1, C2, C3, C7, C8	C4	L7, L6, L1, L0	1.000

While both sensor placement algorithms obtain the same success ratio, the greedy algorithm performs about twice as fast as the combinatorial algorithm. In additon, the greedy algorithm is well-suited for addressing the problem of selecting a required number of sensors and finding the minimum number of sensors that would satisfy an estimation ratio. Thus, we can conclude that the greedy algorithm is better overall.

With the greedy algorithm as the chosen approach, we proceeded to test it further for a scenario in which selected sensors become unhealthy. In this scenario, the greedy algorithm can be run again with new databases. These databases are obtained by simulating erroneous sensor readings corresponding to unhealthy sensors. For example, if an unhealthy sensor were placed at L_0, then the database reads a 1 for a fault at G0. In this simulation, contactors (C1;C2;C3;C7;C8) are controllable and contactor C4 is uncontrollable.

Given an estimation ratio of 0.6, the greedy algorithm can output a new sensor placement strategy once sensors become unhealthy. We test this for a given sensor placement in which sensor locations L6 and L7 are initially selected. The results we obtain are as follows. When the sensor placed at L7 becomes unhealthy, the new sensor placement strategy adds a sensor at L5 in order to achieve a ratio of 0.6. If instead sensor at L6 becomes unhealthy, the new sensor placement strategy adds sensors at locations L4 and L5 in order to obtain the same ratio. From these results, we can observe that some sensor locations are more informative than others and additional healthy sensors are needed in order to meet the given threshold.

B. Fault Detection and Fault Restoration Results

After connecting sensors to locations L4-L7 in the circuit and setting C4 as the controllable variable as determined by the sensor placement strategy for Fig. 2 in Table V, we proceed to implement the second step of the methodology in real-time. First, the state estimation algorithm is run for the circuit with a fixed fault configuration. The resulting state estimate is then passed to the controller synthesis method.

In order to run the controller synthesis method, we specify the system variables, assumptions on admissible environments and the desired system specifications for the circuit in Fig. 2. The circuit is modeled as a system with environment variables (g_0;g_1;ru_1;ru_2) for the generators and rectifier units, controlled variables (c_1-c_8) for the contactors, and dependent variables (b_1-b_4) for the buses. The overall goal of this method is to reconfigure the controlled variables so that power will be delivered to buses and guarantee the following system specifications:

Environment Assumption: At least one generator and rectifier unit is always healthy. Also we assume that once a generator becomes unhealthy, it will remain unhealthy. The specifications to accomplish this are:

$$\square\{(g_0 = 1) \vee (g_1 = 1)\}, \tag{17}$$

$$\square\{(ru_1 = 1) \vee (ru_2 = 1)\}, \tag{18}$$

$$\square \bigwedge_{i=0}^{1} \{(g_i = 0) \longrightarrow (\bigcirc(g_i = 0))\}. \tag{19}$$

Unhealthy Sources: All neighboring contactors to generators and rectifier units that become unhealthy should be set to open. Set of neighboring contactors to generators are N(G0) = C1 and N(G1) = C2. Neighboring contactors to rectifier units are N(RU1) = {C4, C6} and N(RU2) = {C5, C7}. For example, if generator G0 becomes unhealthy, contactor C1 will be set to open. In this instance, the specification for G0 is:

$$\square\{(g_0 = 0) \longrightarrow (c_1 = 0)\}. \tag{20}$$

No Paralleling of AC Sources: In Fig. 2 there is one generator pair {G0,G1}. We avoid instances of paralleling ac sources. For example, a live path for the pair G1,G2 exists if contactors C1, C2 and C3 are all closed. The specification to disallow paralleling of AC sources is:

$$\square\neg\{c_1 = 1 \wedge c_2 = 1 \wedge c_3 = 1\}. \tag{21}$$

Power Status of Buses: A bus will be powered if a live path exists between itself and a generator. A DC bus will be powered if it is connected to a healthy rectifier unit and a live path exists between itself and a powered AC bus. Otherwise, the bus will be unpowered. For instance, the specifications for AC bus B1 and DC bus B6 to be powered are:

$$\square\{(c_1 = 1 \wedge g_0 = 1) \longrightarrow (b_1 = 1))\}, \tag{22}$$

$$\square\{(g_1 = 1 \wedge c_2 = 1 \wedge b_2 = 1 \wedge c_3 = 1) \longrightarrow (b_1 = 1)\}, \tag{23}$$

$$\square\{(b_1 = 1 \wedge c_4 = 1 \wedge ru_1 = 1 \wedge c_6 = 1) \longrightarrow (b_3 = 1))\}, \tag{24}$$

$$\square\{(b_2 = 1 \wedge c_5 = 1 \wedge ru_2 = 1 \wedge c_7 = 1 \wedge b_4 = 1 \wedge c_8 = 1) \longrightarrow (b_3 = 1)\}. \tag{25}$$

Essential Buses: Another specification we consider is that buses B1 and B2 be connected to safety-critical loads, and can be unpowered for no longer than two time steps. Each increment of the clock variable θ_{B_1} and θ_{B_2} represents one time step δ=1. Clock variables function as counter variables and are either increased by 1 or reset to 0 at each step of the execution.

If bus status B1 is unpowered, then at the next time step, clock θ_{B_1} increments by one:

$$\square\{(b_1 = 0) \longrightarrow (\bigcirc\theta_{B_1} = \theta_{B_1} + 1)\}. \tag{10}$$

If bus status B1 is powered, then at the next time step, reset clock θ_{B_1} to zero:

$$\square\{(b_1 = 1) \longrightarrow (\bigcirc\theta_{B_1} = 0)\}. \tag{11}$$

To ensure that the status of B1 is never unpowered for more than two steps, we have:

$$\square\{\theta_{B_1} \leq 2\}. \tag{12}$$

We also require that all DC buses must always remain powered by:

$$\square\{b_3 = 1 \ \wedge \ b_4 = 1\}. \tag{13}$$

We now turn to define the initial, transition, and goal values for the synthesis in GR(1) form:

$$\varphi_i^e = \bigwedge_{i=0}^{1}\{(g_i = 1), (r_i = 1)\} \tag{14}$$

$$\varphi_i^s = \bigwedge_{i=1, i\neq 3}^{8}\{(c_i = 0)\} \tag{15}$$

$$\varphi_t^e = \vee_{i=0}^{1}\{(\bigcirc\neg g_i), (\bigcirc\neg r_i)\} \tag{16}$$

$$\varphi_g^e = \square\diamond(True) \tag{14}$$

$$\varphi_g^s = \square\diamond(True) \tag{15}$$

In the following simulation, we use the centralized controller design. The synthesis method is initiated with the following fault configuration: G0 = 0, G1 = 1, RU1 = 1, RU2 = 0, C2 = 0, C7 = 1. After running the state estimation algorithm with the reduced version of the greedy strategy, a state esimate is obtained in 6.48s. Then, the controller synthesis is run with the specifications and variables set as described above. Processing the specifications and obtaining the controller strategy took 0.58s. In this simulation, a centralized approach is used since the controller has access to all system variables.

In Fig. 3 - 5, we show a simulation trace for the reactive controller. The controller determines which adjacent contactors should be closed to allow power to flow from a healthy component into the components that connect to it or set to open to prevent power flow in from failing components. Assume that at time t = 0, all components of the circuit are in a healthy state as shown in Fig. 3. At time t = 1, G0 and RU2 become unhealthy. The controller then outputs a sequence of actions that reconfigure the circuit by opening contactors C2, C5 and C7 and closing contactor C3 as shown in Fig. 4. Then at time t = 2, G1 and RU1 become unhealthy and the controller reacts by opening contactors C2, C4 and C6 and closing contactor C3 as shown in Fig. 5.

VI. Case Study on a Large Circuit

In this section, we test the scalability and feasibility of the proposed methodology as we examine a large circuit topology shown in Fig. 6. All three algorithms of the methodology are performed on this circuit using a centralized approach and a distributed approach. First, a description of the circuit topology is presented followed by the results obtained from applying the methodology to the circuit.

A. Circuit Topology Description

At the top of the circuit topoloy are six AC generators G1-G6. An AC bus is connected to each AC generator. There are 8 AC buses in total. The circuit also consists of rectifier units R1-R8, each connected to a DC bus. R1 and R2 connect to the same DC bus. R3 and R4 also share the same DC bus. All other rectifier units connect to one DC bus each. In total, there are 6 DC buses and 33 contactors. From Fig. 6, it can be observed that the circuit topology is symmetric.

1) Environment Variables: The environment variables consist of all generators G1-G6 and all rectifier units R1-R8. The health statuses for generators and rectifier units can take values of healthy (1) or unhealthy (0). At each step of the simulation, the environment variables can take a status of healthy or unhealthy subject to the assumption that one generator in the set: {G1, G2, G5} and one generator in the set: {G3, G4, G6} remain healthy. A similar assumption is put into place for rectifier units, at least one rectifier in the set: {R1, R2, R5, R6} and one rectifier in the set: {R3, R4, R7, R8} remain healthy. We impose these assumptions when decomposing the circuit so that local specifications can be satisfied in the right side of the circuit as well as in the left side.

2) Controlled Variables: The controlled variables consist of all contactors C1-C33. Contactors can each take values of open (0) or closed (1). All contactors are assumed to be directly controlled without delays. AC and DC buses are considered as dependent variables since their statuses can be either powered (1) or unpowered (0) depending on the status of neighboring contactors, rectifier units and generators.

B. Centralized approach

A system model with environment variables E and system variables P, including controlled and dependent variables, is used to represent the circuit. Given environment assumptions φ_e as in (10)-(12) and system assumptions φ_s as the conjunction of all specifications from (23)-(33), the synthesis problem is to find a control protocol such that $\varphi = \varphi_e \rightarrow \varphi_s$ holds. The output of the synthesis procedure is a discrete planner represented as a finite-state automaton where states are valuations of environment and controlled variables.

The simulation is run on a 2.6 GHz Intel Core i7 processor with 16 GB memory. For the sensor placement algorithm, L1 - L27 are set as possible sensor locations. We attempted to run the sensor placement algorithm and controller synthesis for the entire circuit. The program run for more than 10 hours after which it was stopped without obtaining an output. Due to the computational complexity of building the databases for

the entire topology as well as performing synthesis for a large number of states, we opted to execute the methodology using a distributed approach.

C. Distributed approach

The entire circuit is divided into four subcomponents when applying the distributed approach. We take advantage of the symmetric properties of the circuit and select its subcomponents so that they share the same topology. Subcomponent A has the same topology as subcomponent B, while subcomponent C has the same topology as subcomponent D. A and B are shown in Fig. 6. C is the bottom right part of the circuit while D is the bottom left part as shown in Fig. 7. First, the sensor placement step of the methodology is applied to each subcomponent of the circuit. For all subcomponents, a threshold of 0.7 is provided. For subcomponents A and B, we set (C1;C3;C4;C5;C6;C7) and (C9;C11;C12;C13;C14;C15) as controllable contactors respectively. The sensor placement strategy obtained included sensor locations L1 and L2 for A and sensor locations L6 and L7 for B. It took 42.35 seconds to build and load the measurements databases and 9.31 seconds to output the strategy for each subcomponent. For subcomponents C and D, all contactors except for C25 in C and C33 in D were set as controllable. Since C and D are connected to A and B respectively, we opted to model A and B as additional generators in C and D respectively. The sensor placement strategy obtained selected sensor locations L12 and L13 for C and sensor locations L17 and L18 for D. It took 72.4 seconds to build and load the measurements databases and 72.6 seconds to output the chosen sensors for each component.

Using the sensor placement strategies obtained above, we proceeded to test the state estimation algorithm. To do so, 10 different fault configurations were simulated for A as well as for C. For A, 8 of the states simulated were uniquely identified. On average the state estimation process took 6.16 seconds. For C, all of the states simulated were uniquely identified. On average the state estimation process took 12.67 seconds. In the final step, we proceeded to run the controller synthesis using the specifications described in Section II-D assuming two scenarios. In the first scenario, it is assumed that contactors C8 and C34 are set to open at all times, disconnecting A and B. When the controller synthesis was performed for each of these subcomponents, it took 0.48 seconds to solve for a control protocol with 25 states for each subcomponent. The resulting control protocol was able to guarantee power for the essential buses as long as one generator and one rectifier was kept healthy at all times in each of the subcomponents. In this scenario, the behaviors of A and B were independently controlled by each of their control protocols.

In the second scenario, we allowed A and B to be connected. We also changed the assumption requiring both of these subcomponents to have at least one generator and one rectifier healthy at all times to require only one of these subcomponents to have at least one generator and one rectifier healthy at all times. Let this assumption be placed on B. Placing this assumption ensures that for any allowable sequence of environment actions, the controller is able to supply power to A at any step. Health status information for G1 and G2 is sent to B via a health status variable $h1$. The variable is set to 0 if neither source is healthy, and 1 if either g1 or g2 is healthy. If health status $h1 = 0$, i.e., both g1 and g2 are unhealthy then, whenever the AC bus connected to G3 is powered, c8 will be set to closed. Similarly, if r1 = 0 and r2 = 0, then c34 will be set to close. Running the controller synthesis for this scenario took 41.2 seconds solving for a control protocol with 341 states. For both scenarios, running the controller synthesis for C took 40.48 seconds to solve for a control protocol with 257 states. The same results were obtained for D.

VII. Results

Here go the results!

VIII. 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.

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.

Huan Xu is an assistant professor at the University of Maryland, College Park, in the Department of Aerospace Engineering and the Institute for Systems Research. She received her BS degree in mechanical engineering and material science from Harvard University in 2007, and M.S. and Ph.D. in mechanical engineering from the California Institute of Technology in 2008 and 2013, respectively. Her work focuses on the use of formal methods and controls in the design and analysis of unmanned autonomous systems.