AIAA Guidance, Navigation, and Control Conference
10 - 13 August 2009, Chicago, Illinois

AIAA 2009-6215

Specification and Planning of Interactive UAV Missions in Adversarial Environments

Sertac Karaman * Mariam Faied † Emilio Frazzoli * Anouck Girard †

In this paper, specification and planning of UAV missions, in which interaction with the operators, adversaries, and the environment plays a crucial role, are studied. A novel specification method is introduced to model interactive tasks in UAV missions and process algebra framework is used to model complex interactive tasks from relatively simple ones. An anytime state-space search algorithm is proposed. The algorithm effectively searches for a feasible solution and improves on such solution over time, eventually terminating with an optimal solution.

I. Introduction

Over the last decade, Unmanned Aerial Vehicle (UAV) missions took crucial role in tactical military operations. Mainly used for surveillance, monitoring, and reconnaissance, UAVs can provide an increased situational awareness to military personnel both in the field and at off-site mission control facilities. UAVs also show great potential for civilian applications, especially for disaster response in situations such as forest wildfires, earthquakes, or floods. In many of these civilian applications, UAVs can provide monitoring for assessment and planning, or they can be used to quickly construct an infrastructure such as a data network.

Most of the aforementioned UAV missions require multiple UAVs in cooperation due to the large scale of the mission. While some missions such as a wide area surveillance absolutely require multiple UAVs in simultaneous operation, some others greatly benefit from presence of a heterogeneous set of multiple UAVs.

For numerous reasons, including, but not limited to, the examples above, cooperative control of multiple UAVs has received significant attention over the last decade (see for example Refs. 1–12). Algorithms to handle combinatorially complex scenarios, uncertain environments, adversarial actions, and human operator interactions have been proposed, and experimental demonstrations of some of these algorithms have been successfully realized.[5,13] More recently, books have been published, which are devoted to or related to cooperative control,[7,9] leading to a large body of literature.

The focal part of most of the related research has been the exploitation of the complex combinatorics of the problem. In this regard, recently proposed task assignment algorithms have considered ordering constraints on tasks, tasks with time windows, or tasks that are coupled in time with relative timing constraints.[1,4,6] Moreover, formal languages have been employed to consider fairly complex scenarios.[14,15] All of these approaches, however, design and implement an "open-loop" control, which does not take into account the interaction with neither the environment nor the adversaries, but rather replan in such cases.

Yet, on the other hand, most military UAV operations take place in uncertain, dynamic, and adversarial environments, ignoring which may result in significant loss of performance and may lead to catastrophic failures such as loss of valuable assets. Recently, modeling these effects has been a promising line of research resulting in algorithms that mainly employ game theory, queueing theory, or dynamic programming.[2,16] These recent approaches, however, tend to simplify the combinatorial counterpart of the problem by posing severe assumptions such as assuming identical targets and identical UAVs or unrealistically simplified vehicle dynamics.

Inspired by the recent research in verification of reactive systems (see for instance[17]), we will categorize the type of cooperative UAV mission planning systems into three: static (open-loop) systems, interactive systems, and reactive systems. In the first category of approaches, the mission planning software generally produces an open-loop plan that satisfies the objective of the mission, however the environment is assumed to

*Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA.
†ARC Lab, the University of Michigan, Ann Arbor, MI.

be completely static. In the second category, there is inherent interaction with the environment, adversaries, or operators, but the interaction occurs in intervals that can be decided by the mission planning software. Hence, the mission planning algorithms of this category have to generate plans for different responses after each interaction. In the final category, on the other hand, the environmental changes, or adversarial actions occur in largely unpredictable time intervals. In this case, the mission planning algorithms have to be able to react to these changes or actions at any time.

A large body of literature exists regarding several static mission planning systems.[1,4,6] Similarly, recent research has explored interactive and reactive systems, though limiting the attention to combinatorially simple cases as discussed before. In this paper, we consider interactive planning problems of larger combinatorial complexity, which constitute a step towards specification and planning of complex reactive systems.

The paper utilizes process algebra for formal specification of the coupling between the tasks and provides an anytime state-space search algorithm with interesting algorithmic properties. In particular, the algorithm effectively searches and locates a feasible solution quickly and it returns an optimal solution in finite time.

Our work in this paper builds on our previous work.[2,3,14,15,18,19] In Refs. 3, 14, 15, formal languages such as Linear Temporal Logic (LTL) and Metric Temporal Logic (MTL) were employed to specify complex combinatorial constraints of military UAV operations. Mixed Integer Linear Programming (MILP) based solution methods were proposed for optimally scheduling UAVs to accomplish those tasks. Though these algorithms were shown to terminate in reasonable amount of time for quite practical cases, their applicability had been limited in large-scale problem instances. Mainly to tackle computational complexity, in Ref. 19, process algebras were employed for similar specification purposes and a tree search based algorithm was proposed, which quickly yields a sub-optimal solution and improves it given time—that is, it is guaranteed to terminate with the optimal solution in finite time. The computational efficiency of the algorithm stems from both the utilization of process algebras instead of temporal logic as well as the use of the tree search instead of MILPs. Process algebras, if not as expressive, are known to be computationally more efficient when compared to temporal logics. Moreover, standard tree-search algorithms (see for instance Ref. 4) can be used to implement anytime algorithms, which feature the aforementioned solution improvement and termination properties. However, none of these references considered either the uncertainties in the environment, or the actions of the adversaries, but rather proposed an open-loop control and choose to "re-plan" under such circumstances, like many others.[1,6] On the other hand, in Ref. 2, adversarial UAV operations and a computationally effective algorithm were presented for a SEAD-inspired, though combinatorially simple, mission. This algorithm was further improved to incorporate MTL type constraints in[18] resulting in a heuristic solution method. In this paper, we continue all the above work with proposing exact algorithms as well as computationally effective heuristics for interactive UAV missions.

The paper is organized as follows. Some preliminary definitions are presented in Section II. Section III is devoted to the introduction of process algebra. In Section IV the problem definition is outlined and in Section V an effective algorithm to solve this problem is presented. In Section VI, simulation results are provided for an illustrative example and the paper is concluded with remarks in Section VII.

II. Preliminary Definitions

In this section we provide some preliminary mathematical definitions and introduce our notation. In particular, we define sequences and labeled trees, both of which will be heavily used throughout the paper.

Let us denote the set of natural numbers by $\mathbb{N}$ and the set of non-negative real numbers by $\mathbb{R}_+$. Given a set A, a sequence σ of elements from A is a partial function, which maps the set $\{1, 2, \ldots, k\} \subset \mathbb{N}$ to A. The set of all sequences defined on a set A is denoted by Σ_A.

A directed graph $G = (N, E)$ consists of a set N of nodes and a set $E \subseteq N \times N$ of edges. An edge e is said to be an incoming (or outgoing) edge of a node n if $e = (n', n)$ (or $e = (n, n')$) for some $n' \in N$. Two edges $e, e' \in E$ are said to be adjacent if $e = (n, n')$ and $e' = (n', n'')$ for some $n, n', n'' \in N$. A directed path on G is a sequence $p = (e_1, e_2, \ldots, e_l)$ of edges such that $e_i \in E$ for all $i \in \{1, 2, \ldots, l\}$ and e_i and e_{i+1} are adjacent, for all $i \in \{1, 2, \ldots, l-1\}$. The directed graph G is said to include a path p, if p is a path on G. The path $p = (e_1, e_2, \ldots, e_l)$ is said to start at node n_s and end at node n_s if $e_1 = (n_s, n')$ and $e_l = (n'', n_e)$ for some $n', n'' \in N$. A directed path is called a cycle if it starts and ends at the same node.

A directed tree $T = (N, E)$ is a graph that does not include any cycles. It can be shown that every tree has a node, called the root node, that has no incoming edge. Given a directed tree T, the unique root node of T will be denoted by $\texttt{Root}(T)$. It can also be shown that all other nodes in the tree have

exactly one incoming edge each. Let n be a node in directed tree other than the root. Let $e = (n', n)$ be the unique incoming edge of n. The node n' is called the parent of n and will be denoted by $\texttt{Parent}_T(n)$. Let $e_1 = (n, n_1), e_2 = (n, n_2), \ldots, e_l = (n_l, n)$ be all the outgoing edges of n. Then, the set $\{n_1, n_2, \ldots, n_l\}$, which will be denoted as $\texttt{Children}_T(n)$, is the set of children of n. If a node n has no outgoing edge, then it is called a leaf node. The function $\texttt{Leaf}_T(n)$ is used to decide this case; it maps n to True if n is a leaf node and to False otherwise.

A labeled directed tree is a tree $T = (N, E)$, in which each node in N is associated with a label from an alphabet Σ_N and each edge is associated with a label from another alphabet Σ_E. Given a node n (or an edge e), its label will be denoted by $\texttt{Label}_T(n)$ (or $\texttt{Label}_T(e)$).

We will often refer to directed graphs, directed trees, and labeled directed trees as graphs, trees, and labeled trees, respectively, for short.

III. Process Algebra

Process Algebra has been used in the Computer Science literature to specify and reason about computer software.[20,21] Recently, it has been employed in specification and planning of UAV missions.[19] In Ref. 19, though less expressive than many other formalisms, process algebra specifications were shown to adequately describe a broad class of UAV missions, and yet provide computational tractability. In this section, we present the essentials of process algebra. For further details, the reader is referred to Refs 19, 21.

The syntax of process algebra is defined by its set of terms. Let A be a set of *actions*. The set of *process algebra terms* is formally defined as follows:

Definition III.1 *Given a set $\mathcal{A}$ of actions, the set $\mathbb{T}_{\mathcal{A}}$ of* process algebra terms *defined on $\mathcal{A}$ is the smallest set which satisfies the following:*

- *if $a \in \mathcal{A}$, then $a \in \mathbb{T}_{\mathcal{A}}$,*
- *if $p, p' \in \mathbb{T}_{\mathcal{A}}$, then $p + p', p \cdot p', p \parallel p' \in \mathbb{T}_{\mathcal{A}}$.*

When $\mathcal{A}$ is clear from the context, we will drop the subscript and write $\mathbb{T}$ instead of $\mathbb{T}_{\mathcal{A}}$.

Each term in $\mathbb{T}$ represents a valid specification and represents a set of desired behavior, i.e., a sequence of actions. For instance, an action $a \in \mathcal{A}$ is itself a term and represents the set of behavior with the single element (a) in it. More complicated examples are constructed using the operators of process algebra. If p and p' are two terms, each of which have their own associated set of behavior, then $p + p'$ is another term which represent behavior that either executes a behavior of p and that of p'. The term $p \cdot p'$, on the other hand, represents behavior that first executes a behavior of p followed by a behavior of p'. Finally, $p \parallel p'$ represents those behavior that execute a behavior of p and a behavior of p' in an interleaving manner. This informal definition can be formalized through the *process graphs* and *operational semantics*. Formally, a process graph is defined as follows:

Definition III.2 *The process graph of a term $p \in \mathbb{T}_{\mathcal{T}}$ is defined as a* labeled transition system $\mathcal{TS} = (Q, \mathcal{T}, \Rightarrow)$*, together with an initial state $q_0 \in Q$, and a labeling function $\pi : Q \to \mathbb{T}$ such that*

- *Q is a set of labeled transition system states,*
- *$\mathcal{T}$ is the set of targets indicating the set of actions,*
- *$\Rightarrow \subseteq Q \times \mathcal{T} \times Q$ is a tenary state transition relation, and*
- *π is a function labeling each state in Q with a term in $\mathbb{T}$ such that $\pi(q_0) = p$.*

Denoted by $q \overset{a}{\Rightarrow} q'$, a transition from a state q to q' is said to exists if and only if $(q_1, t, q_2) \in \Rightarrow$. Moreover, if there is a transition $q \overset{a}{\Rightarrow} q'$ such that $p = \pi(q)$ and $p' = \pi(q')$, then the term p is said to evolve to p' after executing a, denoted by $p \xrightarrow{a} p'$. More generally, if there is a set of actions containing $a_1, a_2, \ldots, a_k \in \mathcal{A}$ and a set of processes $p_0, p_1, \ldots, p_k$ such that $p_{i-1} \xrightarrow{a_i} p_i$ for all $i \in \{0, 1, \ldots, k\}$, then process p_0 is said to evolve to process p_k after executing the sequence $(a_1, a_2, \ldots, a_k)$ of actions. This relationship is denoted by $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} p_k$ or shortly by $p_0 \xrightarrow{\sigma} p_k$ where $\sigma = (a_1, a_2, \ldots, a_k)$.

The operational semantics is defined in terms of the notion of transition rules, which is formalized as follows:

Definition III.3 *A transition rule ρ is an expression $\frac{H}{\nu}$, where H is a set of transitions called the* positive premises *and ν is a transition called the conclusion.*

One straightforward way to represent a set of labeled transition systems (such as process graphs) is to provide the states and the labeling function explicitly for each labeled transition system in the set. Another, generally more compact, way is to use transition rules. Intuitively, given a transition rule $\rho = \frac{H}{\nu}$, if the set H of premisses are possible transitions of some process p, then so is the conclusion ν. A set of transition rules is referred to as a *transition system specification.* The transition system specification of the process algebra is given as follows:

Definition III.4 (Operational Semantics of Process Algebra) *The operational semantics of the process algebra is given by the following set of transition system specifications:*

$$\frac{}{a \xrightarrow{a} \surd}$$

$$\frac{p_1 \xrightarrow{a} \surd}{p_1 + p_2 \xrightarrow{a} \surd} \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 + p_2 \xrightarrow{a} p_1'} \quad \frac{p_2 \xrightarrow{a} \surd}{p_1 + p_2 \xrightarrow{a} \surd} \quad \frac{p_2 \xrightarrow{a} p_2'}{p_1 + p_2 \xrightarrow{a} p_2'}$$

$$\frac{p_1 \xrightarrow{a} \surd}{p_1 \cdot p_2 \xrightarrow{a} p_2} \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 \cdot p_2 \xrightarrow{a} p_1' \cdot p_2}$$

$$\frac{p_1 \xrightarrow{a} \surd}{p_1 \| p_2 \xrightarrow{a} p_2} \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 \| p_2 \xrightarrow{a} p_1' \| p_2} \quad \frac{p_2 \xrightarrow{a} \surd}{p_1 \| p_2 \xrightarrow{a} p_1} \quad \frac{p_2 \xrightarrow{a} p_2'}{p_1 \| p_2 \xrightarrow{a} p_1 \| p_2'}$$

where $a \in A$ and $p_1, p_1', p_2, p_2' \in \mathbb{T}$.

The symbol $\surd$ represents a process which has terminated, i.e., can not execute any further actions. In essence, the first rule states that $a \in \mathcal{A}$ can execute a and then it must terminate. The transition rules in the second line define the alternative composition indicating, for instance, that if a process p_1 can execute a and terminate, then so can $p_1' + p_2$, or, similarly, if a process p_1 can execute a and evolve to term p_1', then so can $p_1 + p_2$, for an arbitrary term $p_2 \in \mathbb{T}$. The third and fourth lines define the sequential and the parallel composition in a similar fashion.

The following definition formally presents the notion of a trace.

Definition III.5 *Given a term $p \in \mathbb{T}$, a sequence σ defined on $\mathcal{A}$ is said to be a* trace *of p if and only if $p \xrightarrow{\sigma} \surd$ holds. The set of all traces of a term p is denoted by Γ_p.*

IV. Specification of Interactive UAV Missions

This section is devoted to the presentation of a specification methodology, which employs process algebra to specify UAV missions with interactive tasks. In particular, we first refine the definition of atomic objectives, which was first presented in[19] inspired by the generic tasks in.[22] Afterward, we provide the necessary definitions to formalize the problem definition.

A. Interactive Atomic Objectives

Atomic objectives, as defined in ,[19] are individual tasks that can not be executed as a series of other tasks. In this sense, atomic objectives are indivisible operations in UAV missions. In this section, we extend the definition of atomic objectives to *interactive atomic objectives* as follows:

Definition IV.1 ((Interactive) Atomic Objective) *An atomic objective o is a tuple $(U_o, x_o^i, x_o^f, T_o, \mathcal{R}_o)$, where*

- *$U_o \in \mathcal{U}$ is a UAV that is capable of executing the atomic objective*
- *x_o^f is a coordinate that U_o has to get to before starting the execution*
- *x_o^f is a coordinate that U_o ends up after executing the atomic objective*

- $T_o \in \mathbb{R}_+$ *is the amount of time it takes to execute this atomic objective*
- $\mathcal{R}_o$ *is a set of responses*

Intuitively, an atomic objective $o = (U_o, x_o^i, x_i^f, T_o, \mathcal{R}_o)$ is a task to be performed by UAV U_o after moving to coordinate x_o^i. It takes exactly T_o amount of time for U_o to execute this task and after the execution U_o ends up at coordinate x_o^f. If the set $\mathcal{R}_o$ of responses is non-empty, then right after executing the task the decision maker observes a non-deterministically chosen element from the set $\mathcal{R}_o$; however, the realized response is not known a priori. In essence, $\mathcal{R}_o$ denotes the set of all possible responses from the environment, the adversary, or the operator, after executing o. Note that if the decision maker decides not to execute o, then none of the responses in $\mathcal{R}_o$ is realized.

If $\mathcal{R}_o$ is empty, then no response is observed after the execution. This reduces the interactive atomic objective to the definition of (non-interactive) atomic objective as defined in.[19] Many realistic tasks can be modeled as an atomic objective. As noted in[19] and,[4] some of these tasks include area and sector search, classification, attack, and verification of destruction tasks, object tracking etc. Together with the interactivity component, it is possible to define for instance those tasks that require interacting with the human operators, environment, or enemy. An interesting type of interactive atomic objective is the classification task, in which a human operator classifies the target as no-target or a known target type looking at video footage from an onboard camera. In this the task involves the UAV traveling to the target location and transmitting the video footage back to the base. The response, on the other hand, is the target type as reported by the operators.

After we define the notion of specification, we will associate each response with a specification, indicating the desired series of actions that the decision maker has to take once one of the responses is realized.

From here on we fix a set $\mathcal{O}$ of atomic objectives. When necessary, we will denote the sets static and interactive atomic objectives with $\mathcal{O}_S$ and $\mathcal{O}_I$, respectively. Note that $\mathcal{O}_S \cup \mathcal{O}_I = \mathcal{O}$ and $\mathcal{O}_S \cap \mathcal{O}_I = \emptyset$.

B. Specification, Strategy, and Observations

Informally speaking, a specification is a process algebra term p_{spec} defined on the set $\mathcal{O}$ of atomic objectives. In essence, p_{spec} encodes the high-level objective of the mission at hand. For each response $r \in \mathcal{R}_o$ of each interactive atomic objective $o \in \mathcal{O}_I$ that is in p_{spec}, there is an associated process algebra term p_o^r, which indicates the desired set of behavior when for the response r after o is executed (if it is executed by the decision maker at all). Note that p_o^r may also contain interactive atomic objectives, all responses of which are associated with other process algebra terms.

Definition IV.2 (Specification) *A* specification *is a process algebra term defined on the set $\mathcal{O}$ of atomic objectives such that*

- *for any atomic objective o with a non-empty set of responses, each response $r \in \mathcal{R}_o$ is associated with another process algebra term p_o^r,*
- *any atomic objective $o \in \mathcal{O}$ appears at most once in p_{spec} or p_o^r for any $o \in \mathcal{O}_I$ and $r \in \mathcal{R}_o$.*

For reasons to become clear in later in this section, this eliminates the risk of infinite executions. Note that otherwise the second condition is without loss of any generality, since one can use copies of atomic objectives represented with a different symbol. An *assignment* $A = (o, t)$ is an atomic objective and time pair, i.e., $A \in \mathcal{O} \times \mathbb{R}_+$. Informally speaking, the decision maker decides a set of assignments so as to satisfy the specification p_{spec}. If an atomic objective $o \in \mathcal{O}$ is scheduled to occur at time t, then it is required to have a set of assignments, which occur after o and satisfies the p_o^r for all responses $r \in \mathcal{R}_o$. Formalizing these set of assignments, we define a strategy (or a policy) as follows:

Definition IV.3 (Strategy) *A strategy $\mathcal{S}$ is a connected labeled tree such that each node is associated with an assignment $(o, t) \in \mathcal{O} \times \mathbb{R}_+$, each edge is labeled with a symbol from $\{\rho\} \cup \{r \in \mathcal{R}_o \,|\, o \in \mathcal{O}_I\}$, and the following conditions hold:*

- *each node that is labeled with an assignment $(o, t) \in \mathcal{O}_S \times \mathbb{R}_+$ has one outgoing edge labeled with ρ,*
- *each node that is labeled with an assignment $(o, t) \in \mathcal{O}_S \times \mathbb{R}_+$ has exactly $|\mathcal{R}_o|$ number of outgoing edges, each of which is labeled with a distinct element of $\mathcal{R}_o$,*

Hence, a strategy encodes a plan for every response of all scheduled interactive atomic objectives. More formally, following a path on the tree, one observes a plan for a specific sequence of responses that occur after taking interactive actions. A schedule is such that there is exactly one path for each such sequence of responses. These paths on a schedule are important enough to be a definition on their own right.

Definition IV.4 (Outcome) *An* outcome *of a strategy $\mathcal{S}$ is a sequence $\theta = (\,(o_1,t_1),(o_2,t_2),\ldots,(o_n,t_n)\,)$ of assignments such that there exists a path on $\mathcal{S}$ starting from the root node and ending at a leaf node and the corresponding node labels of this path is θ. The set of all outcomes of a strategy $\mathcal{S}$ is denoted by $\Theta_{\mathcal{S}}$.*

For every outcome, each UAV is scheduled to execute a set of atomic objectives of its own. For notational simplicity, we will denote the sequence of assignments executed by a UAV $u \in \mathcal{U}$ in an outcome θ as $[\theta]_u$ in their execution order. More formally, $[\theta]_u$ is the order preserving projection of the sequence θ onto the set $\{o \in \mathcal{O} \,|\, U_o = u\} \times \mathbb{R}_+$.

Definition IV.5 (Valid Outcome) *An outcome is said to be* valid *if for all UAVs $u \in \mathcal{U}$, the sequence of atomic objectives $[\theta]_u = (\,(o_1,t_1),(o_2,t_2),\ldots,(o_n,t_n)\,)$ executed by u satisfies the following:*

- *for all $i \in \{1,2,\ldots,n-1\}$, there holds $t_i + T^t_{o_i} + T^t_{o_i,o_{i+1}} \leq t_{i+1}$.*

Hence, an outcome is said to be valid only if each UAV can timely execute all of the atomic objectives it is assigned. The property of validity can be naturally extended to schedules as follows:

Definition IV.6 (Valid Schedule) *A schedule $\mathcal{S}$ is said to be valid if every outcome $\theta \in \Theta_{\mathcal{S}}$ of $\mathcal{S}$ is valid.*

Up to now, we presented the necessary definitions that formalizes an implementable strategy. In this respect, a valid strategy is the one that can be executed by the UAVs within the time constraints. Next, we present a formalism to distinguish those schedules that satisfy a given specification from others. For this purpose, we first associate each outcome with a set of its observations as in.[19]

Definition IV.7 (Observation) *An observation of an outcome $\theta = (\,(o_1,t_1),(o_2,t_2),\ldots,(o_n,t_n)\,)$ is a sequence $\pi = (o_1,o_2,\ldots,o_n)$ of atomic objectives such that*

- *o_i appears in π if and only if there exists some $t_i \in \mathbb{R}_+$ such that $(o_i,t_i) \in \theta$ holds,*
- *there exists $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_n$ such that (i) for all $(o_i,t_i) \in \theta$, $t_i \leq \tau_i \leq t_i + T_o$ and (ii) for all $(o_i,t_i),(o_j,t_j) \in \mathcal{S}$, $o_i \prec_\pi o_j$ implies $\tau_i \leq \tau_j$.*

The set of all observations of an outcome θ is denoted as Π_θ.

Given a strategy $\mathcal{S}$ and $\theta \in \Theta_{\mathcal{S}}$ be an outcome of $\mathcal{S}$. Recall that θ is induced by a path on the tree $\mathcal{S}$. Let $R_\theta = (r_1,r_2,\ldots,r_k)$ be the corresponding sequence of edge labels to this path when projected on the set of all responses, i.e., $\cup_{o\in\mathcal{O}_I}\mathcal{R}_o$. In essence, $(r_1,r_2,\ldots,r_n)$ contains all the elements in the corresponding sequence of edge labels, but the symbol ρ, which is eliminated while preserving the order of the sequence. Moreover, let $(p_1,p_2,\ldots,p_n)$ be the sequence of process algebra terms, in which p_i is PA term associated with the response r_i for all $i \in \{1,2,\ldots,n\}$. Finally, let p^θ_{spec} be the process algebra term obtained as follows:

$$p^\theta_{spec} = \Big[\cdots\Big[[p_{spec}]^{o_1\cdot p_1}_{o_1}\Big]^{o_2\cdot p_2}_{o_2}\cdots\Big]^{o_n\cdot p_n}_{o_n}$$

Then, a schedule can be distinguished as a satisfying schedule as follows.

Definition IV.8 (Satisfaction of Specifications) *A schedule $\mathcal{S}$ is said to* satisfy *a given specification p_{spec} if for all outcomes θ of $\mathcal{S}$ we have that all observations of θ is a trace of p^θ_{spec}, i.e., for all $\theta \in \Theta_{\mathcal{S}}$, there holds $\Pi_\theta \subset \Gamma_{p^\theta_{spec}}$.*

C. Problem Definition

Every strategy $\mathcal{S}$ can naturally be associated with a cost function in several ways. To define a cost function on the set of all strategies, let us first discuss ways to associate cost to an outcome. Let θ be an outcome of $\mathcal{S}$ and let τ_u^c, defined for all $u \in \mathcal{U}$, be the *completion time* of UAV u in this outcome, in other words, the time that u completes all its assignments. Then, two different natural cost functions associated to θ are

$$J_1(\theta) = \max_{u \in \mathcal{U}} \tau_u^c, \tag{1}$$

$$J_2(\theta) = \sum_{u \in \mathcal{U}} \tau_u^c. \tag{2}$$

The former cost function, also referred to as the mission time, indicates the time it takes to complete the outcome, whereas the second cost function indicates a type of risk since it increases as the more UAVs are involved in the mission for longer periods of time.

The cost of a strategy is determined as a function of the cost of its outcomes. A natural cost function is

$$c(\mathcal{S}) = \max_{\theta \in \Theta_{\mathcal{S}}} J_i(\theta),$$

where $i \in \{1, 2\}$ refers to one of the two different cost functions. This particular cost yields a game theoretic problem formulation, where the decision maker tries to minimize the maximum cost that can occur during the course of the mission. Yet, it assumes that the decision maker has no a priori knowledge of the outcome that is realized. Instead, if, for each interactive atomic objective $o \in \mathcal{O}_I$, a probability distribution $\mathbb{P}_o$ over the of set $\mathcal{R}_o$ of responses is known, then one can minimize the expected value of $J_i(\theta)$, where the expectation is taken over the set of all of outcomes of a given schedule. More formally, given an outcome θ recall that $R_\theta = (r_1, r_2, \ldots, r_k)$ was the projection of the sequence of edge labels corresponding to the path that induces θ on the set of all responses. Let $P(r_i)$ be the probability that r_i is realized, then the expected value is

$$c(\mathcal{S}) = \sum_{\theta \in \Theta_{\mathcal{S}}} \prod_{r \in R_\theta} P(r) J_i(\theta),$$

where $i \in \{1, 2\}$ again induces two different cost functions.

Given the definitions provided in this and the previous sections, the problem definition can be formalized as follows:

Problem IV.9 *Given a set $\mathcal{O}$ of atomic objectives, and a specification p_{spec} defined on $\mathcal{O}$, the problem is to find a valid strategy $\mathcal{S}$ such that (i) $\mathcal{S}$ satisfies p_{spec}, (ii) the cost function $c(\mathcal{S})$ is minimized.*

In the next section we propose a state-space search algorithm, which extends the one in,[4,19] and show that this algorithm quickly returns a feasible strategy and improves it over time.

V. State-space Search Algorithm

In this section, we provide a state-space search algorithm, which solves Problem IV.9.

A. Assignment Tree

The algorithm that we describe in this section encodes all possible assignments into a labeled tree structure $(\mathcal{N}, \mathcal{E})$, called the *assignment tree*, in which each node is labeled with a process algebra term and each edge is labeled with a atomic objectives and response pair. Moreover, each leaf node of the tree encodes an outcome of a set of strategies.

Definition V.1 (Assignment Tree) *Assignment tree of a process algebra term p_{spec} is the smallest tree $(\mathcal{N}_{p_{spec}}, \mathcal{E}_{p_{spec}})$ satisfying the following properties:*

- *The root node $n_r \in \mathcal{N}_{p_{spec}}$ is labeled with p_{spec},*
- *If a node $n \in \mathcal{N}$ is labeled with a process algebra term p and if there exists a process algebra term p' and a static atomic objective $o' \in \mathcal{O}_S$ such that $p \xrightarrow{o'} p'$ holds then there exists another node $n' \in \mathcal{N}$ that is labeled with p' and an edge $e = (n, n')$ that is labeled with (o', ρ),*

- *If there exists a node $n \in \mathcal{N}$ that is labeled with a process algebra term p and there is an interactive atomic objective $o \in \mathcal{O}_I$ such that $p \xrightarrow{o'} p'$, then for each response $r \in \mathcal{R}_o$, there exists a node $n' \in \mathcal{N}$ that is labeled with $[p']_o^{p_o^r}$ and an edge $e = (n, n')$ that is labeled with where (o, r), where p_o^r is the specification associated with the response r of atomic objective o.*

Notice that each leaf node of the assignment tree is labeled with the terminated process $\surd$. Hence, for any specification p_{spec}, its corresponding assignment tree p_{spec} is a finite tree.

Most of the stats-space search methods on trees explore the nodes of the tree by recursively evaluating the children nodes at each node. Hence, while performing a search over the nodes of the assignment tree, it is crucial to efficiently determine the children nodes of a given node in the tree. Let $n \in \mathcal{N}_{p_{spec}}$ be a node, then all its children, that is the labels of the children nodes and the labels of the connecting edges can be determined efficiently. Given a process algebra term p, Algorithm 1 runs in time polynomial with respect to the size of p and returns the set of all (p', o', r') such that there is a child node labeled with p' and a connecting edge labeled with (o', r').

Algorithm 1: Computation of $Next(p)$

switch p **do**
  case $p_1 + p_2$
    return $Next(p_1) \cup Next(p_2)$
  case $p_1 \cdot p_2$
    return $\{(p_1' \cdot p_2, o', \rho) \,|\, (p_1', o', \rho) \in Next(p_1)\} \cup \{([p_1']_o^{p_{o'}^{r'}} \cdot p_2, o', r') \,|\, (p_1', o', r') \in Next(p_1)\}$
  case $p_1 \,\|\, p_2$
    return $\{(p_1' \,\|\, p_2, o', \rho) \,|\, (p_1', o', \rho) \in Next(p_1)\} \cup \{([p_1']_{o'}^{p_{o'}^{r'}} \,\|\, p_2, o', r') \,|\, (p_1', o', r') \in Next(p_1)\}$
    $\cup \{(p_1 \,\|\, p_2', o', \rho) \,|\, (p_2', o', \rho) \in Next(p_2)\} \cup \{(p_1 \,\|\, [p_2']_{o'}^{p_{o'}^{r'}}, o', r') \,|\, (p_2', o', r') \in Next(p_2)\}$
  otherwise
    if $p \in \mathcal{O}_s$ **then**
      return $\{(\surd, p, \rho)\}$
    end
    if a **then**
      return $\{(\surd, p, r) \,|\, r \in \mathcal{R}_p\}$
    end
  end
end

B. Computation of the Assignments

Each node n of the assignment tree is naturally associated with an assignment $(o, t) \in \mathcal{O} \times \mathbb{R}_+$, where (o, r) is the label of the edge $e = (n', n)$, $r \in \mathcal{R}_o$, and n' is the parent node of n in the assignment tree. The execution time of the assignment is such that every sub-tree with a specific structure naturally yields a valid strategy. In the rest of this section, we first provide an algorithm that computes an assignment for any given node in the assignment tree, and then formalize the aforementioned specific structure of the sub-trees that yield a valid strategy.

The algorithms to be presented in this section heavily employ a definition referred to as the precedence relation between the atomic objectives.[19] Formally, an atomic objective o' is said to precede an atomic objective o in a specification p_{spec}, if (i) there exists a trace γ of p_{spec} such that $o' \prec_\gamma o$ and (ii) there exists no trace of p_{spec} such that o precedes o'. This condition can be shown to be equivalent to another condition which can be checked in polynomial time.[19] More precisely, o' precedes o in p_{spec} if and only if there exists a node n' in the parse tree of p_{spec} labeled with the sequential combination operator $(\cdot)$ and the left and right children ofn' contain the atomic objectives o' and o, respectively, in their set of children atomic objectives. For further details and the proof the reader is referred to.[19] Given a process algebra term p_{spec} and an atomic objective o that appears in p_{spec}, we will denote the set of all precedences of o by $\texttt{Pred}_{p_{spec}}(o)$.

Let p_{spec} be a process algebra specification. The sequence σ_n of assignments corresponding to a node n of $(\mathcal{N}_{p_{spec}}, \mathcal{E}_{p_{spec}})$ is computed recursively as follows. The sequence $\sigma_{n_{root}}$ of assignments is the empty sequence

for the root node. Let n' be the parent of node n and let $\sigma_{n'} = ((o_1, t_1), (o_2, t_2), \ldots, (o_k, t_k))$ be the corresponding sequence of assignments. Then, the sequence σ_n of assignments is the sequence $\sigma_{n'}|(o', t')$, where (o', r) is the label of the edge (n', n) for some $r \in \{\rho\} \cup \mathcal{R}_{o'}$. The execution time t' of the atomic objective $o' = (U_{o'}, x^i_{o'}, x^f_{o'}, T^e_{o'}, \mathcal{R}_{o'})$ is computed with respect to the assignment times of the atomic objectives o_i, $i \in \{1, 2, \ldots, k\}$. With this regard, let $\tau^c_{o_i}$ be the completion time of the execution of o_i, i.e., $\tau^c_{o_i} = t_i + T^e_{o_i}$. Then, in words, t' is the maximum of the maximum completion time of all its predecessors that are assigned in the sequence $\sigma_{n'}$ and the completion time of $U_{o'}$. More formally,

$$t_i = \max\big\{\, \max\{\tau^c_{o_i} \,|\, o_i \in i \in \{1, 2, \ldots, k\} \text{ and } \texttt{Pred}_{p_{spec}}(o')\},\\ \max\{\tau^c_{o_i} \,|\, U_{o'} \text{ is the capable UAV to execute } o_i\}\big\}.$$

Then the (single) assignment associated with node n is (o', t'), which is computed efficiently from the sequence of all assignments corresponding to nodes on the unique path from the root node n_{root} to n.

Next, we characterize those sub-trees of $(\mathcal{N}_{p_{spec}}, \mathcal{E}_{p_{spec}})$ that naturally yield a valid strategy that satisfies p_{spec}. Let (N_S, E_S) be such a sub-tree and let A_n be the assignment corresponding to node $n \in N_S \subseteq \mathcal{N}_{p_{spec}}$ as computed earlier in this section. Then, (N_S, E_S) satisfies the following properties:

- The root node n_{root} of the assignment tree is in N_S;
- If a node n is in N_S and n is not a leaf node of the assignment tree (or equivalently n is not labeled with $\surd$ in the assignment tree), then, for exactly one atomic objective

  $$o' \in \{o \in \mathcal{O} \,|\, \text{there exists } p' \text{ such that } p \xrightarrow{o} p'\},$$

  all the edges (n, n') with label (o', r') either for $r' = \rho$ or for all $r' \in \mathcal{R}_{o'}$ are in E_S and the nodes n' are in N_S;
- A node $n \in N_S$ is labeled with the assignment $A_{n'}$, where n' is one of the children of n in (N_S, E_S);
- An edge $e = (n, n') \in E_S$ is labeled with r, where r is such that (o, r) is the label of the edge $(n, n') \in \mathcal{E}_{p_{spec}}$.

Notice that (N_S, E_S) is indeed a sub-tree of $(\mathcal{N}_{p_{spec}}, \mathcal{E}_{p_{spec}})$. It is also easy to show that (N_S, E_S) is a valid schedule, considering the computation of the assignments A_n defined for all $n \in \mathcal{N}_{p_{spec}}$.

C. State-space Exploration Algorithm

In,[19] a best-first branch-and-bound state space search algorithm was presented to evaluate the accumulated cost of each node in an assignment tree. The best-first nature of the algorithm provided the guarantee of locating a feasible solution quickly, by exploring the nodes in the tree in a depth first manner. In,[19] since interactive atomic objectives did not exist, each leaf node of the tree encoded a feasible solution to the problem and showing that the maximum depth of the assignment tree was polynomial with respect to the size of the specification proved this fact.

However, in the presence of interactive atomic objectives, one needs to locate a strategy, which amounts to find a set of leaf nodes, in general, with more than one element. In particular, for each interactive atomic objective $o \in \mathcal{O}_I$ that is in the strategy one needs to find at least $|\mathcal{O}_I|$ number of leaf nodes. Hence, a naïve best-first search strategy may take large amount of time to find a feasible strategy, while evaluating each children node in the tree one by one.

Instead, in this paper, we propose an algorithm, which combines depth-first and breadth-first search methods to effectively search the tree and locate a feasible solution quickly. The search algorithm assumes two global data structures: $\texttt{Evaluated}(n)$ and $\texttt{CurrentAO}(n)$, both defined for all $n \in \mathcal{N}_{p_{spec}}$. The variable $\texttt{Evaluated}(n)$ takes boolean values and is initially set to $False$ for each node $n \in \mathcal{N}_{p_{spec}}$. Let p_n denote the label of the node n. Then, the variable $\texttt{CurrentAO}(n)$ takes values from the set of atomic objectives that can p_n can execute in one step, i.e., $\{o \in \mathcal{O} \,|\, \text{there exists } p \text{ such that } p_n \xrightarrow{o} p\}$. Initially, $\texttt{CurrentAO}(n)$ is set to a null value that is not in $\mathcal{O}$, for all $n \in \mathcal{N}_{p_{spec}}$. Finally, the method $\texttt{Cost}(n)$ computes the assignment for node n as explained in Section V.B and evaluates its accumulated cost as defined in Equation (1) or (2).

The state-space search algorithm makes use of a method called `OneStepExpand`, which takes a node in the tree and computes the corresponding assignment of only a subset of its children nodes and, subsequently, returns the set of children it expanded. The `OneStepExpand` method is provided in Algorithm 2.

Algorithm 2: `OneStepExpand` method

if `Leaf`$(n) = True$ **then**
  `UpdateStrategyCost`(n)
  `Evaluated`$(n) = True$
  return `Parent` (n)
end
$S := \{n' \,|\, n' \in$ `Children`(n) and `Evaluated`$(n) = False\}$
if $S = \emptyset$ **then**
  `Evaluated`$(n) = True$
  return `Parent`(n)
else
  if *there exist* $n' \in S$ *such that* $($`CurrentAO`$(n), r) =$ `Label`$((n, n'))$ *for some* $r \in \mathcal{R}_{\texttt{CurrentAO}(n)}$ **then**
    return $\emptyset$
  else
    Let $n' = argmin\{Cost(\bar{n}) \,|\, \bar{n} \in S\}$
    Let $(o', r') = Label(n, n')$
    `CurrentAO`$(n) = o'$
    return $\{\bar{n} \in S \,|\,$ `Label`$((n, \bar{n})) = (\bar{o}, \bar{r})$ *for some* $\bar{r} \in \mathcal{R}_{\bar{o}} \cup \{\rho\}\}$
  end
end

Before discussing the operation of the `OneStepExpand` method, let us introduce the search algortihm. The state-space search algorithm is provided in Algorithm 3, which employs the *OneStepExpand* method to function. Algorithm 3 implements a forward state-space search method, which keeps track of multiple nodes on the tree that it is working on. This set of nodes is stored in the `NodeList` variable. In each iteration, in Lines 2-8, the algorithm runs the `OneStepExpand` method for each node in the `NodeList`, builds a set of new nodes returned by the `OneStepExpand` method and stores them in variable `NewNodeList`. Essentially, `NewNodeList` is the union of the sets of nodes that are returned by the `OneStepExpand` method during the single iteration of Algorithm 3. Before proceeding with the next iteration, `NodeList` is updated to be the `NewNodeList`. The iterations are continued until there is no new node to explore.

Clearly, the Algorithm 3 expands the nodes in the assignment tree one-by-one. At each `OneStepExpand` operation, the new set of nodes include the those children of the current one that are expanded. Noting that in a single `OneStepExpand` operation multiple nodes can be expanded (happens if the child nodes correspond to different responses of the same atomic objective), the `NodeList` may grow in size at the end of the iteration. Note also that if all the children nodes of the node n that `OneStepExpand`(n) method is called with are evaluated, then the `OneStepExpand` method returns the parent of n. If n is the root node, then it will return the empty set. Moreover, if n has some non-evaluated children which assign the atomic proposition that n is currently expanding, then `OneStepExpand`(n) will again return an empty set of nodes.

Algorithm 3: State-space search algorithm

`NodeList` $:= \{n_{root}\}$
while `NodeList` $\neq \emptyset$ **do**
  $NewNodeList := \emptyset$
  for $\forall n \in$ `NodeList` **do**
    $NewNodeList := NewNodeList \cup$ `OneStepExpand`(n)
  end
  $NodeList := NewNodeList$
end

Notice that the state space search algorithm explores the assignment tree in a best-first manner, with the only difference that when assigning an interactive atomic objective o incurs the minimum accumulated cost, the algorithm explores the tree by exploring one node at a time on each branch corresponding to different

outcomes of o. This procedure is executed recursively.

VI. Simulations

In this section, we present an illustrative example motivated by a military scenario. This scenario consists of 5 target locations that are to be identified by UAVs. First, a classification task has to be performed for each target independently. That is, a specific high flying UAV has to take a video footage and transmit it to the base. The human operators on the base have to identify the targets as (i) no target, (ii) target type A, or (iii) target type B. If the human operator decides that there is no target in the footage, then no further action should be taken. If, on the other hand, the target is type A or type B, it must first be destroyed then the destruction must be verified by taking another footage and transmitting it back to the base. If the target is of type A, then both destruction and damage verification tasks take 0.1 hours each. If it is of type B, on the other hand, the same tasks take 0.2 hours each.

There are three different UAVs in this mission. UAVs 1 and 2 can execute either attacking or verification of destruction tasks. They differ in that UAV 2 can travel with a top speed of 20 mph whereas UAV 1 can travel with 20 mph top speed. UAV 3, on the other hand, can only execute the classification tasks and it can travel with a maximum speed of 30 mph.

This mission can be modeled with the proposed specification method as follows. Let $o_{i,c}$ denote the atomic objective for classifying target $i \in \{1, 2, \ldots, 5\}$, which can only be executed by UAV 3. Note that $o_{i,c}$ is an interactive atomic objective, since it requires a response from the operator regarding the target type. Let us associate each $o_{i,c}$ with the set $\mathcal{R}_{o_{i,c}} = \{r_{i,nt}, r_{i,A}, r_{i,B}\}$ of responses, where the three elements indicate "no target," "target A," and "target B," respectively. For the response $r_{i,nt}$ we associate the process algebra term $\surd$. That is no further action needs to be taken. To define the term regarding $r_{i,A}$, let us define the two attack options $o_{i,A,a,1}$ and $o_{i,A,a,2}$, which are the task for attacking target i of type A with UAV 1 and UAV 2, respectively. Also, let us define the verification tasks as $o_{i,A,v,1}$ and $o_{i,A,v,2}$, which correspond to tasks for verifying the destruction on target i of type A with UAVs 1 and 2 respectively. Using these atomic objectives, the term associated with the response $r_{i,A}$ can be written as $(o_{i,A,a,1}+o_{i,A,a,2})\cdot(o_{i,A,v,1}+o_{i,A,v,2})$, which simply amounts to first attacking the target either with UAV 1 or with UAV 2, then verifying the destruction with one of the UAVs. The atomic objectives $o_{i,B,a,1}, o_{i,B,a,2}, o_{i,B,v,1}, o_{i,B,v,2}$ can be defined similarly so that the response $r_{i,B}$ is associated with the term $(o_{i,B,a,1} + o_{i,B,a,2}) \cdot (o_{i,B,v,1} + o_{i,B,v,2})$. To summarize, the specifications corresponding to different responses of the $o_{c,i}$ task can be written as follows:

$$o_{i,c} = \begin{cases} \surd & \text{if } r_{i,nt}, \\ (o_{i,A,a,1} + o_{i,A,a,2}) \cdot (o_{i,A,v,1} + o_{i,A,v,2}) & \text{if } r_{i,A}, \\ (o_{i,B,a,1} + o_{i,B,a,2}) \cdot (o_{i,B,v,1} + o_{i,B,v,2}) & \text{if } r_{i,B}. \end{cases} \tag{3}$$

The mission specification is then executing all these classification tasks (and tasks following to them) in parallel, which amounts to the process algebra term:

$$p_{spec} = o_{1,c} \parallel o_{2,c} \parallel o_{3,c} \parallel o_{4,c} \parallel o_{5,c}.$$

In this mission the duration of tasks $o_{i,c}$ is 0.1 hours, $o_{i,A,a,1}, o_{i,A,a,2}, o_{i,A,v,1}, o_{i,A,v,2}$ are 0.1 hours, and $o_{i,B,a,1}, o_{i,B,a,2}, o_{i,B,v,1}, o_{i,B,v,2}$ are 0.2 hours.

The algorithm was implemented in C++ and run on a laptop computer equipped with a 2.5GHz processor and 4GB RAM. The execution time of the algorithm versus the cost of the optimal strategy is shown in Figure 1. Note that the algorithm returns a feasible solution in about 0.1 seconds, which includes the initialization time of the algorithm.

VII. Conclusions

This paper presented a novel process algebra-based method for the specification of interactive tasks in UAV missions. The method was shown to model realistic complex interactive high-level mission specification in military scenarios. To solve the problem, a state-space search algorithm was presented. This algorithm returns a best-first feasible solution quickly and improves this solution given more time.

The future research directions will include improving the specification methodology to handle persistent missions, considering applications to more realistic surveillance and reconnaissance scenarios. In this case,

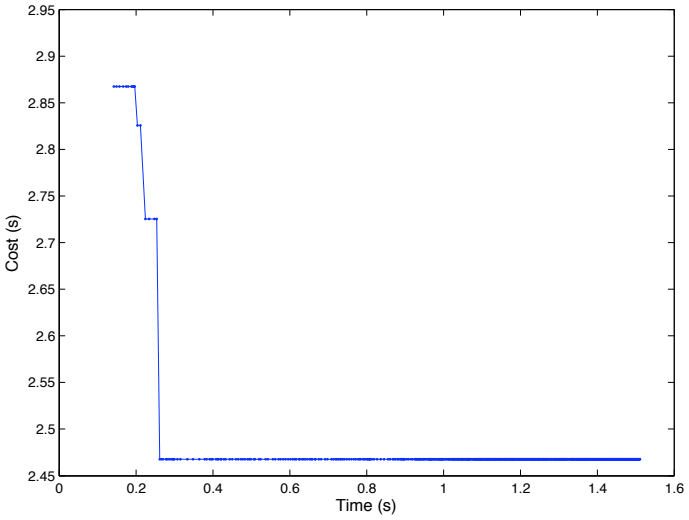

Figure 1. Cost of the strategy versus computation time.

the resulting strategy must be a closed-loop infinite horizon control. Another research direction is modeling of reactive UAV missions, in which interactions with the environment, adversary, or the operator may happen at times that are not known a priori to the decision maker.

Acknowledgments

The authors are grateful for the discussions with Steven Rasmussen and Derek Kingston of the Air Force Research Laboratory, Dayton, OH. This research was supported by the Michigan/AFRL Collaborative Center on Control Sciences, AFOSR grant no. FA 8650-07-2-3744. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the supporting organizations.

References

[1]Alighanbari, M., Kuwata, Y., and How, J., "Coordination and Control of Multiple UAVs with Timing Contraints and Loitering," *American Control Conference*, 2003.

[2]Faied, M. and and, A. G., "Adversarial UAV Operations," *AIAA Guidance, Navigation, and Control Conference*, 2008.

[3]Karaman, S. and Frazzoli, E., "Complex Mission Optimization for Multiple-UAVs using Linear Temporal Logic," *American Control Conference*, 2008.

[4]Rasmussen, S. and Shima, T., "Tree search algorithm for assigning cooperating UAVs to multiple tasks," *International Journal of Robust and Nonlinear Control*, Vol. 18, 2008, pp. 135–153.

[5]Richards, A., Kuwata, Y., and How, J., "Experimental Demonstration of Real-time MILP control," *AIAA Guidance, Navigation, and Control Conference*, 2003.

[6]Schumacher, C., Chandler, P., Pachter, M., and Patcher, L., "Optimization of air vehicles operations using mixed-integer linear programming," *Journal of the Operational Research Society*, Vol. 58, 2007, pp. 516–527.

[7]Shima, T. and Rasmussen, S., editors, *Cooperative Decision and Control: Challenges and Practical Approaches*, SIAM Control series, Philadelphia PA, 2008.

[8]Ren, W., Beard, R., and Atkins, E., "Information consensus in multivehicle cooperative control," *IEEE Control Systems Magazine*, Vol. 27, No. 2, 2007, pp. 71–82.

[9]Shamma, J., editor, *Cooperative Control of Distributed Multi-Agent Systems*, Wiley, 2008.

[10]Murray, R., "Recent Research in Cooperative Control of Multivehicle Systems," *Journal of Dynamical Systems, Measurement, and Control*, Vol. 129, No. 5, 2008, pp. 571–584.

[11]Olfati-Saber, R., Dunbar, W., and Murray, R., "Cooperative Control of multi-vehicle systems using cost graph optimization," *American Control Conference*, Vol. 3, 2003, pp. 2217–2222.

[12]Klavins, E. and Murray, R., "Distributed Algorithms for Cooperative Control," *IEEE Pervasive Computing*, Vol. 3, No. 1, 2004, pp. 56–65.

[13]How, J., Bethke, B., Frank, A., Dale, D., and Vian, J., "Real-Time Indoor Autonomous Vehicle Test Environment," *Control Systems Magazine, IEEE*, 2008.

[14]Karaman, S. and Frazzoli, E., "Vehicle Routing with Linear Temporal Logic Specifications: Applications to Multi-UAV Mission Planning," *AIAA Guidance, Navigation, and Control Conference*, 2008.

[15]Karaman, S. and Frazzoli, E., "Optimal Vehicle Routing with Metric Temporal Logic Specifications," *IEEE Conference on Decision and Control*, 2008.

[16]Enright, J. and Frazzoli, E., "UAV Routing in a Stochastic Time-varying Environment," *IFAC World Congress*, 2005.

[17]Schneider, K., *Verification of Reactive Systems*, Springer, 2004.

[18]Faied, M. and Girard, A., "Dynamic Optimal Control of Multiple Depot Vehicle Routing Problem with Metric Temporal Logic," *To appear in American Control Conference*, 2009.

[19]Karaman, S. and Frazzoli, E., "Specification and Planning of UAV Missions: A Process Algebra Approach," *American Control Conference*, 2009.

[20]Baeten, J., "A Brief History of Process Algebra," *Theoretical Computer Science*, Vol. 335, 2005, pp. 131–146.

[21]Baeten, J., *Process Algebra*, Cambridge University Press, 1990.

[22]Rasmussen, S. and Kingston, D., "Assignment of heterogeneous tasks to a set of heterogenous unmanned aerial vehicles," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2008.