2013 IEEE International Conference on Robotics and Automation (ICRA)
Karlsruhe, Germany, May 6-10, 2013

Temporal Logic Control for an Autonomous Quadrotor in a Nondeterministic Environment

Alphan Ulusoy* Michael Marrazzo† Konstantinos Oikonomopoulos† Ryan Hunter† Calin Belta*†

Abstract— **We present an experimental setup for automatic deployment of a quadrotor in an environment with known topology and nondeterministically changing properties. The missions are specified as rich, temporal logic statements about the satisfaction of the properties. The main objective is to be able to synthesize, test, and evaluate control policies for complex aerial missions. Our testbed consists of quadrotors, a motion capture system that provides precise and continuous position information of the quadrotor, projectors that can emulate dynamically changing environments, physical obstacles, and computers that control the quadrotor, the motion capture system, and the projectors. Our computational approach is hierarchical. At the bottom level, we partition the environment and construct an abstraction in the form of a finite transition system such that the quadrotor can execute its transitions by using low level feedback controllers. At the top level, we draw inspiration from LTL model checking and use a value iteration algorithm to determine an optimal control policy that guarantees the satisfaction of the specification under nondeterministically changing properties. We illustrate the approach for the particular case of a surveillance mission in a city-like environment.**

I. Introduction

The last decade has witnessed many exciting advances in the field of small scale unmanned aerial vehicles (UAVs), commonly referred to as micro air vehicles (MAVs). As opposed to typical UAVs with wing spans exceeding 1 m and gross takeoff weights exceeding 5 kg, MAVs are typically smaller than 15 cm in all dimensions with gross takeoff weights less than 200 g [1], [2]. Among different MAVs, quadrotors have attracted significantly more attention than their similarly sized counterparts. This has been mostly due to their ability to perform vertical takeoff and landing, operate in closed spaces with small volumes, and their relative operational safety when compared to a single rotor vehicle with a considerably larger spinning blade.

Current literature on MAVs focuses on design [3]–[5] and control [5]–[9] problems involving quadrotors. There are also various works on autonomous quadrotor navigation and planning [10]–[12]. The majority of such works either focus on waypoint based navigation or classical reach-avoid problems, where a quadrotor is expected to reach a goal region safely. Recently, several groups presented various UAV testbeds [7], [13]–[15]. Most of these testbeds are geared towards flight performance evaluation of various low-level controllers.

This work was partially supported by the ONR under grants MURI N00014-09-1051 and MURI N00014-10-10952 and by NSF under grant CNS-1035588.

* Division of Systems Engineering, Boston University, Boston, MA 02215 (alphan@bu.edu, cbelta@bu.edu)

† Dept. of Mechanical Engineering, Boston University, Boston, MA 02215 (marrazzo@bu.edu, kotsos@bu.edu, rghunter@bu.edu, cbelta@bu.edu)

In this paper, we focus on complex aerial missions expressed as temporal logic statements over some properties of interest. We present an experimental setup for automatic deployment of quadrotors in known environments with nondeterministically changing properties. The hardware part of the experimental setup consists of an in-house built quadrotor, a motion capture system, projectors, computers, and a miniature model of the environment. We use the projectors to emulate the dynamically changing events and use 3-D objects to mimic static obstacles in the environment. Low-level controllers utilize the accurate position information from the motion capture system to guide the quadrotor from one point to another in the environment.

The computational part takes as input the mission specification expressed in a fragment of Linear Temporal Logic (LTL), called syntactically co-safe LTL (scLTL) [16], and the topology of the environment, and computes a reactive control strategy guaranteeing the satisfaction of the mission. Our approach is based on a partition of the environment guided by regions of interest, followed by control synthesis on a finite nondeterministic transition system capturing the changing properties on the quotient of the partition. The control synthesis procedure, which is performed offline, relies on the conversion of the specification to a finite state automaton, the computation of the product between the automaton and the transition system, and a value iteration algorithm that yields an optimal satisfying control policy.

As a case-study, we consider a surveillance mission, in which the quadrotor is deployed from a base and is required to satisfy a rich temporal logic specification about properties that can change nondeterministically at the regions while avoiding obstacles at all times. An overhead view of our testbed, which shows the four surveillance regions, a base station, and the two obstacles can be seen in Fig. 1. An example specification is "Start at the base, collect either reward 1 or both reward 2 and reward 3 before returning to the base. Do not incur both damage 1 and damage 2".

The rest of this paper is organized as follows: In Sec. II we formally the state the problem of autonomous control in nondeterministic environments, and provide our approach in Sec. III. We discuss the details of our implementation in Sec. IV. The results of our experiments are presented in Sec. V. We conclude with final remarks in Sec. VI.

II. Problem Formulation and Approach

In this section we introduce the control synthesis problem for a quadrotor operating in a known environment with nondeterministically changing properties under temporal logic constraints.

978-1-4673-5643-5/13/$31.00 ©2013 IEEE

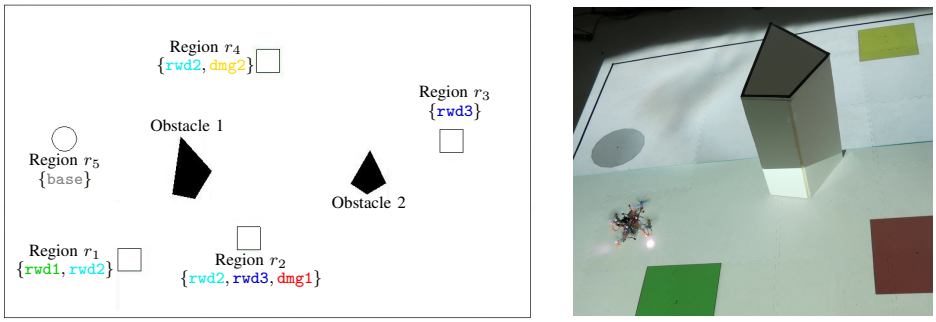

Fig. 1. Left figure shows a schematic representation of a city environment with four surveillance regions r_1, r_2, r_3, r_4, a base region r_5, and two obstacles. We have $\Pi = \{\texttt{rwd1}, \texttt{rwd2}, \texttt{rwd3}, \texttt{dmg1}, \texttt{dmg2}, \texttt{base}\}$, and the properties that can be satisfied nondeterministically at regions r_1, r_2, r_3, r_4, and r_5 are $\{\texttt{rwd1}, \texttt{rwd2}\}$, $\{\texttt{rwd2}, \texttt{rwd3}, \texttt{dmg1}\}$, $\{\texttt{rwd3}\}$, $\{\texttt{rwd2}, \texttt{dmg2}\}$, and $\{\texttt{base}\}$, respectively. Right figure gives an overhead view of a portion of the experimental setup with the quadrotor, regions r_1, r_2, r_4, r_5, and obstacle 1. The different colors of the regions represent different properties and we use green, cyan, blue, red, yellow, and gray to denote properties rwd1, rwd2, rwd3, dmg1, dmg2, and base, respectively.

Let $\mathcal{E}$=$(b, R, O, \Pi, \sim)$ be a planar polytopic environment, where b is a polytope defining the area, $R = \{r_1, ..., r_l\}$ is a finite set of polytopic regions of interest, $O = \{o_1, ...o_m\}$ is a finite set of disjoint polytopic obstacles, $\Pi = \{\pi_1, ..., \pi_n\}$ is a finite set of properties that can be satisfied at the regions of the environment, and $\sim \subseteq R \times \Pi$ is a satisfaction relation between R and Π. In this paper, we consider the case where the property satisfied at a region changes nondeterministically, *i.e.*, for a region $r \in R$, any property that belongs to the set $\{\pi | (r, \pi) \in \sim\}$ can be satisfied there. We assume that the actual property satisfied at a region can be sensed by the quadrotor when it flies over that region.

Example II.1. *Fig. 1 illustrates a planar environment where surveillance locations r_1, r_2, r_3, and r_4 are represented by squares, the base region r_5 is represented by a circle, and the obstacles are represented by filled polygons. The set of properties that can be satisfied at the regions of the environment are given by* $\Pi = \{\texttt{rwd1}, \texttt{rwd2}, \texttt{rwd3}, \texttt{dmg1}, \texttt{dmg2}, \texttt{base}\}$, *where* rwd1, rwd2, *and* rwd3 *stand for rewards of type 1, 2, and 3 that need to be gathered during the surveillance mission, and* dmg1, dmg2 *are damages that the quadrotor may incur during the mission. The properties that the quadrotor can observe at regions r_1, r_2, r_3, r_4, and r_5 are given by the sets* $\{\texttt{rwd1}, \texttt{rwd2}\}$, $\{\texttt{rwd2}, \texttt{rwd3}, \texttt{dmg1}\}$, $\{\texttt{rwd3}\}$, $\{\texttt{rwd2}, \texttt{dmg2}\}$, *and* $\{\texttt{base}\}$, *respectively. Thus, the satisfaction relation is defined as* $\sim = \{(r_1, \texttt{rwd1}), (r_1, \texttt{rwd2}), (r_2, \texttt{rwd2}), (r_2, \texttt{rwd3}), (r_2, \texttt{dmg1}), (r_3, \texttt{rwd3}), (r_4, \texttt{rwd2}), (r_4, \texttt{dmg2}), (r_5, \texttt{base})\}$.

We consider mission specifications expressed as syntactically co-safe LTL (scLTL) formulas [16]. Informally, an scLTL formula ϕ over a set Π of atomic propositions comprises Boolean operators $\neg$ (negation), $\vee$ (disjunction) and $\wedge$ (conjunction), and temporal operators $\mathbf{X}$ (next), $\mathcal{U}$ (until) and $\mathbf{F}$ (eventually). The semantics of scLTL formulas are defined over infinite sequences of symbols called words $\omega = \omega^0, \omega^1, \ldots$ over 2^Π such that $\omega^i \in 2^\Pi \ \forall \ i \geq 0$. For example, $\mathbf{X}\,\pi$ states that proposition π is true at the next position of the word, $\mathbf{F}\,\pi$ states that π is true at some future position of the word, and $\pi_1 \,\mathcal{U}\, \pi_2$ states that there is a future position of the word when π_2 is true, and π_1 is true at least until π_2 is true. A particular property of scLTL is that one can determine if a given infinite word satisfies an scLTL formula by considering only a finite prefix of it [16].

Since our experimental setup includes an accurate motion capture system (see Sec. IV), we assume that the position and the orientation of the quadrotor are precisely known, and the quadrotor can be controlled such that its center of mass can follow an arbitrary desired trajectory (see Sec. IV). As it will become clear in Sec. III-A, in our case, these trajectories are made up of straight line segments between the centers of the regions in the environment. As the quadrotor flies in the environment, the projection of its center produces a trajectory on the plane of the environment. This trajectory, in return, produces a word over Π, which can be checked against a given scLTL formula ϕ. We assume that the quadrotor can perfectly observe which properties are satisfied at a region, *i.e.*, it can actually keep track of the word that it generates as it flies. Based on the fact that the quadrotor can travel between the regions by following the lines connecting their centers, we define a control policy μ as a time-varying map from the current region and property to the next region to be visited. The control policy will be formally defined in Sec. III-B. As the satisfaction of an scLTL formula ϕ can be guaranteed in finite time, the maximum total distance that the quadrotor must travel until ϕ is satisfied is also bounded regardless of the nondeterministic nature of the properties. This maximum distance, however, depends on the policy that the quadrotor follows during its flight. Motivated by this observation, we aim to synthesize control policies that satisfy ϕ and minimize the worst-case total distance traveled by the quadrotor. We can now formulate the problem that we consider in this paper.

Problem II.2. *Given an environment $\mathcal{E}$ and a syntactically co-safe LTL formula ϕ over Π, generate a quadrotor control policy $\mu^\star$ such that the produced trajectory satisfies ϕ and minimizes the worst-case total traveled distance.*

Example II.1 Revisited. *We require the quadrotor to complete the following mission in the environment illustrated in Fig. 1. Starting at the base, the quadrotor must collect either* rwd1 *or both* rwd2 *and* rwd3 *before reaching base again. Throughout the mission, the quadrotor is allowed to incur either one of* dmg1 *or* dmg2 *but not both. This mission can be expressed as the syntactically co-safe LTL formula:*

$$\begin{aligned}\phi := \texttt{base} \wedge \mathbf{X}(((\neg\texttt{base}\ \mathcal{U}\ \texttt{rwd1}) \vee \\ ((\neg\texttt{base}\ \mathcal{U}\ \texttt{rwd2}) \wedge (\neg\texttt{base}\ \mathcal{U}\ \texttt{rwd3}))) \wedge \\ ((\neg\texttt{dmg1}\ \mathcal{U}\ \texttt{base}) \vee (\neg\texttt{dmg2}\ \mathcal{U}\ \texttt{base}))).\end{aligned} \quad (1)$$

Our solution to Prob. II.2 comprises the following steps: Given the models of the environment and the quadrotor, we first construct the environment graph $\mathbf{E}$ whose edges give the shortest routes between the regions in the environment (see Sec. III-A). Next, we use the environment graph $\mathbf{E}$, the satisfaction relation $\sim$, and the formula ϕ to synthesize a control policy $\mu^\star$ that both satisfies ϕ and minimizes the worst-case total distance traveled by the quadrotor (see Sec. III-B). After computing the high-level control policy $\mu^\star$ offline as given above, we implement it online using low-level controllers (see Sec. IV-B).

III. Problem Solution

A. Environment Graph

Given an environment $\mathcal{E}$=$(b, R, O, \Pi, \sim)$ as discussed in Sec. II, we define $\mathbf{E} = (\mathcal{Q}_\mathrm{E}, \delta_\mathrm{E}, w_\mathrm{E})$ to be the corresponding environment graph where $\mathcal{Q}_\mathrm{E}$ is the set of labels of the regions in R, δ_E is the set of edges such that $(r_i, r_j) \in \delta_\mathrm{E}$ if and only if the quadrotor can go from region r_i to r_j without passing through any obstacles or any other regions in the environment, and $w_\mathrm{E}(r_i, r_j)$ gives the Euclidean length of the shortest path between r_i and r_j. To see if there is an edge between regions $r_i, r_j \in \mathcal{Q}_\mathrm{E}$, where $r_i \neq r_j$, and to determine its weight if it exists, we use the classical configuration space approach [17] in which the quadrotor is represented as a single point.

Let $\mathbf{C}_{i,j}$ denote the configuration space that we use to define the edge between r_i and r_j. We construct $\mathbf{C}_{i,j}$ as follows. Starting from the assumption that the quadrotor can detect a property satisfied at a region if its center is vertically aligned with the center of the region, we shrink regions r_i and r_j to their centroids. Then, we consider the smallest square that encloses all possible rotations of the quadrotor, and enlarge it by an experimentally determined safety margin. We shrink the boundary b of the environment and enlarge the obstacles in O and the regions in $R \backslash \{r_i, r_j\}$ by sliding this square along their original borders.

Within the defined configuration space, we want to find the shortest path between regions r_i and r_j that does not go through any of the enlarged obstacles in O or any of the enlarged regions in $R \backslash \{r_i, r_j\}$. As shown in [17], the shortest path between any two points in a polytopic environment is a polygonal path whose inner vertices are vertices in a visibility graph defined on the configuration space. As we assume that the environment represents a city on a larger scale, we ignore the dynamics of the vehicle during this construction and consider only straight line trajectories. We construct the visibility graph [17] $\mathbf{V}_{i,j}$, where the set of nodes consists of the vertices of all enlarged obstacles in O, the vertices of all enlarged regions in $R \setminus \{r_i, r_j\}$, and the centroids of regions r_i and r_j. The set of edges of $\mathbf{V}_{i,j}$ is defined such that an edge between two vertices exists if and only if they are in direct line of sight of each other, *i.e.*, edges cannot go through any enlarged obstacles or regions, and the weight of an edge is defined to be the Euclidean distance between the vertices it connects. Finally, we define δ_E such that $(r_i, r_j) \in \delta_\mathrm{E}$ if and only if there is a path between r_i and r_j in $\mathbf{V}_{i,j}$ and define $w_\mathrm{E}(r_i, r_j)$ as the length of the shortest path between r_i and r_j in $\mathbf{V}_{i,j}$.

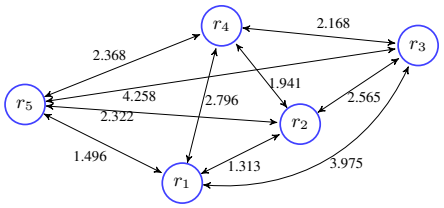

Fig. 2. Environment graph of Exp. II.1. Weight of an edge (r_i, r_j) is the distance of the shortest path between r_i and r_j in the visibility graph $\mathbf{V}_{i,j}$.

Example II.1 Revisited. *Fig. 2 shows the environment graph* $\mathbf{E}$ *that we obtain for Example II.1. The weight* $w_\mathrm{E}(r_i, r_j)$ *of an edge* $(r_i, r_j) \in \delta_\mathrm{E}$ *is the distance of the shortest path between regions* r_i *and* r_j *in the visibility graph* $\mathbf{V}_{i,j}$.

B. Control Policy Synthesis

The environment graph $\mathbf{E}$ that we obtain in Sec. III-A shows how the quadrotor can move between the regions in the environment but does not give any information regarding the properties satisfied at those regions. We use the environment graph $\mathbf{E} = (\mathcal{Q}_\mathrm{E}, \delta_\mathrm{E}, w_\mathrm{E})$ and the satisfaction relation $\sim$ to construct the nondeterministic transition system $\mathbf{T}$ (see Def. III.1), which also captures the nondeterministically changing properties at the regions of the environment.

Definition III.1 (Transition System). *A (nondeterministic, weighted) transition system (TS) is a tuple* $\mathbf{T} := (\mathcal{Q}_\mathrm{T}, \mathcal{Q}_\mathrm{T}^{init}, \mathcal{A}_\mathrm{T}, \alpha_\mathrm{T}, \delta_\mathrm{T}, \Pi_\mathrm{T}, \mathcal{L}_\mathrm{T}, w_\mathrm{T})$, *where* $\mathcal{Q}_\mathrm{T}$ *is a finite set of states;* $\mathcal{Q}_\mathrm{T}^{init} \subseteq \mathcal{Q}_\mathrm{T}$ *is the set of initial states;* $\mathcal{A}_\mathrm{T}$ *is a finite set of actions;* $\alpha_\mathrm{T} : \mathcal{Q}_\mathrm{T} \to 2^{\mathcal{A}_\mathrm{T}}$ *is a map giving the set of actions available at a state;* $\delta_\mathrm{T} \subseteq \mathcal{Q}_\mathrm{T} \times \mathcal{A}_\mathrm{T} \times \mathcal{Q}_\mathrm{T}$ *is the transition relation;* Π_T *is a finite set of atomic propositions;* $\mathcal{L}_\mathrm{T} : \mathcal{Q}_\mathrm{T} \to \Pi_\mathrm{T}$ *is a satisfaction map giving the atomic proposition satisfied at a state;* $w_\mathrm{T} : \delta_\mathrm{T} \to \mathbb{R}_{>0}$ *is a map that assigns a positive weight to each transition.*

The states of $\mathbf{T}$ are the pairs in $\sim$, *i.e.*, $\mathcal{Q}_\mathrm{T} = \sim$, $\mathcal{Q}_\mathrm{T}^{init} = \{(r^{init}, \pi) | (r^{init}, \pi) \in \mathcal{Q}_\mathrm{T}\}$ where r^{init} is the initial region of the quadrotor, $\Pi_\mathrm{T} = \Pi$, and $\mathcal{L}_\mathrm{T}((r, \pi)) = \pi \ \forall \ (r, \pi) \in \mathcal{Q}_\mathrm{T}$. The set of actions of $\mathbf{T}$ are given by $\mathcal{A}_\mathrm{T} = \delta_\mathrm{E}$ and an action $(r_i, r_j) \in \mathcal{A}_\mathrm{T}$ is enabled at all corresponding states in $\mathcal{Q}_\mathrm{T}$ where the first element of the tuple is r_i. These actions are the high-level controls of the from *go from* r_i *to* r_j which we use to drive the quadrotor between the regions in the environment. The trajectory that the quadrotor follows to go from r_i to r_j is the shortest path between the corresponding vertices on the visibility graph $\mathbf{V}_{i,j}$, which is made up of straight line segments. We discuss how we translate these high-level controls to quadrotor attitude commands by the low-level control loop in Sec. IV-B. For each $(r_i, \pi_i), (r_j, \pi_j) \in \mathcal{Q}_\mathrm{T}$, where $(r_i, r_j) \in \delta_\mathrm{E}$, we define a transition $((r_i, \pi_i), a, (r_j, \pi_j))$ in $\mathbf{T}$ such that $a = (r_i, r_j)$ and the weight of the transition is defined as the weight given by $\mathbf{E}$, *i.e.*, $w_\mathrm{T}((r_i, \pi_i), a, (r_j, \pi_j)) = w_\mathrm{E}(r_i, r_j)$.

Then, given the scLTL formula ϕ, we obtain the deterministic FSA $\mathbf{A}$ that recognizes all and only the finite words that satisfy ϕ. For any scLTL formula ϕ over a set Π, one can construct a FSA with input alphabet 2^Π accepting all and only the finite words over 2^Π that satisfy ϕ, as defined next.

Definition III.2 (Finite State Automaton). *A (deterministic) finite state automaton (FSA) is a tuple* $\mathbf{A} :=$

$(\mathcal{Q}_\mathrm{A}, q_\mathrm{A}^{init}, \Sigma_\mathrm{A}, \delta_\mathrm{A}, \mathcal{F}_\mathrm{A})$*, where* $\mathcal{Q}_\mathrm{A}$ *is a finite set of states;* $q_\mathrm{A}^{init} \in \mathcal{Q}_\mathrm{A}$ *is the initial state;* Σ_A *is an input alphabet;* $\delta_\mathrm{A} : \mathcal{Q}_\mathrm{A} \times \Sigma_\mathrm{A} \to \mathcal{Q}_\mathrm{A}$ *is a deterministic transition function;* $\mathcal{F}_\mathrm{A} \subseteq \mathcal{Q}_\mathrm{A}$ *is a set of accepting (final) states.*

A *run* of $\mathbf{A}$ over an input word $\omega = \omega^0, \omega^1, \ldots, \omega^l$ where $\omega^i \in \Sigma_\mathrm{A}\ \forall\ i = 0 \ldots l$ is a sequence $q^0, q^1, \ldots, q^l, q^{l+1}$, such that $\delta_\mathrm{A}(q^i, \omega^i) = q^{i+1}\ \forall i = 0 \ldots l$, and $q^0 = q_\mathrm{A}^{init}$. An FSA $\mathbf{A}$ accepts a word over Σ_A if and only if the corresponding run ends in some $q \in \mathcal{F}_\mathrm{A}$. Such an FSA can be constructed using readily available model checking tools [18]. Next, we define the product automaton $\mathbf{P}$ as the product of $\mathbf{T}$ and $\mathbf{A}$ as follows.

Definition III.3 (Product Automaton). *The product* $\mathbf{T} \otimes \mathbf{A}$ *of the nondeterministic transition system* $\mathbf{T}$ *and the deterministic finite state automaton* $\mathbf{A}$ *is a tuple* $\mathbf{P} := (\mathcal{Q}_\mathrm{P}, \mathcal{Q}_\mathrm{P}^{init}, \mathcal{A}_\mathrm{P}, \alpha_\mathrm{P}, \Pi_\mathrm{P}, \mathcal{L}_\mathrm{P}, \delta_\mathrm{P}, w_\mathrm{P}, \mathcal{F}_\mathrm{P})$*, where* $\mathcal{Q}_\mathrm{P} \subseteq \mathcal{Q}_\mathrm{T} \times \mathcal{Q}_\mathrm{A}$ *such that a state* $q \in \mathcal{Q}_\mathrm{P}$ *exists if and only if it is reachable from the initial states;* $\mathcal{Q}_\mathrm{P}^{init} = \{(q_\mathrm{T}, q_\mathrm{A}) | q_\mathrm{T} \in \mathcal{Q}_\mathrm{T}^{init}, \delta_\mathrm{A}(q_\mathrm{A}^{init}, \mathcal{L}_\mathrm{T}(q_\mathrm{T})) = q_\mathrm{A}\}$*;* $\mathcal{A}_\mathrm{P} = \mathcal{A}_\mathrm{T}$*;* $\alpha_\mathrm{P}((q_\mathrm{T}, q_\mathrm{A})) = \{a | a \in \alpha_\mathrm{T}(q_\mathrm{T}), \exists\, (q'_\mathrm{T}, q'_\mathrm{A}) \in \mathcal{Q}_\mathrm{P}$ *s.t.* $((q_\mathrm{T}, q_\mathrm{A}), a, (q'_\mathrm{T}, q'_\mathrm{A})) \in \delta_\mathrm{P}\}$*;* $\Pi_\mathrm{P} = \Pi_\mathrm{T}$*;* $\mathcal{L}_\mathrm{P}((q_\mathrm{T}, q_\mathrm{A})) = \mathcal{L}_\mathrm{T}(q_\mathrm{T})$*;* $\delta_\mathrm{P} = \{((q_\mathrm{T}, q_\mathrm{A}), a, (q'_\mathrm{T}, q'_\mathrm{A})) | (q_\mathrm{T}, a, q'_\mathrm{T}) \in \delta_\mathrm{T}, \delta_\mathrm{A}(q_\mathrm{A}, \mathcal{L}_\mathrm{T}(q'_\mathrm{T})) = q'_\mathrm{A}\}$*;* $w_\mathrm{P}((q_\mathrm{T}, q_\mathrm{A}), a, (q'_\mathrm{T}, q'_\mathrm{A})) = w_\mathrm{T}(q_\mathrm{T}, a, q'_\mathrm{T})\ \forall\ ((q_\mathrm{T}, q_\mathrm{A}), a, (q'_\mathrm{T}, q'_\mathrm{A})) \in \delta_\mathrm{P}$*;* $\mathcal{F}_\mathrm{P} = \{(q_\mathrm{T}, q_\mathrm{A}) | (q_\mathrm{T}, q_\mathrm{A}) \in \mathcal{Q}_\mathrm{P}, q_\mathrm{A} \in \mathcal{F}_\mathrm{A}\}$.

Note that $\mathbf{P}$ is actually a nondeterministic weighted transition system (Def. III.1) with the addition of the set $\mathcal{F}_\mathrm{P}$ of final states, and captures both the requirements of the mission and the behavior of the quadrotor in the environment. Thus, Prob. II.2 can be reduced to the problem of computing an optimal control policy $\mu_\mathrm{P}^\star$ for $\mathbf{P}$ which minimizes the maximum cost of reaching a state in $\mathcal{F}_\mathrm{P}$ from $\mathcal{Q}_\mathrm{P}^{init}$ by taking the optimal action at each state. We solve this problem using value iteration [17] as we explain next.

Value iteration is a numerical method that iteratively converges to an optimal policy given the cost of taking an action at a state [17]. Let $J^k(q, a)$ and μ_P^k denote the cost of taking action $a \in \alpha_\mathrm{P}(q)$ at state $q \in \mathcal{Q}_\mathrm{P}$ and the policy, respectively, at the k^{th} iteration. First, we set $J^0(q, a) = 0$ for all $q \in \mathcal{F}_\mathrm{P}$ and $J^0(q, a) = \infty$ for all $q \in \mathcal{Q}_\mathrm{P} \setminus \mathcal{F}_\mathrm{P}$. We also set $\mu_\mathrm{P}^0(q)$ to some $a \in \alpha_\mathrm{P}(q)$ for all $q \in \mathcal{Q}_\mathrm{P}$. Then, at each iteration k, $k > 0$, we update the cost of taking action $a \in \alpha_\mathrm{P}(q)$ at each state $q \in \mathcal{Q}_\mathrm{P}$ such that $J^k(q, a) = 0$ for $q \in \mathcal{F}_\mathrm{P}$, $J^k(q, a) = \max_{q' \in Post(q,a)} \{w_\mathrm{P}(q, a, q') + J^{k-1}(q', \mu_\mathrm{P}^{k-1}(q'))\}$ for $q \notin \mathcal{F}_\mathrm{P}$, where $Post(q, a) = \{q' | (q, a, q') \in \delta_\mathrm{P}\}$ is the set of states that we can reach after taking action a at state q. Then, we update the policy for each state such that $\mu_\mathrm{P}^k(q) = \arg\min_{a \in \alpha_\mathrm{P}(q)} J^k(q, a)$.

Notice that, at the k^{th} iteration, $J^k(q, a)$ gives our estimate of the maximum distance that the quadrotor has to travel if we take action a at state q. Next, we define an arbitrarily small threshold $thr \geq 0$ such that we terminate when $\max_{q \in \mathcal{Q}_\mathrm{P}} |J^k(q, \mu_\mathrm{P}^k(q)) - J^{k-1}(q, \mu_\mathrm{P}^{k-1}(q))| \leq thr$, and define the cost of a control policy μ_P for a product automaton $\mathbf{P}$ as $J(\mu_\mathrm{P}) = \max_{q \in \mathcal{Q}_\mathrm{P}^{init}} J(q, \mu_\mathrm{P}(q))$. Upon termination at the end of some iteration k, the policy μ_P^k is the optimal policy $\mu_\mathrm{P}^\star$ which satisfies $J(q, \mu_\mathrm{P}^\star(q)) = \min_{a \in \alpha_\mathrm{P}(q)} J(q, a)\ \forall\ q \in \mathcal{Q}_\mathrm{P}$, and $J(\mu_\mathrm{P}^\star) = \min_{\mu \in M} J(\mu)$, where we used M to denote the set of all admissible policies. Thus, if there exists a control policy with finite cost, we are guaranteed to find it, and if $J(\mu_\mathrm{P}^\star)$ is finite, then $\mu_\mathrm{P}^\star$ also satisfies ϕ.

Given a control policy $\mu_\mathrm{P}^\star$ for $\mathbf{P}$, the corresponding control policy $\mu_\mathrm{T}^\star$ for $\mathbf{T}$ takes the form of a control automaton, which we define next.

Definition III.4 (Control Automaton). *Given the control policy* $\mu_\mathrm{P}^\star$ *for* $\mathbf{P}$*, the corresponding control policy* $\mu_\mathrm{T}^\star$ *for* $\mathbf{T}$ *is a control automaton* $\mathbf{C} = (\mathcal{Q}_\mathrm{C}, q_\mathrm{C}^{init}, \Sigma_\mathrm{C}, \delta_\mathrm{C}, \psi_\mathrm{C})$*, where*

- $\mathcal{Q}_\mathrm{C} = \mathcal{Q}_\mathrm{P} \cup \{q_\mathrm{C}^{init}\}$ *is the set of states;*
- $q_\mathrm{C}^{init} = (\emptyset, q_\mathrm{A}^{init})$ *is the initial state;*
- $\Sigma_\mathrm{C} = \mathcal{Q}_\mathrm{T}$ *is the input alphabet;*
- $\delta_\mathrm{C} : \mathcal{Q}_\mathrm{C} \times \Sigma_\mathrm{C} \to \mathcal{Q}_\mathrm{C}$ *is the next state function* $\delta_\mathrm{C}(q_\mathrm{C}, q'_\mathrm{T}) = q'_\mathrm{C}$*, with* $q'_\mathrm{C} = (q'_\mathrm{T}, q'_\mathrm{A})$*, such that*
  - *if* $q_\mathrm{C} = q_\mathrm{C}^{init}$*, then* $q'_\mathrm{T} \in \mathcal{Q}_\mathrm{T}^{init}$*,* $q'_\mathrm{A} = \delta_\mathrm{A}(q_\mathrm{A}^{init}, \mathcal{L}_\mathrm{T}(q'_\mathrm{T}))$*,* $q'_\mathrm{C} \in \mathcal{Q}_\mathrm{P}^{init}$*, and*
  - *if* $q_\mathrm{C} \neq q_\mathrm{C}^{init}$*, then* $q_\mathrm{C} = (q_\mathrm{T}, q_\mathrm{A})$*,* $q'_\mathrm{A} = \delta_\mathrm{A}(q_\mathrm{A}, \mathcal{L}_\mathrm{T}(q'_\mathrm{T}))$*,* $(q_\mathrm{C}, \mu_\mathrm{P}^\star(q_\mathrm{C}), q'_\mathrm{C}) \in \delta_\mathrm{P}$.
- $\psi_\mathrm{C} : \mathcal{Q}_\mathrm{C} \to \mathcal{A}_\mathrm{T}$ *is the output function, such that* $\psi_\mathrm{C}(q_\mathrm{C}) = \mu_\mathrm{P}^\star(q_\mathrm{C})$ *if* $q_\mathrm{C} \neq q_\mathrm{C}^{init}$*,* $\psi_\mathrm{C}(q_\mathrm{C})$ *is undefined otherwise.*

Note that the control automaton defined above takes as input the current region and observation of the quadrotor, and uses this information to keep track of the progress of the quadrotor on $\mathbf{P}$. Once $\mathbf{C}$ leaves its initial state q_C^{init}, its current state q_C is the state of the quadrotor on $\mathbf{P}$, and its output $\psi_\mathrm{C}(q_\mathrm{C})$ gives the optimal action to take at each state. Thus, for any sequence of region-property pairs corresponding to some trajectory of the quadrotor, the actions given by $\mu_\mathrm{P}^\star$ and $\mu_\mathrm{T}^\star$ are identical. Consequently, if $\mu_\mathrm{P}^\star$ satisfies ϕ, then the corresponding optimal policy $\mu_\mathrm{T}^\star$ for $\mathbf{T}$ also satisfies ϕ, and therefore is a solution to Prob. II.2.

IV. Implementation

A. Hardware

The hardware used in our experimental setup consists of four Viewsonic short-throw projectors, an Optitrack motion capture system, and a quadrotor. A network of three Lenovo ThinkStation computers is used to integrate the components and serves as a ground station for computation and communication with the quadrotor. The quadrotor used in the experiments, shown in Fig. 3, is manufactured in-house at Boston University. It is equipped with a Caspa VL camera, a Gumstix Overo Fire computer, an Xbee Pro 900 radio module, and an OpenPilot CopterControl board [19].

Our testbed emulates a dynamically changing, city-like environment using both 3-D obstacles and 2-D images projected onto the ground. The four short-throw projectors are capable of projecting a single large image onto a white background on the ground with limited obstruction from the quadrotor or the physical obstacles in the environment. We represent dynamically changing properties by changing the colors of the regions. In the experiments, colors green, cyan, blue, red, yellow, and gray represent the respective properties `rwd1`, `rwd2`, `rwd3`, `dmg1`, `dmg2`, and `base`. Measurements of the positions of the quadrotor and the regions in the

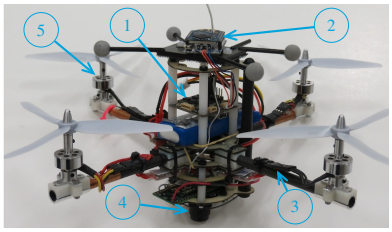

Fig. 3. The quadrotor, built around a carbon fiber frame, has the following components: (1) OpenPilot CopterControl board, (2) Xbee Pro 900 radio module, (3) Turnigy 6a ESC's, (4) Caspa VL camera with Gumstix Overo Fire board, and (5) 2580 Kv motors.

environment are obtained using an Optitrack motion capture system. The cameras track reflections of infrared light off the markers placed on the quadrotor and in the environment to provide position data with sub-millimeter precision.

B. Software

1) Construction of the Environment Graph: In the problem formulation, we assume prior knowledge of the positions of the environment boundary, regions of interest, and the obstacles. To obtain this information, we first physically construct the environment. Then, images representing the regions in the environment are projected onto the ground and physical obstacles are laid out in the workspace. Markers are placed on the vertices of all of the elements that make up environment. The motion capture system then provides the necessary information for the polytopic definitions of the environment boundary, regions of interest, and obstacles which are used to construct the environment graph.

2) Low-level Quadrotor Controllers: To execute the actions given by the control policy, the quadrotor must travel from region to region while safely avoiding obstacles. As shown in Sec. III-A, the trajectories between regions are given as a sequence of straight line segments corresponding to the edges of a visibility graph. Here we briefly describe how we map the position of the quadrotor with respect to a trajectory to the appropriate actuation of the rotors to drive the vehicle along a desired trajectory.

We first define a coordinate frame W fixed to the environment, with the $X_W - Y_W$ plane coplanar with the environment $\mathcal{E}$, and with the Z_W-axis defining the altitude. A body frame B is fixed to the quadrotor, with its origin at the center of mass of the quadrotor. The attitude of the vehicle is represented using Z-X-Y Euler Angles, describing its roll, pitch, and yaw with respect to X_W, Y_W, and Z_W axes.

For a given trajectory, we use an approach similar to [9] to create desired position and velocity setpoints at locations along the path. Using position measurements from the motion capture system, we use a linear Kalman Filter [20] to estimate the true position and velocity of the center of mass of the quadrotor relative to the setpoint. We use PID feedback control to determine a desired translational acceleration to drive the center of mass precisely along the path.

An accurate kinematics model of the quadrotor, defined by classical Euler-Lagrange equations, is described in [14]. Using this model to define the relationship between translational acceleration, total thrust, and attitude, we can express the desired attitude and total thrust as a function of the desired translational accelerations. The computation for trajectory generation, state estimation, and mapping the position measurements to desired attitude and thrust commands is performed off-board at the ground station. The resulting reference commands are sent to the quadrotor via zigbee radio communication.

The board on the quadrotor is equipped with the sensing equipment necessary to accurately measure the orientation. The reference commands from the ground station are used in a second feedback control loop, mapping the desired attitude and thrust to appropriate actuation of the rotors. Details of this mapping can be found in [19]. The low-level control design, a nested loop of position and attitude control, effectively maps the measurement data from the motion capture system to the actuation of the rotors that drives the quadrotor along prescribed trajectories.

3) Control Policy Synthesis: The formal control synthesis part of our approach detailed in Sec. III-B is implemented in Python. Our implementation uses the NetworkX graph package [21] to represent the various models that we use in our solution, and the scheck2 tool [18] to obtain the FSA $\mathbf{A}$ corresponding to an scLTL formula ϕ. Our implementation is available online[1] as part of the LTL Optimal Multi-Agent Planner (LOMAP) Python package. Given an environment graph $\mathbf{E}$ (see Sec. III-A) and an scLTL formula ϕ, our implementation follows the steps detailed in Sec. III-B to obtain the optimal satisfying control policy $\mu^\star$. Then, since the low-level quadrotor control is implemented in Matlab, our implementation exports $\mu^\star$ in the form of a Matlab function that returns the next region to be visited based on the current region and observation of the quadrotor.

V. Experimental Results

In this section, we return to the surveillance mission given in Example II.1 and present the results of our experiments where the quadrotor shown in Fig. 3 satisfies the specification ϕ given in Eq. (1). Following the steps detailed in Sec. III, we obtain the optimal satisfying control policy $\mu^\star$ and deploy the quadrotor in our testbed accordingly. The worst-case total distance that the quadrotor has to travel when it moves according to $\mu^\star$ is 9.632 m. For the sake of conciseness, we do not list the optimal action to take at each and every state of the control automaton corresponding to $\mu^\star$. Instead, we discuss two different executions of $\mu^\star$ by the quadrotor as illustrated in Fig. 4. Here, each subfigure shows the trajectory followed by the quadrotor for a particular set of properties observed at the regions. These subfigures are obtained by plotting the actual position data from the motion capture system over a schematic representation of the environment. The video accompanying the paper shows the actual corresponding flights of the quadrotor in the environment.

Run 1 (Fig. 4 top): In this run the quadrotor starts at the base (r_5) and proceeds to r_1 as per $\mu^\star$. At r_1, the quadrotor observes the property `rwd1`, and now has to return to the base to complete the mission. As given by $\mu^\star$, the quadrotor takes the optimal action and goes directly to the base (r_5) without visiting any other regions. The total distance that the quadrotor travels in this run is approximately 3 m.

[1]LTL Optimal Multi-Agent Planner (LOMAP) Python package is available at `http://hyness.bu.edu/lomap/`.

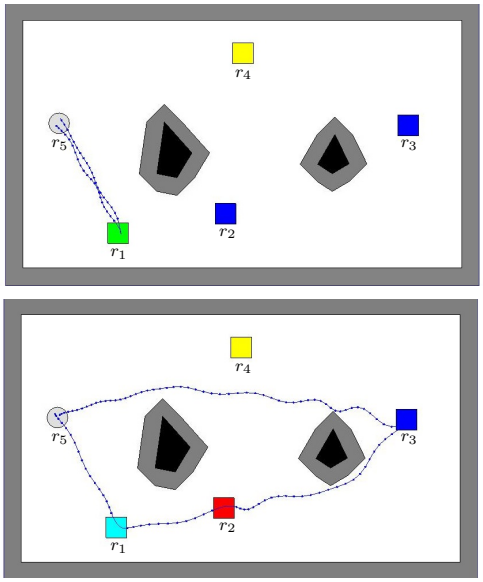

Fig. 4. Two different executions of the optimal satisfying control policy $\mu^\star$ by the quadrotor plotted over a schematic representation of the environment. Green, cyan, blue, red, yellow, and gray stand for `rwd1`, `rwd2`, `rwd3`, `dmg1`, `dmg2`, and `base`, respectively. The dark shades of gray around the environment boundary and the obstacles correspond to the construction of the configuration space (see Sec. III-A). The blue lines show the trajectories of the quadrotor as provided by the motion capture system.

Run 2 (Fig. 4 bottom): In this run, the quadrotor observes `rwd2` at r_1. Thus, the quadrotor needs to collect either `rwd3` or `rwd1` before going back to base, and it proceeds to r_2 as given by $\mu^\star$. At r_2, the quadrotor observes `dmg1`. From r_2 the quadrotor can possibly go to any one of the regions in the environment as given in Fig. 2. However, if we look closer, we realize that the quadrotor can only go to r_3, which can be explained as follows: Returning to r_5 is not an option, as the quadrotor has not collected the required rewards yet, and it cannot go to r_4 either, as doing so would risk violation of ϕ as `dmg2` can be observed there. Finally, the quadrotor cannot go back to r_1 either, as doing so could potentially result in an infinite loop where it keeps going back and forth between r_1 and r_2, observing `rwd2` and `dmg1` all the time, failing to complete the mission. Thus, as given by $\mu^\star$, the quadrotor takes the optimal action and goes to r_3, where it observes `rwd3`. Then, it returns back to base to satisfy ϕ. The total distance covered by the quadrotor in this run is approximately 10 m, which also corresponds to the worst-case total distance that the quadrotor can travel under $\mu^\star$.

Notice that, all the runs that satisfy ϕ are cyclic, *i.e.*, they all start at the base and end at the base, which is in agreement with the limited flight time of a quadrotor. Thus, in order to achieve persistent surveillance, it suffices to execute the optimal policy $\mu^\star$ repeatedly after each satisfaction of ϕ.

VI. Conclusion

We described a computational and experimental setup for automatic deployment of a quadrotor in an environment with known topology and nondeterministically changing properties. We considered mission specifications expressed as syntactically co-safe LTL formulas over the set of properties that can be satisfied in the environment. The computational framework is based on a two-level hierarchy, in which a finite abstraction of the quadrotor motion constructed at the lower level is controlled at the higher using tools from model checking and dynamic programming. The experimental setup is based on an in-house manufactured medium-size quadrotor moving in an environment equipped with a precise motion capture system and short-throw projectors that create dynamically changing events. For future work, we plan to consider probabilistic environments, richer mission specifications, more onboard sensors, and quadrotor teams.

References

[1] D. J. Pines and F. Bohorquez, "Challenges facing future micro-air-vehicle development," *Journal of Aircraft*, vol. 43, no. 2, pp. 3–24, 2006.

[2] V. Kumar and N. Michael, "Opportunities and challenges with autonomous micro aerial vehicles," *Intl. Journal of Robotics Research*, vol. 31, no. 11, pp. 1279–1291, 2012.

[3] P. Rounds, R. Maloy, P. Hynes, and J. Roberts, "Design of a four-rotor aerial robot," in *Australian Conference on Robotics and Automation*, 2002.

[4] D. Gurdan, J. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus, "Energy-efficient autonomous four-rotor flying robot controlled at 1 khz," in *IEEE Intl. Conf. Robotics and Automation*, 2007, pp. 361–366.

[5] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying," Ph.D. dissertation, EPFL, 2007.

[6] I. D. Cowling, O. A. Yakimenko, J. F. Whidborne, and A. K. Cooke, "A prototype of an autonomous controller for a quadrotor UAV," in *European Control Conference*, 2007.

[7] S. Lupashin, A. Schollig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," in *IEEE Intl. Conf. Robotics and Automation*, 2010, pp. 1642–1648.

[8] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive manuevers with quadrotors," *Intl. Journal of Robotics Research*, 2012.

[9] G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter trajectory tracking controls," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008.

[10] A. Richards, J. Bellingham, M. Tillerson, and J. How, "Co-ordination and control of multiple UAVs," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2002.

[11] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a GPS-denied environment," in *IEEE Intl. Conf. Robotics and Automation*, 2008, pp. 1814–1820.

[12] M. Achtelika, A. Bachrachb, R. Heb, S. Prenticeb, and N. Roy, "Autonomous navigation and exploration of a quadrotor helicopter in GPS-denied indoor environments," in *First Symposium on Indoor Flight Issues*, 2009.

[13] J. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *Control Systems, IEEE*, vol. 28, no. 2, pp. 51 –64, 2008.

[14] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-UAV testbed," *Robotics Automation Magazine, IEEE*, vol. 17, no. 3, pp. 56–65, 2010.

[15] P. Twu, R. Chipalkatty, J. de la Croix, A. Rahmani, M. Egerstedt, and R. Young, "A hardware testbed for multi-UAV collaborative ground convoy protection in dynamic environments," in *AIAA Modeling and Simulation Technologies Conference*, 2011.

[16] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, pp. 291–314, October 2001.

[17] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[18] T. Latvala, "Efficient model checking of safety properties," in *Model Checking Software. 10th International SPIN Workshop*. Springer, 2003, pp. 74–88.

[19] "OpenPilot," http://www.openpilot.org.

[20] G. Welch and G. Bishop, "An introduction to the Kalman filter," 1995, university of North Carolina at Chapel Hill, TR95-041.

[21] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA, 2008, pp. 11–15.