Provably Correct Persistent Surveillance for Unmanned Aerial Vehicles Subject to Charging Constraints

Kevin Leahy, Dingjiang Zhou, Cristian-Ioan Vasile, Konstantinos Oikonomopoulos, Mac Schwager, and Calin Belta

Boston University, Boston MA 02215

Abstract. In this work, we present a novel method for automating persistent surveillance missions involving multiple vehicles. Automata-based techniques were used to generate collision-free motion plans for a team of vehicles to satisfy a temporal logic specification. Vector fields were created for use with a differential flatness-based controller, allowing vehicle flight and deployment to be fully automated according to the motion plans. The use of charging platforms with the vehicles allows for truly persistent missions. Experiments were performed with two quadrotors over 50 runs to validate the theoretical results.

Keywords: Persistent Monitoring, Multi-Robot Systems, Aerial Robotics, Formal Methods

1 Introduction

In this paper, we investigate the automatic deployment of multiple quadrotors under resource constraints. The relatively short battery life in many unmanned aerial vehicles (UAVs) presents a significant barrier to their use in complex, long term surveillance missions. Moreover, the use of multiple vehicles allows for more complex behavior and longer mission horizons, but further complicates the task of deploying those vehicles given limited flight time. We present an algorithm that generates a feedback controller for multiple quadrotors with charging constraints to meet a complex temporal logic specification. The algorithm comprises a three-part tool chain that first plans a high level routing schedule for the quadrotors, then generates a vector field control input for the quadrotors to accomplish the schedule, and finally controls the quadrotors' nonlinear dynamics to follow the vector field with a feedback controller. The performance of the complete system, with its three interacting parts, is investigated in 50 experimental runs using two quadrotors and three charging stations in a motion capture environment.

We consider the following problem: given an environment and a temporal logic mission specification with time deadlines that needs to be satisfied infinitely often, generate control policies for a team of quadrotors to complete the mission, while ensuring vehicles remain charged and collisions are avoided.

As a motivating example, we consider the environment shown in Fig. 1 consisting of three charging stations, three regions of interest, and two aerial vehicles. We assume vehicle battery life is 40 time units, and charging takes 120 time units, where time units are a generic unit that can be instantiated based on a particular implementation. Given this environment and these battery and charging constraints, the vehicles must perform a persistent surveillance mission defined by a rich linear temporal formula which imposes time bounds on each loop of the vehicle's (infinite) run. Thus, the specification is given as a bounded time formula which needs to be satisfied infinitely often. In this example, we wish the multi-robot system to satisfy the following mission specification *infinitely often*: "within 16 time units observe Region R3 for at least 3 time units; within 28 time units, observe Region R1 for at least 2 time units; and within 46 time units, observe Region R2 for at least 2 time units then within 8 time units observe Region R1 or Region R3 for at least 2 time units." We seek a method to generate a control policy ensuring that vehicles can be automatically deployed to successfully complete this mission in the specified environment.

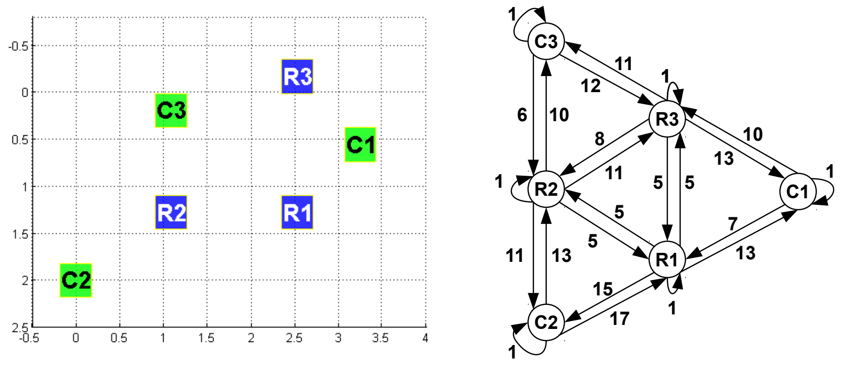


Fig. 1: Partitioned environment viewed from above and transition system. Green squares are charging stations, while blue squares are regions of interest. States in the transition system are charging stations and regions of interest. Weights on transitions are based on calculated time bounds.

The solution to this problem requires the use of several sophisticated systems, whose interaction both at a theoretical level and an experimental level produces many unique challenges. Our approach is related to the Vehicle Routing Problem (VRP) [1], which can be summarized as: given N identical vehicles at a depot and the distances among all sites and the depot, find a minimum distance tour for each vehicle such that it begins and ends at the depot and visits each site at least once. By placing time bounds on when each site must be visited, we obtain a version of VRP called Time Window VRP (VRPTW) [2]. Multi-agent control for the VRPTW has also been considered without temporal logic constraints in [3, 4]. Our work uses temporal logic constraints for the VRPTW with richer specifications.

Temporal logic and formal methods [5] have been used for robot motion planning and control in persistent surveillance as in [6, 7]. These works, while considering optimal persistent surveillance with temporal logic constraints, do not consider resource constraints. These works also do not consider time windows, which we use in this paper. Resource constraints have been modeled in the routing problem for one vehicle without temporal logic constraints in [8]. Our work allows for richer mission specification while still modeling resource constraints.

The most closely related recent work includes [9] in which the authors propose a fragment of metric temporal logic, which restricts temporal operators to atomic propositions and their negation. In that work, each site may be visited only once, and bounds on transition duration are not allowed. Additionally, their work does not take into account resource constraints, and optimizes a weighted sum of distance traveled over a finite horizon. Our approach allows for a vehicle to visit a site multiple times during a tour if it is required, capturing resource constraints, and allowing bounds on transition durations.

A version of this work, involving formal methods for creating task plans, appears in [10]. Additionally, details on the differential flatness approach to vehicle control appears in [11].

2 Technical Approach

The solution is outlined as follows: first, motion plans are generated to satisfy the mission specification in Sect. 2.1. A vector field is constructed for navigating the quadrotors, from which the transition system is abstracted as explained in Sect. 2.2. Finally, in Sect. 2.3 a differential flatness-based approach is used to control the vehicles through the previously constructed vector field.

2.1 Control Policy Generation

Generating a control policy for our persistent surveillance problem first requires creating an abstraction of the environment and quadrotor behavior, including a model of the quadrotor battery charging and discharging. By specifying the mission using a temporal logic formula, we are able to use automata theoretic techniques in conjunction with theses abstractions to synthesize a control policy.

Finite Models of the Environment and Quadrotors. For simplicity of presentation, we assume the team is made of N identical quadrotors. Consider a finite abstraction of the environment given as a graph $G = (V = \mathcal{S} \cup \mathcal{C}, E, w)$, where $\mathcal{S}$ is the set of sites and $\mathcal{C}$ is the set of charging stations or depots. An edge $e \in E \subseteq V \times V$ denotes that a vector field can be constructed such that a vehicle can move between the source and destination of the edge (see Sect. 2.2). Quadrotors can deterministically choose to traverse the edges of G, stay at a site for service, or stay docked in a charging station. A duration is associated with each edge, which represents the flight time and includes docking or undocking, if applicable, and is given by $w : E \to \mathbb{Z}_{\geq 1}$. The construction of the environment graph G is described in Sect. 2.2.

In this paper, we assume that the team has a mutual-exclusive (ME) operation mode, i.e. at any moment in time at most one quadrotor is flying. Thus, collision avoidance is conservatively guaranteed.

Each vehicle has a limited amount of battery life, specified as an integer value, and must regularly return to a charging station. The maximum operation time starting with a fully charged battery is denoted by t_{op}, while the maximum charging time starting with an empty battery is denoted by t_{ch}. The charge-discharge ratio, which denotes the amount of time required to charge the battery vs. how long the vehicle may fly on a fully-charged battery, is $\gamma = \frac{t_{ch}}{t_{op}} \geq 1$ and may only take integer values. For simplicity, we assume that time is discretized, and all durations (e.g., $w(E)$, t_{op}, t_{ch}) are expressed as an integer multiple of a time interval $\varDelta t$.

A battery is abstracted by a discrete *battery state* $b_t(i) \in \{0, \ldots, t_{ch}\}$, corresponding to quadrotor i at time $t \in \mathbb{Z}_{\geq 0}$, and an update rule, which specifies the change of charge after d time units:

$$b_{t+d}(i) = \begin{cases} \min\{b_t(i) + d, t_{ch}\} & \text{vehicle } i \text{ is docked} \\ b_t(i) - \gamma d & \text{otherwise} \end{cases} \tag{1}$$

It is assumed that the quadrotors are equipped with identical batteries. The batteries may be charged at any of the charging stations $\mathcal{C}$. Charging may start and stop at any battery state. Once a quadrotor is fully charged, it will remain fully charged until it leaves the charging station. We assume that at the start of the mission all quadrotors are fully charged and docked.

We will say that a quadrotor is *active* if it is flying, i.e. moving between sites and charging stations or servicing a request. A request at a site is said to be serviced if a quadrotor hovers above it. The time bounds in (2) represent the duration for which each site is to be serviced. A time interval in which all vehicles are docked and none are charging is called *idle time*.

Control Policy. For $q \in V$, we use $\vec{q}$ to denote that a quadrotor is flying towards q. Let $\vec{V} = \{\vec{q} \,|\, q \in V\}$. A *control policy* for the team of quadrotors is a sequence $\mathbf{v} = v_1 v_2 \ldots$ where $v_t \in (V \cup \vec{V})^N$ specifies at each time $t \in \mathbb{Z}_{\geq 0}$ and for each quadrotor $i \in \{1, \ldots, N\}$ if quadrotor i is at a site or charging station or if it is moving. Let $v_t(i)$ and $v(i)$, $i \in \{1, \ldots, N\}$, denote the control value for quadrotor i at time t and the control policy for quadrotor i (i.e., the sequence of control values), respectively. Then a transition $(q_1, q_2) \in E$ performed by quadrotor i starting at time t will correspond to $v_t(i) = q_1$, $v_{t+d}(i) = q_2$ and $v_{t+k}(i) = \vec{q}_2$, $k \in \{1, \ldots, d-1\}$, where $d = w((q_1, q_2))$ is the duration of the transition. Servicing or charging for one time interval ($\varDelta t$ time) by quadrotor i at time t corresponds to $v_t(i) = v_{t+1}(i) \in V$. A control policy $\mathbf{v} = v_1 v_2 \ldots$ determines an *output word* $\mathbf{o} = o_1 o_2 \ldots$ such that $o_t = \{v_t(i) | v_t(i) \in \mathcal{S}, i \in \{1, \ldots, N\}\}$ is the set of all sites occupied by the N quadrotors at time $t \in \mathbb{Z}_{\geq 0}$. We use ϵ to denote that no site is occupied. Note o_t is either ϵ or a singleton set, because of the ME operation mode assumption. Let $q^{[d]}$ and q^{ω} denote d and infinitely many repetitions of q, respectively.

Bounded Linear Temporal Logic. To capture the richness of the specification, we use bounded linear temporal logic (BLTL) [12], a temporal logic with time bounds on each of its temporal operators. The mission specification presented in Sect. 1 can be expressed as $\mathbb{G}\phi$, where ϕ is given in (2) as a BLTL formula and the $\mathbb{G}$ operator indicates that ϕ should be satisfied infinitely often.

$$\phi = \mathbf{F}^{\leq 16}\mathbf{G}^{\leq 3}R3 \wedge \mathbf{F}^{\leq 28}\mathbf{G}^{\leq 2}R1 \wedge \mathbf{F}^{\leq 46}(\mathbf{G}^{\leq 2}R2 \wedge \mathbf{F}^{\leq 10}\mathbf{G}^{\leq 2}(R1 \vee R3)) \qquad (2)$$

In (2), $\wedge$ and $\vee$ are the usual Boolean operators indicating conjunction and disjunction, while $\mathbf{F}$ and $\mathbf{G}$ are the temporal operators "eventually" and "always", respectively. Superscripts on the temporal operators are time bounds on those operators. Each Ri is a request associated with the region. A control policy is said to *satisfy* the persistent surveillance specification $\mathbb{G}\phi$, where ϕ is a BLTL formula, if the generated output word satisfies the BLTL formula ϕ infinitely often and there is no idle time between any two consecutive satisfactions of ϕ. Note that, between successive satisfactions of ϕ, the quadrotors may recharge their batteries, i.e. at least one may not be idle, because it is charging its battery.

Problem Formulation and Overview of the Approach. Let $\mathbf{v}$ be a control policy. We say that $\mathbf{v}$ is *feasible* if at each moment in time all N quadrotors have non-negative battery states, i.e., $b_t(i) \geq 0$ for all $i \in \{1, \ldots, N\}$ and $t \in \mathbb{Z}_{\geq 0}$.

Problem 1. Given an environment $G = (V = \mathcal{S} \cup \mathcal{C}, E, w)$, N quadrotors with operation time t_{op} and charging time t_{ch}, and a BLTL formula ϕ over $\mathcal{S}$, find a feasible control policy that satisfies $\mathbb{G}\phi$ if one exists, otherwise report failure.

Let $\mathbf{v}$ be a feasible control policy satisfying $\mathbb{G}\phi$. We define a *loop* as a finite subsequence of $\mathbf{v}$ starting with the satisfaction of the formula ϕ and ending before the next satisfaction of ϕ.

The proposed approach to Prob. 1 is based on automata techniques [5]. The motion model of the quadrotor team is represented as a product transition system between N copies of G which is pruned of any states and transitions which violate the ME operation mode. The product transition system is then composed with a finite state automaton which captures the charging constraints. The resulting product model is then composed with another finite state automaton which accepts the satisfying language corresponding to the given BLTL formula ϕ. The finite state automaton encoding ϕ is obtained by first translating it [13] to a syntactically co-safe Linear Temporal Logic formula [14] and then to an automaton using the *scheck* tool [15]. The satisfiability problem (Prob. 1) is solved on the resulting product automaton by considering all possible states of the team at the start of a loop and paths between these states obtained with Dijkstra's algorithm. For more details about the procedure see [10], where the authors prove the completeness of the proposed approach. In [10], they also consider the fully-concurrent mode of operation and optimality.

2.2 Vector Field and Transition System Weights

We use a vector field for the implementation of the control policies synthesized as explained in Sect. 2.1, because it allows for the discrete environment model to be combined with the continuous dynamics necessary for vehicle navigation. Additionally, once the vector field has been created, upper limits on travel times through the vector field provide the weights w for the environment graph G such that a control policy can be synthesized.

Partition. To generate the vector field, we first partition the environment into cubes. Each cube is defined by two vectors, $a = (a_1, a_2, a_3)$ and $b = (b_1, b_2, b_3)$ where $a_i < b_i$ for all $i = 1, 2, 3$. Thus, each cube may be written as

$$C\left(a, b\right) = \left\{x \in \mathbb{R}^3 | \forall i \in \{1, 2, 3\} : a_i \leq x_i \leq b_i\right\}. \tag{3}$$

Paths corresponding to edges in the environment are found as sequences of these cubes. The paths are constrained such that quadrotors fly to a fixed height from the charging stations and perform all observations from that fixed altitude. From these paths, we generate to ensure each sequence of cubes is followed.

Vector Field Construction. A vector field everywhere inside a given cube can be created as a convex combination of a set vectors at its vertices [16], expressed as

$$h\left(x_1, \ldots, x_N\right) = \sum_{v \in \mathbf{V}(a,b)} \prod_{i=1}^{N} \left(\frac{x_i - a_i}{b_i - a_i}\right)^{\xi_i(v_i)} \left(\frac{b_i - x_i}{b_i - a_i}\right)^{1-\xi_i(v_i)} \cdot h\left(v\right), \tag{4}$$

where x_i is the coordinate in the i^{th} dimension of a point in the cube, $\mathbf{V}\left(a, b\right)$ are the vertices of cube $C\left(a, b\right)$, $h\left(v\right)$ are the vectors at each vertex $v \in \mathbf{V}\left(a, b\right)$, $N = 3$, and $\xi_i\left(v_i\right)$ is an indicator function such that $\xi_i\left(a_i\right) = 0$ and $\xi_i\left(b_i\right) = 1$. Such a vector field can be used to keep the vehicle from leaving the cube (stay-in-cell) or to force it to leave through a given facet (control-to-facet), as displayed in Fig. 2.

For each cube in any given path, we create a control-to-facet vector field to lead to the next cube in the path. Because discontinuities in the vector field could result in undesirable behavior of the quadrotors, we must ensure that velocity is continuous from one cube to the next. We ensure continuity by examining vectors at the facet where cubes meet. For each corner of such a facet, the vectors from the two cubes are compared to each other. Only the vector components that the two vectors have in common are kept. This process is illustrated in Fig. 3. In

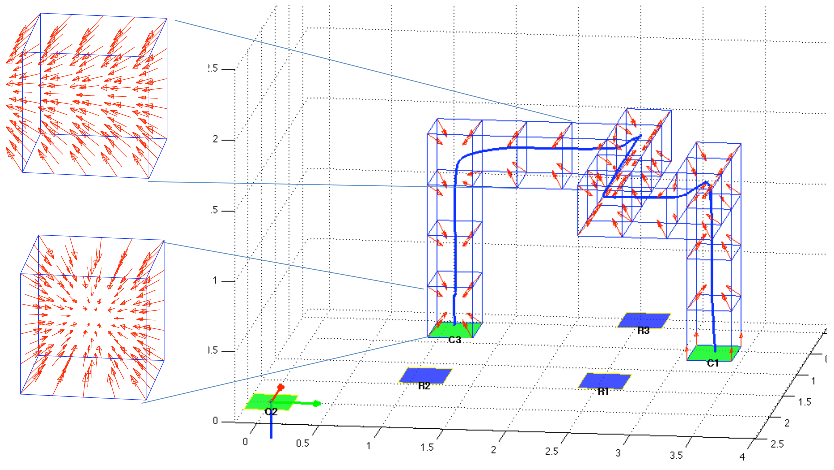


Fig. 2: Vector field detail and quadrotor flight data. The cube at the top left shows a control-to-facet vector field, and the cube at the bottom left shows a stay-in-cell vector field. One of these two kinds of fields is given to the quadrotor in each cell along its path to guide it through the desired trajectory.

the figure, cells A, B, and C are joined together, and B then shares a facet with A and C. The vectors for cell B and C on their shared facet are identical, and continuity is ensured. But the vectors on A's shared facet with B are different (Fig. 3b). Thus the vertical components of these vectors are discarded, but the horizontal components, which are identical, are kept (Fig. 3c). Because of this process, there are limitations to the types of arrangements of cubes that can be constructed, because they would result in a vector of zero magnitude (see Fig. 4b), but in practical examples, such arrangements are unlikely to be desirable and can be avoided by using a finer partition of the environment if necessary.

Weights. Because satisfaction of (2) depends on the time to travel among the regions of the environment, these times must be known. We can calculate the upper bound on the travel time between any two regions, which are captured as weights on the transition system described above. We model hovering over a region or charging as self-loop transitions of weight 1. Calculating the upper time bound for leaving a cube depends on the vectors at the vertices. If none of these vectors has a component of magnitude zero, we calculate the time bound for exiting the cube through facet F as

$$T^F = ln\left(\frac{s_F}{s_{\bar{F}}}\right)\frac{b_i - a_i}{s_F - s_{\bar{F}}}\ , \tag{5}$$

where $\bar{F}$ is the facet opposite F, and s_F, $s_{\bar{F}}$ are the minimum vector components in the i^{th} direction on facet F and $\bar{F}$, respectively [17]. In the event that s_F approaches $s_{\bar{F}}$, T^F approaches $(b_i - a_i)/s_{\bar{F}}$.

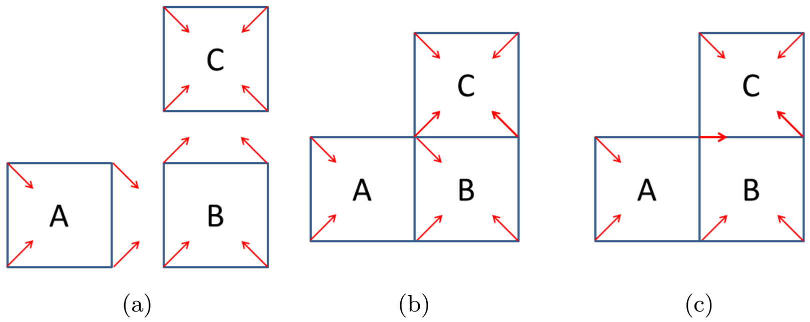


Fig. 3: Two-dimensional example of combining vectors. (a) Control-to-facet vector field from A to B and B to C, and stay-in-cell vector field for cell C. (b) Vector conflict where A, B and C meet. (c) Final vector field, keeping only non-conflicting vector components.

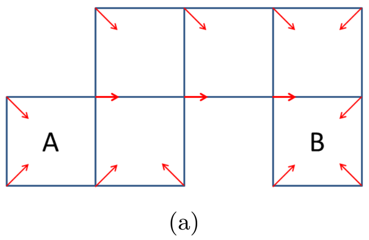
Because of the continuity requirements on the vector field, it is possible to have a vector with a component of magnitude zero (i.e. as seen in Fig. 4a). In this case, as long as there remains a non-zero component in another direction, there is a guaranteed upper bound on the time to leave the cell. This time bound, in the case of a zero-magnitude component in the i^{th} direction and a non-zero component in the j^{th} direction, while exiting in the i^{th} direction through the facet containing the zero-magnitude component, can be expressed as

$$\begin{aligned} T^F &= T_i^F + T_j^F \\ &= \left(\frac{b_i - a_i}{s^F \left(\frac{M}{2} - 1 \right)} \right) ln \left(\frac{M}{2} \right) + \left(\frac{b_j - a_j}{-2 s^{\overline{F}}} \right) ln \left(1 - M \right) \ , \end{aligned} \tag{6}$$

where $0 < M < 1$ is a measure of “conservatism.” The closer M is to 1, the larger the time bound, and the higher the guarantee of the time bound being correct. This is due to the asymptotic nature of the solution approaching the zero-magnitude component in the i^{th} direction.

2.3 Vector Field Following

Motion planning often involves the use of vector fields to be followed by a robot. This is easily accomplished with most ground robots as well as slow aerial robots. In our experiments however, we use quadrotors, which cannot easily follow a vector field because of their high dimensional, nonlinear dynamics. Thus, we exploit the differential flatness of quadrotor dynamics to design a controller which will allow the quadrotor to follow the vector field, compensating for the quadrotor’s nonlinear dynamics [11].



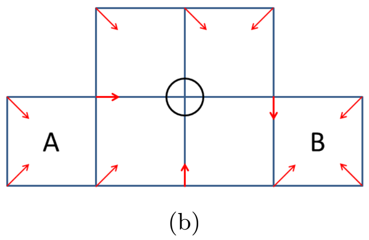


Fig. 4: Two-dimensional example of vector field configurations from A to B. (a) Allowable configuration results in vectors with some zero-magnitude components, while resulting in no vectors with zero-magnitude. (b) Not allowable configuration with an occurence of zero-magnitude for all components (circled).

Differential Flatness. Quadrotor dynamics are given by the nonlinear system of equations

$$\dot{v} = ge_3 + \frac{1}{m}Rf_ze_3 \tag{7}$$
$$\dot{R} = R\Omega \tag{8}$$
$$\dot{\omega}_b = J^{-1}\tau - J^{-1}\Omega J\omega_b \tag{9}$$
$$\dot{h} = v\,, \tag{10}$$

where $v = [v_x, v_y, v_z]^T$ is the velocity in the world frame, g is the acceleration due to gravity, m is the mass, f_z is the total thrust force from the rotors, $e_3 = [0, 0, 1]^T$, and hence f_ze_3 is aligned with the negative vertical direction of the body frame, $-z_b$. R is the rotation matrix from the world frame to the body frame, defined in terms of Euler angles ψ, θ, and ϕ. $\omega_b = [p, q, r]^T$ is the angular velocity of the quadrotor expressed in the body frame, $\Omega = \omega_b^\wedge$ is the tensor form of ω_b. The torque on the quadrotor is given by τ in the body frame F_b. J is the inertia matrix of the quadrotor, and h is the position of the quadrotor in the world frame.

The system as defined in (7)–(10) has a 12-dimensional state, $\xi = [x,\ y,\ z,\ v_x,\ v_y,\ v_z,\ \psi,\ \theta,\ \phi,\ p,\ q,\ r]^T$, and input, $\mu = [f_z, \tau_x, \tau_y, \tau_z]^T$, which is the total thrust and three torques. The state and input are differentially flat. Their flat outputs

$$\sigma = [\sigma_1, \sigma_2, \sigma_3, \sigma_4]^T := [x, y, z, \psi]^T\ , \tag{11}$$

consisting of position and yaw, are such that the state, ξ is a function of these outputs and there derivatives. More precisely, $\xi = \beta(\sigma, \dot{\sigma}, \ddot{\sigma}, \dddot{\sigma})$, with

$$\begin{cases} [x, y, z, v_x, v_y, v_z, \psi]^T = \beta_{1:7}(\sigma, \dot{\sigma}) = [\sigma_1, \sigma_2, \sigma_3, \dot{\sigma}_1, \dot{\sigma}_2, \dot{\sigma}_3, \sigma_4]^T \\ \theta = \beta_8(\sigma, \dot{\sigma}, \ddot{\sigma}) = \text{atan2}(\beta_a, \beta_b) \\ \phi = \beta_9(\sigma, \dot{\sigma}, \ddot{\sigma}) = \text{atan2}(\beta_c, \sqrt{\beta_a^2 + \beta_b^2}) \\ [p, q, r]^T = \beta_{10:12}(\sigma, \dot{\sigma}, \ddot{\sigma}, \dddot{\sigma}) = (R^T\dot{R})^\vee, \end{cases} \tag{12}$$

where

$$\begin{cases} \beta_a = -\cos\sigma_4\ddot{\sigma}_1 - \sin\sigma_4\ddot{\sigma}_2 \\ \beta_b = -\ddot{\sigma}_3 + g \\ \beta_c = -\sin\sigma_4\ddot{\sigma}_1 + \cos\sigma_4\ddot{\sigma}_2, \end{cases} \tag{13}$$

and R is the rotation matrix with the Euler angles (ϕ, θ) defined in (12). Furthermore, the input, μ, is also a function of the flat outputs, expressed as $\mu = \gamma(\sigma, \dot{\sigma}, \ddot{\sigma}, \dddot{\sigma}, \ddddot{\sigma})$, with

$$\begin{cases} f_z = \gamma_1(\sigma, \dot{\sigma}, \ddot{\sigma}) = -m \parallel \ddot{\sigma}_{1:3} - ge_3 \parallel \\ [\tau_x, \tau_y, \tau_z]^T = \gamma_{2:4}(\sigma, \dot{\sigma}, \ddot{\sigma}, \dddot{\sigma}, \ddddot{\sigma}) \\ \quad = J(\dot{R}^T\dot{R} + R^T\ddot{R})^\vee + R^T\dot{R}J(R^T\dot{R})^\vee, \end{cases} \tag{14}$$

where $\ddot{\sigma}_{1:3} = [\ddot{\sigma}_1, \ddot{\sigma}_2, \ddot{\sigma}_3]^T$ for short and the $^\vee$ map is the inverse operation of $^\wedge$. For details and a proof, please refer to [11].

With the flat outputs and their derivatives obtained as described below, the above equations can generate all the states and inputs. A standard $SE(3)$ controller [18] can be implemented to control the quadrotor flight along the vector field using the states and inputs as a control reference.

Vector Field Derivatives. The inputs described in (14) require knowledge of velocity, acceleration, jerk, and snap. Hence it is necessary to find the time derivatives ($\dot{\sigma}$, $\ddot{\sigma}$, $\dddot{\sigma}$, $\ddddot{\sigma}$) by taking spatial derivatives of the vector field. We only consider vector fields which do not specify rotation, hence the yaw angle σ_4 is irrelevant. We arbitrarily set $\sigma_4(t) \equiv 0$. In general, the flat output derivatives $\dot{\sigma}_{1:3}, \ddot{\sigma}_{1:3}, \dddot{\sigma}_{1:3}, \ddddot{\sigma}_{1:3}$ at any point x in a vector field $h(x)$ can be recursively calculated by

$$\begin{cases} \dot{\sigma}_{1:3}(x) &= h(x) \\ \ddot{\sigma}_{1:3}(x) &= \mathcal{J}(\dot{\sigma}_{1:3}(x), x)\dot{\sigma}_{1:3}(x) \\ \dddot{\sigma}_{1:3}(x) &= \mathcal{J}(\ddot{\sigma}_{1:3}(x), x)\dot{\sigma}_{1:3}(x) \\ \ddddot{\sigma}_{1:3}(x) &= \mathcal{J}(\dddot{\sigma}_{1:3}(x), x)\dot{\sigma}_{1:3}(x), \end{cases} \tag{15}$$

where $\mathcal{J}(f(x), x)$ denotes the Jacobian matrix of the function $f(x)$.

The velocity is obtained directly from the vector field described by (4), from which the derivatives required for the differential flatness controller given in (15) can be derived analytically. First (4) is rewritten in matrix form as

$$h(x_1, \ldots, x_3) = [c_1, \ldots, c_8] \begin{bmatrix} h_{1x_1} & h_{1x_2} & h_{1x_3} \\ \vdots & \vdots & \vdots \\ h_{8x_1} & h_{8x_2} & h_{8x_3} \end{bmatrix}. \tag{16}$$

In this form, the coefficients c are functions of position, but the values of h are fixed for any given cube. This form is therefore convenient for computation

of the acceleration and other vector field derivatives.
In general, the acceleration at x is given by

$$a(x) = \mathcal{J}(v(x), x)v(x) \; , \tag{17}$$

where $\mathcal{J}(f(x), x)$ denotes the Jacobian matrix of the function $f(x)$, which is a 3×3 matrix with entries

$$\mathcal{J}_{ij} = \frac{\partial v_i}{\partial x_j} = h_{1x_i}\frac{\partial c_1}{\partial x_j} + \ldots + h_{8x_i}\frac{\partial c_8}{\partial x_j} \; . \tag{18}$$

Through straightforward calculation, acceleration is therefore given by

$$a_i = \sum_{j=1}^{3}\left(\sum_{k=1}^{8} h_{kx_i}\frac{\partial c_k}{\partial x_j}\right) v_j \; . \tag{19}$$

It should be noted that the vector fields for acceleration, jerk, and snap are continuous everywhere within a given cube but may be discontinuous at the facets between cubes. Similar calculations can be done for jerk and snap and are omitted due to space limitations.

3 Results and Experiments

The partitioned environment (Figs. 1 & 7) consists of 385 cubes each with edge length 0.36m. Control policies for $\mathbb{G}\phi$ were calculated over the transition system displayed in Fig. 1. The computation time, excluding encoding of (2), was 301.7 seconds on a Linux system with a 2.1 GHz processor and 32 GB memory, and the final product automaton had 579,514 nodes and 2,079,208 edges. No solutions were found for quadrotors starting on Chargers C2 and C3, but all other combinations of starting positions yielded solutions.

Experiments were performed in the Boston University Multi-robot Systems Lab. The lab consists of a flight space with IR cameras to track reflective markers on the quadrotors using an OptiTrack system. This system allows for real-time localization of the quadrotors during experiments. Two K500 quadrotors from KMel robotics were used to execute the control policies described in Sect. 3.

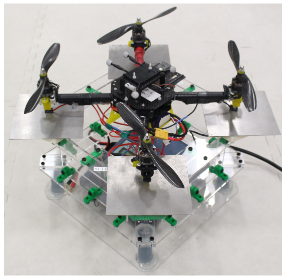


Fig. 5: Quadrotor resting on charging station.

Charging stations (Fig. 5) were designed and built at Boston University for automatic docking and charging of quadrotors. These platforms allow a vehicle to land when its battery requires charging. When using multiple such platforms, another vehicle can then take off, ensuring continuous monitoring in situations where one vehicle may not be able to satisfy a persistent monitoring mission specification on its own.

The charging stations are made of laser cut acrylic parts connected with PLA plastic 3D printed parts. The electronics of the station consist of the Hyperion EOS0720i Net3AD charger, modified to enable control by MATLAB. To secure a robust connection with the stainless steel pads of the charging station, the quadrotors are equipped with stainless steel contacts mounted on springs with magnets. The platform is entirely controlled by MATLAB via USB connection, allowing for the detection of the presence of a quadrotor, real-time monitoring of battery and charging status, and control of the charging parameters including battery type, capacity, and charging rate. The maximum charging rate that can be achieved is 8 Amperes.

Figure 6 shows the results of a flight by two quadrotors. Seconds were used as the time units for these experiments so flights could be rapidly performed and analyzed.

The quadrotors, shown in red (Quad 1) and blue (Quad 2) in Fig. 7, start fully charged from the charging stations $C1$ and $C2$, respectively. The control policy $\mathbf{v}$ for the two quadrotors, generated as described in Sect. 2.1, is the following:

$$\begin{aligned} v(1) = \quad & C1^{[1]}\vec{R1}^{[6]}R1^{[3]}\vec{R3}^{[4]}R3^{[4]}\vec{C3}^{[10]}C3^{[41]} \\ & \left(C3^{[31]}\vec{R2}^{[5]}R2^{[3]}\vec{R3}^{[10]}R3^{[3]}\vec{C3}^{[10]}\right)^{\omega} \\ v(2) = \quad & C2^{[29]}\vec{R2}^{[12]}R2^{[3]}\vec{R1}^{[10]}R1^{[3]}\vec{C1}^{[12]} \\ & \left(C1^{[1]}\vec{R1}^{[6]}R1^{[3]}\vec{R3}^{[4]}R3^{[4]}\vec{C1}^{[12]}C1^{[32]}\right)^{\omega} \ . \end{aligned} \tag{20}$$

Under control strategy (20), in the first loop Quadrotor 1 (red) take-off first and services sites $R1$ and $R3$ and Quadrotor 2 (blue) completes the loop by servicing sites $R2$ and $R1$. In all subsequent loops, Quadrotor 2 (blue) takes-off first and services sites $R1$ and $R3$ and Quadrotor 1 complete the loop by servicing sites $R2$ and $R1$. After the first loop, Quadrotors 1 and 2 always return to $C3$ and $C1$, respectively. The corresponding output word is

$$\begin{aligned} o = \quad & \epsilon^{[7]}R1^{[3]}\epsilon^{[4]}R3^{[4]}\epsilon^{[23]}R2^{[3]}\epsilon^{[10]}R3^{[3]}\epsilon^{[12]} \\ & \left(\epsilon^{[7]}R1^{[3]}\epsilon^{[4]}R3^{[4]}\epsilon^{[18]}R2^{[3]}\epsilon^{[10]}R3^{[3]}\epsilon^{[10]}\right)^{\omega} \ . \end{aligned}$$

The flights presented in the experiments consist of the first two loops each satisfying ϕ. Any subsequent loop would be identical to the second loop. Since ϕ can be satisfied repeatedly, these flights can satisfy the mission specification, $\mathbb{G}\phi$.

Figure 6 shows that the specification was satisfied for both loops in the flight. Region R1 was visited in 5.76 seconds in Loop 1 and 7.48 seconds in Loop 2, ahead of the 28 second deadline. Likewise, Region R3 was visited in 12.44 and 12.64 seconds ahead of the 16 second deadline. In the second portion of each loop, Region R2 was visited in 34.00 and 30.27 seconds with a deadline of 46 seconds, and Region R1 was visited within the 8 second deadline after each visit to Region R2.

The two-loop flight described above was performed 50 times, and both quadrotors were consistent in their flight times. The standard deviation in the length of each portion of the flight time was on the order of $0.1s$. Despite this consistency, the time bound on flying from Charger C1 to Region R1 was violated by the second quadrotor in each flight, while not being violated by the first quadrotor. While the vehicles were nominally identical, small physical differences between them required the controllers to be tuned using different values. Because both quadrotors followed the same vector field using the same controller, this time bound violation suggests some potential for better tuning of the controllers.

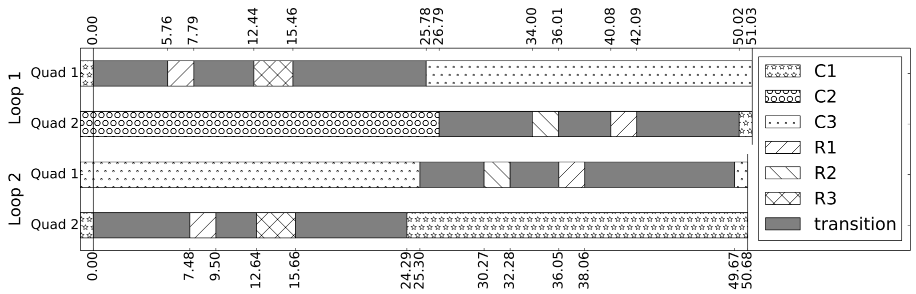


Fig. 6: Timeline of quadrotor flights for two loops. The first two rows display the first loop, with Quadrotor 1 flying before Quadrotor 2. The next two rows show the second loop, with Quadrotor 2 flying first.

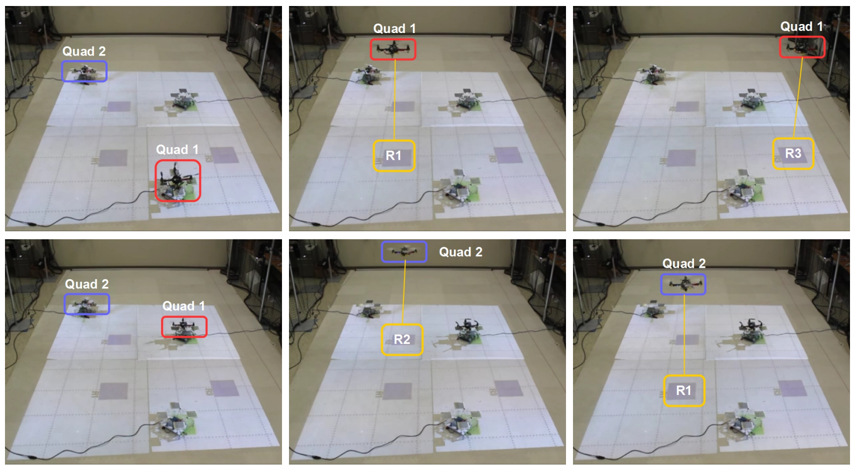


Fig. 7: Screencaps of the first flight loop.

4 Conclusion

The main insight gleaned from this experiment is how to automate a complex persistent surveillance mission specified as a temporal logic formula. The methodology explained herein allows for rapid experimentation following theoretic work using temporal logics. By using the environment partition and transition system generation with time bounds, minimal human input is required to establish an experimental framework for simulating and executing missions. Further, the inclusion of charging stations, whose performance can be modeled using automata, allows for long-term, truly persistent missions involving multiple vehicles not only to be modeled, but to actually be performed in the lab.

The implementation of the persistent surveillance framework required three systems to be integrated together: the BLTL control synthesis algorithm, the vector field generation algorithm, and the quadrotor differential flatness controller. Inevitably, the limitations appear at the interfaces of such systems. For example, the use of multiple vehicles required tuning the controllers quite differently to ensure that the vector field was followed, even though the vehicles are of the same make and model. Regardless of any such complications, because a conservative approach was used, such as using upper bounds on travel time rather than expected travel time, the system met the specifications reliably and predictably. These experiments establish a framework that can be extended to a variety of future work. We are particularly interested in loosening restrictions on mutually exclusive operation so that multiple vehicles may be airborne simultaneously. This would also allow for more complex distributed tasks, such as simultaneously servicing several sites. We are also interested in extending this work to longer mission horizons with more vehicles.

Acknowledgments

This work was supported in part by NSF grant number CNS-1035588, and ONR grant numbers N00014-12-1-1000, MURI N00014-10-10952 and MURI N00014-09-1051. The authors are grateful for this support.

References

1. G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, Oct 1959.
2. P. Toth and D. Vigo. *The vehicle routing problem.* Siam, 2001.
3. N. Michael, E. Stump, and K. Mohta. Persistent surveillance with a team of mavs. In *Proc. of the International Conference on Intelligent Robots and Systems (IROS 11)*, pages 2708–2714. IEEE, 2011.
4. E. Stump and N. Michael. Multi-robot persistent surveillance planning as a vehicle routing problem. In *Proc. of the IEEE Conference on Automation Science and Engineering (CASE)*, pages 569–575. IEEE, 2011.
5. Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

6. S. Smith, J. Tumova, C. Belta, and D. Rus. Optimal Path Planning for Surveillance with Temporal Logic Constraints. *International Journal of Robotics Research*, 30(14):1695–1708, 2011.
7. A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus. Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints. *International Journal of Robotics Research*, 32(8):889–911, 2013.
8. K. Sundar and S. Rathinam. Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots. *Automation Science and Engineering, IEEE Transactions on*, 11(1):287–294, 2014.
9. S. Karaman and E. Frazzoli. Vehicle routing problem with metric temporal logic specifications. In *IEEE Conference on Decision and Control*, pages 3953 – 3958, December 2008.
10. C. Vasile and C. Belta. An Automata-Theoretic Approach to the Vehicle Routing Problem. In *Robotics: Science and Systems Conference (RSS)*, Berkeley, California, USA, July 2014. To Appear.
11. D. Zhou and M. Schwager. Vector field following for quadrotors using differential flatness. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, May 2014. To Appear.
12. S. Jha, E. Clarke, C. Langmead, A. Legay, A. Platzer, and P. Zuliani. A bayesian approach to model checking biological systems. In *Proceedings of the 7th International Conference on Computational Methods in Systems Biology*, CMSB '09, pages 218–234, Berlin, Heidelberg, 2009. Springer-Verlag.
13. I. Tkachev and A. Abate. Formula-free Finite Abstractions for Linear Temporal Verification of Stochastic Hybrid Systems. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, Philadelphia, PA, April 2013.
14. O. Kupferman and M. Vardi. Model checking of safety properties. *Form. Methods Syst. Des.*, 19(3):291–314, October 2001.
15. T. Latvala. Effcient model checking of safety properties. In *10th International SPIN Workshop*, Model Checking Software, pages 74–88. Springer, 2003.
16. C. Belta and L.C.G.J.M. Habets. Controlling a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, 51(11):1749–1759, 2006.
17. E. Aydin Gol and C. Belta. Time-constrained temporal logic control of multi-affine systems. *Nonlinear Analysis: Hybrid Systems*, 10:21–33, 2013.
18. D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.