An Automata-Theoretic Approach to the Vehicle Routing Problem

Cristian-Ioan Vasile
Division of Systems Engineering
Boston University
Brookline, Massachusetts 02446
Email: cvasile@bu.edu

Calin Belta
Division of Systems Engineering
Boston University
Brookline, Massachusetts 02446
Email: cbelta@bu.edu

***Abstract*—We propose a new formulation and algorithms for the Vehicle Routing Problem (VRP). To accommodate persistent surveillance missions, which require executions in infinite time, we define Persistent VRP (P-VRP). The vehicles consume a resource, such as gas or battery charge, which can be replenished when they visit replenish stations. The mission specifications are given as rich, temporal logic statements about the sites, their service durations, and the time intervals in which services should be provided. We define a temporal logic, called Time-Window Temporal Logic (TWTL), whose formulae allow for simple, intuitive descriptions of such specifications. Two different optimization criteria are considered. The first is the infinite-time limit of the duration needed for the completion of a surveillance round. The second penalizes the long-term average of the same quantity. The proposed algorithms, which are based on concepts and tools from formal verification and optimization, generate collision-free motion plans automatically from the temporal logic statements and vehicle characteristics such as maximum operation time and minimum replenish time. Illustrative simulations and experimental trials for a team of quadrotors involved in persistent surveillance missions are included.**

I. Introduction

The Vehicle Routing Problem (VRP) was first formulated in [4] as a distribution problem for gasoline from a terminal to service stations using trucks. The basic formulation of VRP is as follows [12, 9]: given N identical vehicles initially located at a depot, a set of sites, and a distance matrix between the sites and the depot, compute a tour for each vehicle such that each tour starts and ends at the depot, every site is visited exactly once, and the overall travelled distance is minimized. The VRP is known to be NP-hard [6]. Several versions of this problem, which incorporate constraints on the carrying capacity, delivery time frames, and delivery order have been developed [22]. With particular relevance to this paper is the VRP with Time Windows (VRPTW), in which a service time interval (window) is specified for each site [22].

In this paper, we introduce P-VRP, a persistent surveillance version of VRP. The new problem formulation can be seen as a four-fold extension of a relaxed version of VRPTW, in which no restriction is implicitly assumed about the number of visits to the sites. First, we allow for rich, temporal logic constraints on the order in which sites are to be visited. Second, to accommodate persistent surveillance missions, our problem has infinite-time semantics. For example, in our new, user-friendly specification language, called Time-Window Temporal Logic (TWTL), we can describe missions such as "Service sites A, B, and C infinitely often within time windows [2,7], [6,12], and [5,20], respectively. The service times for A, B, and C are 2, 3, and 1, respectively." [1] Third, we incorporate resource constraints. We assume that, while moving in the environment, each vehicle consumes a resource (e.g., battery charge or fuel) proportionally to the time away from a depot. There is an upper limit on the quantity of the resource each vehicle can store. To replenish their reserves, the vehicles need to return to the depots. Finally, to allow for many revisits to a particular location, we explicitly model and deal with the collision avoidance problem.

Our proposed technical approach brings together concepts and tools from automata theory, formal verification, and optimization. Given a specification as a formula of TWTL, we first translate it to a formula in an off-the-shelf temporal logic called syntactically co-safe Linear Temporal Logic (scLTL) [16], and then to a finite state automaton that accepts the satisfying language. This is then composed with finite transition systems modeling the motion of the vehicles in the environment and the charging constraints. In this product automaton, among all the collision-free motion plans that satisfy the specification and the charging constraints, we select an optimal one. We explore two different optimization criteria. The first is the infinite-time limit of the duration needed for the completion of a surveillance round. The second penalizes the long-term average of the same quantity. These criteria lead to NP-complete problems. We impose some additional restrictions to reduce the problems to manageable sizes. We present simulation case studies and experimental trials with a team of quadrotors involved in a temporal logic persistent surveillance mission with deadlines. The quadrotors can automatically land and charge at a set of fixed charging stations.

This work is related to (and inspired from) several recent works that promote the use of temporal logics and formal methods [1] for robot motion planning and control [15, 24, 3, 11, 5, 20, 23]. In particular, [20, 23] consider optimal persistent surveillance problems with temporal logic constraints and

[1] The "classical" VRP constraint that all sites need to be visited exactly once can be easily enforced as a TWTL formula.

optimality guarantees. However, resource constraints are not considered. In addition, the specification language, which is off-the-shelf LTL, does not capture time windows. Resource constraints for the routing problem restricted to one vehicle and the classical setup of servicing all sites (i.e., no temporal logic specifications) are considered in [21].

The closest related work is [10], which contains a mixed integer linear programming formulation of VRP called VRP-MTL. The specifications are given as formulas in a fragment of Metric Temporal Logic (MTL) [14], where the temporal operators can only be applied to atomic propositions or their negations. The durations of the transition between sites are fixed and each site can be visited at most once. Our logic, TWTL, strictly contains the MTL fragment used in [10]. In our approach, a site can be serviced multiple times during a tour if it is required by the specification, and bounds (intervals) on transition durations are allowed. VRL-MTL does not take into account resource constraints related to vehicle movement, such as fuel or battery life, considers a single task over a finite horizon, and optimizes a weighted sum of the distances traveled by the vehicles.

II. Environment and Vehicle Models

For simplicity of presentation, we assume the team is made of N identical vehicles. At the end of the paper, we discuss how this assumption can be relaxed. Let $\mathcal{E} = (Q = \mathcal{S} \cup \mathcal{C}, \Delta, \varpi)$ be a graph environment, where $\mathcal{S}$ is the set of sites and $\mathcal{C}$ is the set of replenish stations or depos. An edge $e \in \Delta \subseteq Q \times Q$ denotes that a vehicle can move between the source and destination of the edge. We assume that the vehicles can deterministically choose to traverse the edges of $\mathcal{E}$, stay at a site for service, or stay docked in a charging station. Each edge has an associated duration given by $\varpi : \Delta \to \mathbb{Z}_{\geq 1}$. We assume that the duration associated to an edge includes the time for obstacle avoidance maneuvers and docking or undocking, if applicable. For now, we assume that this value is the exact time that a vehicle needs to travel the corresponding edge. However, the method developed in this paper also works for the case when this value is an upper bound for the travel time.

We assume that a collision between two vehicles can occur in one of the following three situations: (1) both are at the same node at the same time; (2) both traverse the same edge at the same time (they may start the motion at different times); (3) a vehicle arrives at a node less that t_{col} after the departure of another vehicle from the same node.

Each vehicle has a limited amount of a resource, such as fuel or battery charge, and must regularly return to a replenish station. For simplicity, we assume that the resource is battery charge (level), and we will refer to the replenish stations as charging stations. We use t_{op} to denote the maximum operation time for a vehicle starting with a fully charged battery and t_{ch} to denote the charging time starting with an empty battery. For simplicity, we assume that time is discretized, and all durations (e.g., $\varpi(\Delta)$, t_{col}, t_{op}, t_{ch}) are expressed as an integer multiple of a time interval Δt. Let $\gamma = \frac{t_{ch}}{t_{op}} \geq 1$ be the charge-discharge (integer) ratio.

The *battery state* $b_t(i)$ of vehicle $i \in \{1, \ldots, N\}$ at time $t \in \mathbb{Z}_{\geq 0}$ is discretized such that $b_t(i) \in \{0, \ldots, t_{ch}\}$. The update rule for $b_t(i)$ after d time units is defined as follows:

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

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

At each time, each vehicle may be in one of the following four states: (1) *moving* between sites and charging stations, (2) *servicing* a request at a site, (3) *charging* or (4) *idle* if docked and fully charged. If a vehicle is either moving or servicing a request, we will say that the vehicle is *active*. A time interval such that all vehicles are docked and at least one is charging is called *no flight time* (NFT). A time interval in which all vehicles are idle is called *idle time*. We require that NFTs and idle times are maximal time intervals, i.e. they may not be extended on either side while maintaining their defining property.

For $q \in Q$, we use $\vec{q}$ to denote that a vehicle is moving towards q. Let $\vec{Q} = \{\vec{q} \,|\, q \in Q\}$. A *control policy* for the N vehicle system is a sequence $\mathbf{v} = v_1 v_2 \ldots$ where $v_t \in (Q \cup \vec{Q})^N$ specifies at each time $t \in \mathbb{Z}_{\geq 0}$ and for each vehicle $i \in \{1, \ldots, N\}$ if vehicle 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 vehicle i at time t and the control policy for vehicle i (i.e., the sequence of control values), respectively. Then a transition $(q_1, q_2) \in \Delta$ performed by vehicle 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 $d = \varpi((q_1, q_2))$ is the duration of the transition. Servicing or charging for one time interval (Δt time) by vehicle i at time t corresponds to $v_t(i) = v_{t+1}(i) \in Q$.

For a control policy $\mathbf{v} = v_1 v_2 \ldots$ we define the corresponding output word $\mathbf{o} = o_1 o_2 \ldots$, where $o_t = \{v_t(i) | v_t(i) \in \mathcal{S}, i \in \{1, \ldots, N\}\}$ is the set of all sites occupied by the N vehicles at time $t \in \mathbb{Z}_{\geq 0}$. We use ϵ to denote that no site is occupied. Let $q^{[d]}$ and q^{ω} denote d and infinitely many repetitions of q, respectively.

Example II.1. *An example for the case of $N = 2$ vehicles, 3 sites, and 3 charging stations is shown in Fig. 1. A possible control policy $\mathbf{v}$ for vehicle 1 (blue) and vehicle 2 (red) is:*

$$\begin{aligned} v(1) &= Ch_3^{[1]} \vec{C}^{[3]} C^{[4]} \vec{A}^{[2]}, A^{[3]} \vec{Ch}_1^{[3]}, Ch_1^{[18]} Ch_1^{[54]} \\ &\quad \left(Ch_1^{[1]} \vec{C}^{[3]} C^{[4]} \vec{A}^{[2]} A^{[3]} \vec{Ch}_1^{[3]} Ch_1^{[18]} Ch_1^{[54]} \right)^{\omega} \\ v(2) &= Ch_2^{[1]} Ch_2^{[17]} \vec{B}^{[3]} B^{[3]} \vec{C}^{[4]} C^{[3]} \vec{Ch}_3^{[4]} Ch_3^{[54]} \\ &\quad \left(Ch_3^{[1]} Ch_3^{[17]} \vec{B}^{[3]}, B^{[3]} \vec{C}^{[4]} C^{[3]} \vec{Ch}_3^{[4]} Ch_3^{[54]} \right)^{\omega} \end{aligned} \quad (2)$$

Under control strategy (2), the blue vehicle services sites C and A and the red vehicle services sites B and A infinitely

often. The blue and red vehicles always return to Ch_1 and Ch_3, respectively. The corresponding output word is

$$o = \left(\epsilon^{[4]} C^{[4]} \epsilon^{[2]} A^{[3]} \epsilon^{[8]} B^{[3]} \epsilon^{[4]} C^{[3]} \epsilon^{[58]}\right)^{\omega}.$$

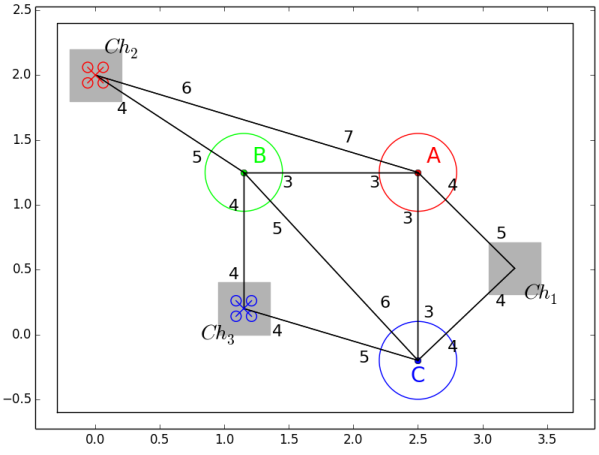


Fig. 1: An environment with 3 sites, $\mathcal{S} = \{A, B, C\}$, and 3 charging stations, $\mathcal{C} = \{Ch_1, Ch_2, Ch_3\}$. The two numbers associated to each edge correspond to the durations for the two directions of the edge, e.g., the durations of edges (B, C) and (C, B) are 5 and 6, respectively. The vehicles, shown in blue and red, start fully charged from the charging stations Ch_3 and Ch_2, respectively. The charging time is $t_{ch} = 60$, the operation time is $t_{op} = 20$ ($\gamma = 3$), and the collision time is $t_{col} = 2$.

III. Specification Language: TWTL

Our goal in this work is to be able to deploy (generate a control policy for) the N vehicles from rich, temporal logic specifications about the service time windows and their durations. In this section, we define a temporal logic, called Time Window Temporal Logic (TWTL), whose semantics is rich enough for a large class of robotic missions. In contrast to off-the-shelf logics such as MTL and BLTL, which can, in principle, be used to specify the same class of tasks, TWTL is also easy to use (a comparison is provided at the end of this section).

Formally, the syntax of TWTL is given defined as:

$$\phi ::= \top \mid s \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid \phi_1 \cdot \phi_2 \mid H^d p \mid [\phi_1]^{[a,b]}$$

where $\top$ is the "true" constant; $s \in \mathcal{S}$ is a site label; $\wedge$, $\vee$, and $\neg$ are the conjunction, disjunction, and negation Boolean operators, respectively; $\cdot$ is the concatenation operator; H^d with $d \in \mathbb{Z}_{\geq 0}$ is the *hold* operator; and $[\,]^{[a,b]}$ with $0 \leq a \leq b$ is the *within* operator.

The semantics of the operators are defined with respect to finite subsequences of an output word $\mathbf{o}$. Let $\mathbf{o}_{t_1,t_2}$ be the subsequence of $\mathbf{o}$ which starts at time $t_1 \geq 0$ and ends at time $t_2 \geq t_1$. Trivially, $\mathbf{o}_{t_1,t_2} \models s$, $s \in \mathcal{S}$, if $o_{t_1} \models s$, where $o_{t_1} \models s$ means $s \in o_{t_1}$. $\mathbf{o}_{t_1,t_2}$ satisfies $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, or $\neg\phi$ if $\mathbf{o}_{t_1,t_2}$ satisfies both formulae, at least one formula, or does not satisfy the formula, respectively. $H^d s$ specifies that $s \in \mathcal{S}$ should be repeated for d time intervals, i.e. $\mathbf{o}_{t_1,t_2} \models H^d s$, $d \leq t_2 - t_1$, if $o_{t_1+i} \models s$ for $i \in \{0, \ldots, d\}$. The *within* operator $[\phi]^{[a,b]}$ bounds the satisfaction of ϕ to the time window $[a, b]$. We define the *time bound* $\|\phi\|$ of a TWTL formula ϕ as follows:

$$\|\phi\| = \begin{cases} 0 & \text{if } \phi \in \{\top, s\} \\ \max(\|\phi_1\|, \|\phi_2\|) & \text{if } \phi \in \{\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2\} \\ \|\phi_1\| & \text{if } \phi = \neg\phi_1 \\ \|\phi_1\| + \|\phi_2\| + 1 & \text{if } \phi = \phi_1 \cdot \phi_2 \\ d & \text{if } \phi = H^d s \\ b - a & \text{if} \phi = [\phi_1]^{[a,b]} \end{cases} \quad (3)$$

Formally, $\mathbf{o}_{t_1,t_2} \models [\phi]^{[a,b]}$ if there is $i \in \{a, \ldots, b - \|\phi\|\}$ such that $\mathbf{o}_{t_1+i,t_2} \models \phi$. Concatenation $\phi_1 \cdot \phi_2$ specifies that first ϕ_1 must be satisfied and then immediately ϕ_2 must be satisfied. Formally $\mathbf{o}_{t_1,t_2} \models \phi_1 \cdot \phi_2$ if there is $i \in \{0, \ldots, \|\phi_1\|\}$ such that $\mathbf{o}_{t_1,i} \models \phi_1$ and $\mathbf{o}_{i+1,t_2} \models \phi_2$ and i has minimum value. An output word $\mathbf{o}$ is said to satisfy a formula ϕ if $\mathbf{o}_{0,\|\phi\|} \models \phi$.

TWTL formulae may be used to specify properties such as: (1) servicing within a deadline: "service A for 2 time units before 10" – $[H^2 A]^{[0,10]}$; (2) servicing within time windows: "service A for 4 time units within [3, 8] and B for 2 time units within [4, 7]" – $[H^4 A]^{[3,8]} \wedge [H^2 B]^{[4,7]}$; (3) servicing in sequence: "service A for 3 time units within [0, 5] and after this service B for 2 time units within [4, 9]" – $[H^3 A]^{[0,5]} \cdot [H^2 B]^{[4,9]}$; (4) enabling conditions: "if A is serviced for 2 time units within 9 time units, then B should be serviced for 3 time units within the same time interval (i.e., within 9 time units)" – $[H^2 A \Rightarrow [H^3 B]^{[2,5]}]^{[0,9]}$, where $\Rightarrow$ denotes implication. Note that Boolean operators characterize parallel execution or alternatives, concatenation serial tasks, while the *within* and *hold* operators define satisfaction bounds.

It is important to note that, by using existing temporal logics, it is difficult to specify missions such the ones exemplified above. For example, in BLTL, the formulae for specifications (2) and (3) above are $\mathbf{F}^{\leq 8-4}\mathbf{G}^{\leq 4} A \wedge \mathbf{F}^{\leq 7-2}\mathbf{G}^{\leq 2} B$ and $\mathbf{F}^{\leq 5-3}(\mathbf{G}^{\leq 3} A \wedge \mathbf{F}^{\leq 9-2+3}\mathbf{G}^{\leq 2} B)$, respectively. In the MTL fragment from [10], properties (2) and (3) translate to $\bigvee_{i=3}^{8-4} \mathbf{G}_{[i,i+4]} A \wedge \bigvee_{i=4}^{7-2} \mathbf{G}_{[i,i+2]} B$ and $\bigvee_{i=0}^{5-3}(\mathbf{G}_{[i,i+3]} A \wedge \bigvee_{j=i+3+4}^{i+3+9-2} \mathbf{G}_{[j,j+2]} B)$, respectively. In the above formulae, $\mathbf{F}$ and $\mathbf{G}$ are the "classical" eventually and always operators. Our experience, which motivates the introduction of TWTL, shows that translating common robotics specifications to these logics is also prone to errors. Another advantage of TWTL is that the concatenation operator is explicitly defined. Under some reasonable assumptions, we can show that concatenation takes linear instead of quadratic time to translate to Finite State Automaton (FSA), because the considered languages are composed of bounded words. In BLTL and MTL, concatenation is expressed in terms of the other operators and this advantage is lost. Also, Finite State Cover Automata (FSCA), which are specifically tailored to the structure of TWTL can lead to a significant decrease in the size of the automata (in some cases by an exponential factor).

IV. P-VRP Formulation

Let $\mathbf{v}$ be a control policy. We say that $\mathbf{v}$ is *feasible* if at each moment in time the N vehicles are pairwise in collision

free states and have non-negative battery states, i.e., $b_t(i) \geq 0$ for all $i \in \{1, \ldots, N\}$ and $t \in \mathbb{Z}_{\geq 0}$.

Definition IV.1 (Persistent surveillance)**.** *A control policy is said to satisfy the persistent surveillance specification $\mathbb{G}\phi$, where ϕ is a TWTL formula, if the generated output word satisfies the TWTL formula ϕ infinitely often and there is no idle time between any two consecutive satisfactions of ϕ.*

Note that, while the satisfaction of $\mathbb{G}\phi$ does not allow for idle time between successive satisfactions of ϕ, there may be no flight time to allow for the vehicles to recharge.

Problem IV.2 (P-VRP Completeness)**.** *Given an environment $\mathcal{E} = (Q = \mathcal{S} \cup \mathcal{C}, \Delta, \varpi)$, N vehicles, operation time t_{op}, charging time t_{ch}, collision time t_{col}, and a TWTL 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.

Example IV.3 (Ex. II.1 revisited)**.** *Consider a mission in which it is required to service site A for 2 time units within $[0, 12]$ and site C for 3 time units within $[0, 9]$. In addition, within $[0, 32]$, site B needs to be serviced for 2 time units followed by either A or C for 2 time units within $[0, 8]$. All the above requirements need to be satisfied infinitely often. The corresponding formula is $\mathbb{G}\phi_{tw}$, where*

$$\phi_{tw} = [H^2 A]^{[0,12]} \wedge [H^2 B [H^2 A \vee C]^{[0,8]}]^{[0,32]} \wedge [H^3 C]^{[0,9]}$$

The control policy from Eqn. (2) satisfies the above persistent surveillance specification. It is easy to note that it is also feasible because at most one vehicle is active at all times and the battery states for both vehicles are always non-negative. For each vehicle, the control policy has a loop ending after a NFT, which is marked in gray. The NFTs ensure that the vehicles start each loop fully charged.

Let $T(k)$ be the start time of the k-th loop and $\Delta T(k) = T(k+1) - T(k)$ be the *loop time*. Let

$$J_1(\mathbf{v}) = \limsup_{k \to \infty} \Delta T(k) \tag{4}$$

and

$$J_2(\mathbf{v}) = \lim_{k \to \infty} \frac{T(k+1)}{k} = \lim_{k \to \infty} \frac{\sum_{i=1}^{k} \Delta T(i)}{k} \tag{5}$$

be two cost functions that penalize the asymptotic upper bound of the loop time and the long-term average loop time, respectively.

Problem IV.4 (Optimality)**.** *Under the same assumptions stated in Prob. IV.2, find a satisfying and feasible control policy that minimizes J_1 (or J_2) if one exists, otherwise report failure.*

Our approach to Probs. IV.2 and IV.4 is inspired from automata-based model checking. The TWTL formula is translated to a syntactically co-safe Linear Temporal Logic (scLTL) [16], and then to a finite state automaton that accepts the satisfying language. This is then composed with finite transition systems modeling the motion of the vehicles in the environment and the charging constraints. The satisfiability and optimality problems are solved on the resulting product automaton.

V. Control Policy

For a finite set Σ, we use $|\Sigma|$, 2^{Σ}, and $P_k(\Sigma)$ to denote its cardinality, power set, and set of k-permutations, respectively. The empty set is denoted by $\emptyset$.

Algorithm 1: Product Automaton

Input: $\mathcal{T}$ – transition system
Input: ϕ – specification as a TWTL formula
Input: N – number of vehicles
Input: t_{col} – collision time
Input: t_{op}, t_{ch} – operation time and charging time

Construct product transition system $\mathcal{T}^N$ for the ME or FC operation mode
Generate charging FSA $\mathcal{A}^{ch}$ with t_{op}, t_{ch} for N vehicles
Construct product $\mathcal{P}^{ch} = \mathcal{T}^N \times \mathcal{A}^{ch}$
Transform ϕ to an $scLTL$ formula ψ
Construct the FSA $\mathcal{A}^{spec}$ corresponding to ψ
Construct product automaton $\mathcal{P} = \mathcal{P}^{ch} \times \mathcal{A}^{spec}$

Algorithm 2: Planning Algorithm – completeness

Input: $\mathcal{T}$ – transition system
Input: ϕ – specification as a TWTL formula
Input: N – number of vehicles
Input: t_{col} – collision time
Input: $q_0 \in P_N(\mathcal{C})$ – initial vehicle locations
Input: t_{op}, t_{ch} – operation time and charging time
Output: $\mathbf{v}$ – control policy

$\mathcal{P} \leftarrow ConstructPA(\mathcal{T}, \phi, N, t_{col}, t_{op}, t_{ch})$
$G = (V, E)$, $V = P_N(\mathcal{C})$, $E = \emptyset$
for $(q_1, q_2) \in V \times V$ **do**
    if there is a satisfying path in $\mathcal{P}$ starting fully charged in q_1 and ending fully charged in q_2 then $E \leftarrow E \cup (q_1, q_2)$ and $control_{\mathcal{P}}(q_1, q_2)$ stores the computed path in $\mathcal{P}$
if *G is acyclic or no cycle is reachable from q_0* **then**
    return Failure
else
    find a cycle q_c and a path q_p to the cycle in G
    return $\beta_{\mathcal{T}^N}(control_{\mathcal{P}}(q_p)(control_{\mathcal{P}}(q_c))^{\omega})$

A. Motion model

We consider two modes of operation: (1) mutually exclusive mode, which assumes that at any given time at most one vehicle is active (i.e., moving or servicing a request), and (2) fully concurrent mode, which does not place any restrictions except that the vehicles must be in collision free states at all times. The mutually exclusive mode of operation has the advantage that it guarantees collision free control policies and also extended overall operation time for the vehicles. This is a good fit for surveillance missions, but may not be desired

for rescue missions, where a parallel search approach may be more effective. Also, as discussed at the end of the section, the complexity of the presented algorithms is lower for the mutually exclusive mode.

The motion of a single vehicle is captured by a weighted transition system $\mathcal{T} = (Q, q_0, \bar{\Delta}, \bar{\varpi}, \Pi, h)$, where $Q = \mathcal{S} \cup \mathcal{C}$ is the set of states, $q_0 \in \mathcal{C}$ is the initial state, $\bar{\Delta} = \Delta \cup \{(q,q) | q \in Q\}$ is the set of transitions, $\bar{\varpi} : \bar{\Delta} \to \mathbb{Z}_{\geq 1}$ is the weight function, $\Pi = \mathcal{S} \cup \{\epsilon\}$ is the alphabet, and $h : Q \to \Pi$ is the labeling function. The weights represent the durations of transitions such that $\bar{\varpi}(q,q') = \varpi(q,q')$, for $q \neq q'$, and $\bar{\varpi}(q,q') = 1$, for $q = q'$. Thus, servicing and docking are modeled as self-loop transitions with duration 1. The labeling function only assigns values to sites, i.e. $h(q) = q$ for $q \in \mathcal{S}$ and $h(q) = \epsilon$ otherwise.

1) Mutually exclusive (ME) operation mode: In order to capture the motion of all vehicles at the same time, we define a mutually-exclusive product transition system (PTS) as a tuple $\mathcal{T}^N = (\tilde{Q}, \tilde{q}_0, \tilde{\Delta}, \tilde{\omega}, \Pi^N, \tilde{h})$. The set of states is defined such that there is at most one active vehicle, $\tilde{Q} = P_N(\mathcal{C}) \cup (\bigcup_{k=1}^{N} \{(q_1, \ldots, q_N) | q_k \in \mathcal{S}, (q_1, \ldots, q_{k-1}, q_{k+1}, \ldots, q_N) \in P_{N-1}(\mathcal{C})\})$. At the initial state, it is assumed that all vehicles are docked, $\tilde{q}_0 \in P_N(\mathcal{C})$. A transition $(q_1, \ldots, q_N) \to (q'_1, \ldots, q'_N) \in \tilde{\Delta}$ if: (1) $q_i = q'_i$, $\forall i$, or (2) $q_k \to q'_k \in \Delta$, $q_i = q'_i$, $\forall i \neq k$ and $q'_k \neq q_j$, $\forall j \neq k$. The weight of a transition is 1 if the two endpoints are the same or equal to the weight of the transition in Δ corresponding to the second case above. The labeling function is defined component-wise, $\tilde{h}(q_1, \ldots, q_N) = (h(q_1), \ldots, h(q_N))$.

2) Fully concurrent (FC) operation mode: We define a similar product transition system (PTS) $\mathcal{T}^N = (\tilde{Q}, \tilde{q}_0, \tilde{\Delta}, \tilde{\omega}, \Pi^N, \tilde{h})$ as before, but in this case we account for simultaneous active vehicles and collisions. The simultaneous motions of the vehicles lead to a synchronization problem. Due to space constraints, we only include an informal description of how this synchronization problem is solved. We split all the edges of the single-vehicle transition system $\mathcal{T}$ into edges of duration 1. We then proceed to compute the full PTS, which captures all possible motions of the N vehicles, using this modified transition system. The last step is to eliminate the states and edges of the PTS that determine collisions according to the description from Sec. II.

Note that we achieve collision avoidance using temporal separation, instead of spatial separation as in the case of geometric approaches such as RRT [19, 18] or PRM [13]. In our case, this is beneficial since it prunes the PTS of undesired states and actually helps lower the computation time in the fully concurrent mode. Also, for the case of quadrotors, temporal separation also avoids undesired aerodynamic effects which may arise due to close proximity of the vehicles. One example is the loss of lift when a quadrotor is directly below another one. These phenomena are somewhat hard to encode in geometric approaches.

B. Charging model

The charging process is modeled as a Finite State Automaton (FSA). Recall that the charging time t_{ch} is an integer multiple of Δt and $\gamma = \frac{t_{ch}}{t_{op}} \in \mathbb{Z}_{\geq 1}$ (see Sec. IV). For the ME operation mode, the charging FSA is $\mathcal{A}^{ch} = (S^{ch}_{\mathcal{A}}, s^{ch}_0, \Sigma^{ch}, \delta^{ch}_{\mathcal{A}}, F^{ch}_{\mathcal{A}})$. $S^{ch}_{\mathcal{A}} = (\{0, \ldots, t_{ch}\})^N$ is the set of states. A state stores the battery states for all vehicles. The initial state is $s^{ch}_0 = (t_{ch}, \ldots, t_{ch})$ and corresponds to all vehicles being fully charged. The alphabet is $\Sigma^{ch} = (\{(i,0) | 1 \leq i \leq N\} \cup \{(0,i) | 1 \leq i \leq N\} \cup \{(i,i) | 0 \leq i \leq N\}) \times D$, where D is the set of the durations of all transition of $\mathcal{T}$. Each triple represents the current and previous active vehicle and the duration of a transition from $\mathcal{T}$. The value 0 for the current or previous active fields indicates that no vehicle is active. By this convention, the three sets of pairs in the definition of the alphabet capture undocking, docking, and moving or servicing performed by vehicle i, respectively. The transition function is defined for two cases: (1) if all robots are recharging ($c = p = 0$), then

$$\delta^{ch}_{\mathcal{A}}((t_1, \ldots, t_N), (c, p, d)) = \\ (min(t_1 + d, t_{ch}), \ldots, min(t_N + d, t_{ch}));$$

(2) if one robot is active ($c > 0$ or $p > 0$), then

$$\delta^{ch}_{\mathcal{A}}((t_1, \ldots, t_N), (c, p, d)) = \\ (min(t_1 + d, t_{ch}), \ldots, t_a - \gamma d, \ldots, min(t_N + d, t_{ch})),$$

where $a = \max(c, p)$. Note that the transition function resembles the charging rule defined in Sec. II. The set of final states $F^{ch}_{\mathcal{A}}$ can be the whole set of states $S^{ch}_{\mathcal{A}}$ if no restrictions on the final battery states are defined. However, we will impose some restrictions on $F^{ch}_{\mathcal{A}}$ later in this section.

For the FC operation mode, we modify the alphabet to $\Sigma^{ch} = \{0,1\}^N \times \{0,1\}^N$, which specifies for each vehicle if it was docked or active in the current and previous time steps, respectively. The transition function must also be adapted. Let $(t'_1, \ldots, t'_N) = \delta^{ch}_{\mathcal{A}}((t_1, \ldots, t_N), ((c_1, \ldots, c_N), (p_1, \ldots, p_N)))$. Then $t'_i = \min(t_i + 1, t_{ch})$ if $c_i = p_i = 0$, or $t'_i = t_i - \gamma$ otherwise, for all $i \in \{1, \ldots N\}$. Note that, in this case, we do not include the durations of transitions in the alphabet, because by construction all the transitions of the PTS have duration one in the FC mode.

C. Specification

To enforce the specification, we encode it as an automaton. We first translate the TWTL formula into a formula of a fragment of LTL, called scLTL [16]. We then use an off-the-shelf tool, such as *scheck* [17], to obtain a FSA $\mathcal{A}^{spec}$ that accepts the language over $2^{\mathcal{S}}$ that satisfies the formula.

Formulas of scLTL use standard Boolean operators and temporal operators such as $\mathbf{X}$ (next), $\mathcal{U}$ (until), and $\mathbf{F}$ (eventually). Since we assume that all durations are given as integers, we only use the $\mathbf{X}$ (next) operator to express the TWTL operators. Specifically, $H^d p := \bigwedge_{i=0}^{d} \mathbf{X}^i p$, $[\phi]^{[a,b]} := \bigvee_{i=a}^{b-\|\phi\|} \mathbf{X}^i \tilde{\phi}$ and $\phi_1 \cdot \phi_2 := \tilde{\phi_1} \wedge \left(\bigvee_{i=0}^{\|\phi_1\|} ((\tilde{\phi_1} \wedge \bigvee_{p \in \mathcal{S}} \mathbf{X}^i p) \implies \mathbf{X}^i \tilde{\phi_2}) \right)$, where $\mathbf{X}^i$, $i \geq 0$, is the iterated $\mathbf{X}$ operator, and $\tilde{\phi}$ is the scLTL version of the TWTL formula ϕ. Note that the size of obtained scLTL formulae is of the order of the size of the TWTL

formula times its time bound, thus yielding formulae which are too long to handle by an operator. Consider the example from Sec. III, where the specification is to "satisfy A for 2 time units before 10". The TWTL is $[H^2 A]^{[0,10]}$, while the corresponding scLTL formula has 24 operands and 75 operators for a total size of 99. Thus, a specification language such as TWTL, which incorporates time bounds in the operators and avoids the explosion of the formulae sizes, becomes necessity, rather than convenience.

D. Completeness

To provide a solution to Problem IV.2, we first define a product automaton that captures all feasible motions of the team that satisfy the specification and the charging constraints. First, we construct the product automaton $\mathcal{P}^{ch} = \mathcal{T}^N \times \mathcal{A}^{ch}$ between the motion model $\mathcal{T}^N$ and the charging FSA $\mathcal{A}^{ch}$. We define it as $\mathcal{P}^{ch} = (S^{ch}_{\mathcal{P}}, s^{ch}_{\mathcal{P}0}, \Delta^{ch}_{\mathcal{P}}, \omega^{ch}_{\mathcal{P}}, \Pi^N, h^{ch}_{\mathcal{P}})$, where $S^{ch}_{\mathcal{P}} = \tilde{Q} \times S^{ch}_{\mathcal{A}}$ is the set of states, $s^{ch}_{\mathcal{P}0} = (q_0, s^{ch}_0)$ is the initial state, $\Delta^{ch}_{\mathcal{P}} \subseteq \tilde{\Delta} \times \delta^{ch}_{\mathcal{A}}$ is the transition function, $\omega^{ch}_{\mathcal{P}} : \Delta^{ch}_{\mathcal{P}} \to \mathbb{Z}_{\geq 1}$ is the weight function, and $h^{ch}_{\mathcal{P}} : S^{ch}_{\mathcal{P}} \to \Pi^N$ is the labeling function. $\omega^{ch}_{\mathcal{P}}$ and $h^{ch}_{\mathcal{P}}$ are inherited from the transition system $\mathcal{T}^N$, i.e. for all $(q,t) \in S^{ch}_{\mathcal{P}}$ and $((q,t),(q',t')) \in \Delta^{ch}_{\mathcal{P}}$ we have $\omega^{ch}_{\mathcal{P}}((q,t),(q',t')) = \tilde{\omega}(q,q')$ and $h^{ch}_{\mathcal{P}}(q,t) = \tilde{h}(q)$.

In the ME mode, a transition $(\tilde{q},t) \to (\tilde{q'},t')$ is in $\Delta^{ch}_{\mathcal{P}}$ if $\tilde{q} \to \tilde{q'} \in \tilde{\Delta}$, $t \to^{(c,p,d)} t'$, $d = \tilde{\omega}(q,q')$ and c and p are the indices of the active vehicle in states q' and q, respectively. If all vehicles are charging at state q or q', then $p = 0$ or $c = 0$, respectively. For the FC mode, a transition $(\tilde{q},t) \to (\tilde{q'},t')$ is in $\Delta^{ch}_{\mathcal{P}}$ if $\tilde{q} \to \tilde{q'} \in \tilde{\Delta}$ and $t \to^{(c,p)} t'$, where $c = (c_1, \ldots, c_N)$ and $p = (p_1, \ldots, p_N)$ specify for each vehicle $i \in \{1, \ldots N\}$ if it is active in states q' and q, respectively.

Second, we construct the product automaton $\mathcal{P} = \mathcal{P}^{ch} \times \mathcal{A}^{spec}$. This is defined as $\mathcal{P} = (S_{\mathcal{P}}, s_0, \delta_{\mathcal{P}}, \omega_{\mathcal{P}}, F_{\mathcal{P}})$, where $S_{\mathcal{P}} = S^{ch}_{\mathcal{P}} \times S^{spec}_{\mathcal{A}}$ is the set of states, $s_0 = ((q_0, s^{ch}_0), s^{spec}_0)$ is the initial state, $\delta_{\mathcal{P}} \subseteq \Delta^{ch}_{\mathcal{P}} \times \delta^{spec}_{\mathcal{A}}$ is the transition function, $\omega_{\mathcal{P}} : \delta_{\mathcal{P}} \to \mathbb{Z}_{\geq 1}$ is the weight function and $F_{\mathcal{P}} = S^{ch}_{\mathcal{P}} \times F^{spec}_{\mathcal{A}}$ is the set of final states. A transition $(p,s) \to (p',s) \in \Delta_{\mathcal{P}}$ if $p \to p' \in \Delta^{ch}_{\mathcal{P}}$ and $s \to^{\sigma} s'$. In the ME mode, $\sigma = \epsilon^{[d-1]} h^{ch}_{\mathcal{P}}(q'_c)$ if $c > 0$ and $\sigma = \epsilon^{[d]}$ otherwise, where c is the index of the active vehicle in p' and $d = \omega^{ch}_{\mathcal{P}}((p,s),(p',s'))$ is the duration of the transition. For the FC mode, $\sigma = \{h(q'_i) | i \in \{1, \ldots, N\}\} \in 2^{\Pi}$. As before, the weight function is inherited from the product $\mathcal{P}^{ch}$, i.e. for every $((p,s),(p',s')) \in \Delta_{\mathcal{P}}$ we have $\omega_{\mathcal{P}}((p,s),(p',s')) = \omega^{ch}_{\mathcal{P}}(p,p')$.

The algorithm to compute the product automaton $\mathcal{P}$ is outlined in Alg. 1. A feasible and satisfying control policy is computed in Alg. 2 as a projection on $\mathcal{T}^N$ of a path from $\mathcal{P}$. In Alg. 2, $\beta_{\mathcal{T}}$ denotes the canonical projection on the $\mathcal{T}$-component of the product.

We now show that Alg. 2 produces a feasible and satisfying control policy if one exists, thus solving Prob. IV.2. The following statements are true for both modes of operation (ME and FC). The only difference is in the construction of the product automaton $\mathcal{P}$, line 1 in Alg. 2.

In the following, for two vectors b and b', $b' \geq b$ if the relationship holds component-wise. The following proposition holds trivially.

Proposition V.1. *Let* $\mathbf{v}$ *be a feasible control policy starting with an initial charging state* b*. Then* $\mathbf{v}$ *is a feasible control policy starting with any initial battery state* $b' \geq b$*.*

Theorem V.2. *Algorithm 2 is complete.*

Proof: Let q_0 be the initial state of the N vehicle system. First, we reduce the problem using Prop. V.1, which implies that if a control policy $\mathbf{v}$ is feasible then we can construct another control policy from it by appending at the end of each loop a NFT such that every loop starts with the vehicles fully charged. Thus, in order to asses feasibility we only need to check reachability between states where all vehicles are docked and fully charged. Consider the graph $G = (V = P_N(\mathcal{C}), E)$ from Alg. 2. The existence of a control policy is equivalent with the existence of an infinite path in G. This in turn implies that there must be a cycle in G reachable from the initial state q_0. If we assume that there is no such cycle, then all paths starting at q_0 are finite, which implies that no control policy exists. It follows that Alg. 2 is complete. ■

E. Optimality

In Sec. V-D, we showed that if Prob. IV.2 admits a solution, then there is a feasible control policy which has a prefix-suffix structure that can be computed on a finite graph. In this section, we will establish the same result for the optimal version of the problem (Prob. IV.4) corresponding to the two cost functions J_1 and J_2.

Let $G^{opt} = (V, E, w)$ be a weighted graph, where $V = S^{ch}_{\mathcal{P}}$ is the vertex set. As in Alg. 2, we proceed to construct the edge set E such that $(q,q') \in E$ if there is a satisfying path in $\mathcal{P}$ starting at (q, s^{spec}_0) and ending at (q', s^{spec}_f), $s^{spec}_f \in F^{spec}$. The weight of $w(q,q')$ is equal to the minimum loop time. Note that loops that are not minimal can be replaced by the minimal ones to decrease the overall cost. To minimize J_1, a cycle in G^{opt} with minimum maximum weight must be computed, because the objective is to minimize the maximum loop time of any loop that is repeated infinitely often. The J_2 criterion is attained by a cycle in G^{opt} of minimum average weight as shown by Prop. V.3.

Proposition V.3. *Let* $G = (V, E, w)$ *be a strongly connected graph with possible self-loops and a weight map* $w : E \to \mathbb{R}_+$*. Then there is a path* $\mathbf{v}^* = v_1, \ldots v_p (v_{p+1}, \ldots v_{p+s})^{\omega}$ *that minimizes* J_2 *and* $J_2(\mathbf{v}^*) = \frac{1}{s}(w(v_{p+s}, v_{p+1}) + \sum_{i=1}^{s-1} w(v_{p+i}, v_{p+i+1}))$.

Proof: It is easy to see that ignoring any finite prefix of $\mathbf{v}$ does not change the value of J_2. Let $\mathbf{c}_s$ be the minimum average weight cycle in G and $\mathbf{v}$ be an infinite path. Since $\mathbf{v}$ is infinite and V is finite it follows that there is a node $v_{inf} \in V$ such that v_{inf} appears infinitely often in $\mathbf{v}$. Any finite sub-sequence of $\mathbf{v}$ delimited by v_{inf} defines a cycle. However, since $\mathbf{c}_s$ has minimum average weight it follows that each cycle in $\mathbf{v}$ delimited by v_{inf} has a greater cost that $\mathbf{v_s}$. This in turn implies that $J_2(v) \geq J_2((v_s)^{\omega})$. ■

Remark V.4. *Finding the minimum average weight cycle is NP-complete. The reduction can be made from the Hamiltonian cycle problem. Therefore, we have to impose some additional restrictions on the control policies in order to reduce G^{opt} to a manageable size.*

F. Complexity

The complexities of Alg. 1 are different for the ME and FC modes. The construction of the product transition system is $\mathcal{O}\left(\frac{|\mathcal{C}|!}{(N-|\mathcal{C}|)!} + N\,|R|\,\frac{|\mathcal{C}|!}{(N-|\mathcal{C}|)!} + N\,|\Delta|\right)$ for the ME mode and $\mathcal{O}\left(\left(\left(|Q| + |\Delta|\right)d_{max}t_{col}\right)^N\right)$ for the FC mode, where d_{max} is the maximum duration of an edge in Δ. Constructing the charging FSA takes $\mathcal{O}\left(t_{ch}^N(N^2 d_{max})\right)$ and $\mathcal{O}\left(t_{ch}^N 2^N\right)$ in ME and FC modes, respectively. We describe the complexity of the next steps of Alg. 1 in a unified manner. However, the actual complexity differs between modes for steps 3 and 6, because they depend on the size of the product transition system and the charging FSA. The complexity of these steps are as follows: $\mathcal{O}\left(\left|\tilde{Q}\right|\,|S_{\mathcal{A}}^{ch}| + \left|\tilde{\Delta}\right|\,|\delta_{\mathcal{A}}^{ch}|\right)$ for constructing the first product, $\mathcal{O}\left(\|\phi\|\,|\phi|\right)$ for converting the TWTL formula ϕ to scLTL formula ψ, where $|\phi|$ is the length of the formula (number of operators and propositions), $\mathcal{O}(2^S 2^{2^{|\psi|}})$ for converting the scLTL formula to an FSA using *scheck* and $\mathcal{O}\left(\left|S_{\mathcal{P}}^{ch}\right|\,|S_{\mathcal{A}}^{spec}| + \left|\delta_{\mathcal{P}}^{ch}\right|\,|\delta_{\mathcal{A}}^{spec}|\right)$ for the final product automaton.

The complexity of Alg. 2 is $\mathcal{O}\left(\frac{|\mathcal{C}|!}{(N-|\mathcal{C}|)!}(S_{\mathcal{P}} + \delta_{\mathcal{P}})\right)$ for constructing the graph $G = (V, E)$ and $\mathcal{O}(V + E)$ to test if there is a reachable cycle from the initial state. Obtaining an optimal solution is NP-complete and therefore exponential in $|V|$.

It is not surprising that the proposed algorithms have exponential complexity, because the VRP problem itself is NP-hard. However, the one outstanding question is how our approach compares to a MILP formulation in terms of scalability w.r.t. the number of vehicles N. The automata-based approach is well suited for the persistent VRP problem because it decreases the worst-case complexity over a MILP implementation. In our approach, we compute a product automaton once and from it we can compute control policies for loops. We then solve an NP-complete problem on the one-loop reachability graph, whose vertex set is polynomial in the number of robots N. Thus, the overall procedure has worst-case complexity $O(2^N + N^{k+1}2^N)$, where k is the fixed difference between the number of depos and robots. On the other hand, a MILP approach does not reuse previous computation and redundant operations may be performed. As such, the worst-case complexity is $O(2^N + N^{2k}2^N)$. This analysis only considers N as a variable (the other parameters are fixed) and that the robots are identical. If we lift the latter assumption, the difference in complexity becomes even greater, because the size of the vertex set of the one-loop reachability graph becomes factorial in N. Thus, the automata-based approach has complexity $O(2^N + N(N+k)!2^N)$ and MILP has $O(2^N + (N+k)!^2 2^N)$. In practice, a MILP approach may be faster in computing a solution for a single loop, but since we need to perform the operation repeatedly the automata approach may be faster overall. Also, encoding the whole problem as a MILP program leads to 2-EXPTIME(N) complexity.

G. Generalizations

The presented framework can easily be modified to account for differences among vehicles, with respect to both motion and replenishing models. Different motion models can be specified as different individual transition systems $\{\mathcal{T}_i\}_1^N$, which can be used to construct the product transition system $\mathcal{T}^N$. The resource model can also be customized with vehicle specific charging and operation times, each satisfying the assumptions from Sec. V-B. The framework can also be minimally modified to support the case when $\frac{t_{op}}{t_{ch}} \in \mathbb{Z}_{\geq 1}$, e.g., when the resource is fuel. In this case, t_{ch} and t_{op} are interchanged in the construction of the state set of the charging FSA $\mathcal{A}^{ch}$ and the inverse of γ is used in the update rule.

Also, the proposed algorithms can be used for the case when the weights of the edges in Δ are upper bounds for travel times, rather than fixed durations. In this case, worst-case feasible control policies are computed off-line, i.e. as described above, and replanning is performed on-line when the actual transition durations become available.

VI. Implementation, Results, and Experimental Validation

We implemented the algorithms developed in this paper in a software tool that takes as input an environment topology (i.e., the positions of the sites and charging stations), the durations of the motions, the operation, charging times, and collision times, and a mission specification in the form $\mathbb{G}\phi$, where ϕ is a TWTL formula. The output is a vehicle control policy. The tool, which was implemented in Python2.7, uses the LOMAP [23] and networkx [8] packages to manipulate and process automata. Also, *scheck* [17] is used to translate to FSA the scLTL formulae obtained from the TWTL formulae. The tool has an input-output graphical user interface (Fig. 1 was generated using the tool).

The tool, running on a Linux system with a 2.1 GHz processor and 32GB memory, was used to generate control policies for the case study presented in Examples II.1 and IV.3. The TWTL formula ϕ_{tw} was translated to an FSA with 1468 states and 5845 transitions. In the ME mode, the construction of $\mathcal{T}^N$, $\mathcal{A}^{ch}$, $\mathcal{P}^{ch}$, and $\mathcal{P}$ took 1.7 msec, 491 msec, 13.5 sec, and 16 sec to compute. Their sizes were 24, 3721, 89304, 75538 states and 108, 127734, 332328, 263144 transitions, respectively. The test for feasibility took 11 sec to execute. Note that the size of the final product automaton $\mathcal{P}$ is not larger than the size of $\mathcal{P}^{ch}$. This is due to the implementation of the construction of $\mathcal{P}$, which contains only the states reachable from the initial ones.

A ME control policy is given in Eqn. 2 from Ex. II.1. The control policy is feasible and satisfies the specification. Furthermore, it is also optimal with respect to both J_1 and J_2 with the assumption that vehicles start each loop fully

charged. Without this assumption, the optimization problem would have to be solved on a graph with 89304 vertices, which is intractable (see Sec. V-E).

Using the FC mode and the same setup, but with $t_{op} = 20$ and $t_{ch} = 40$, the construction of $\mathcal{T}^N$, $\mathcal{A}^{ch}$, $\mathcal{P}^{ch}$, and $\mathcal{P}$ took 662 msec, 387 msec, 25.8 min and 35.2 min to compute. Their sizes were 3066, 1681, 5153946, 6487656 states and 5200, 24964, 7942452, 9669808 transitions, respectively. Feasibility was established in 6.65 min. A feasible and satisfying control policy for the FC mode is

$$\begin{aligned} v^{FC}(1) &= \Big(Ch_3^{[2]}\vec{Ch}_3^{[3]}C^{[4]}\vec{A}^{[2]}A^{[3]}\vec{Ch}_2^{[6]}Ch_2^{[40]} \\ &\quad Ch_2^{[1]}\vec{Ch}_2^{[3]}B^{[2]}\vec{Ch}_3^{[3]}Ch_3^{[49]}\Big)^{\omega} \\ v^{FC}(2) &= \Big(Ch_2^{[1]}\vec{Ch}_2^{[3]}B^{[2]}\vec{Ch}_3^{[3]}Ch_3^{[49]} \\ &\quad Ch_3^{[2]}\vec{Ch}_3^{[3]}C^{[4]}\vec{A}^{[2]}A^{[3]}\vec{Ch}_2^{[6]}Ch_2^{[40]}\Big)^{\omega} \end{aligned}$$

and the corresponding output word is

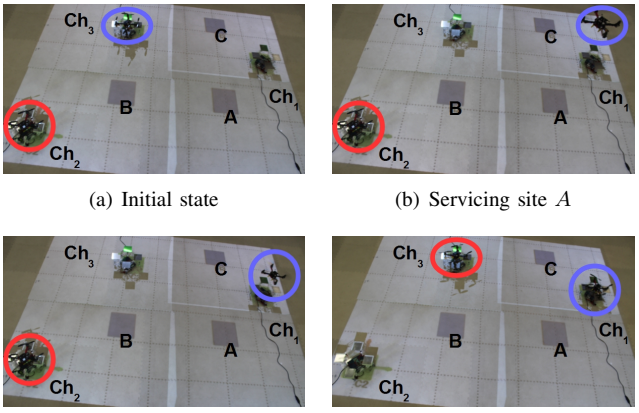
$$o = \left(\epsilon^{[3]}B^{[2]}\{B,C\}^{[1]}C^{[3]}\epsilon^{[2]}A^{[3]}\epsilon^{[46]}\right)^{\omega},$$

where $\{B,C\}^{[1]}$ indicates that both sites B and C are occupied at the same time.

The above case study was also implemented in our aerial vehicle experimental setup, which consists of a team of quadrotors flying autonomously in an indoor space equipped with a motion capture system, short-throw projectors that generate images on the floor, and fully automatic charging stations that can detect the presence of a vehicle and its charging level (see Figs. 3 and 2). To generate transitions among sites and charging stations, the 3D space was partitioned into small rectangular regions. Using the framework developed in [2], vector fields were designed in each rectangle to guarantee safe transitions between adjacent rectangles and stabilization to the center of a rectangle (i.e., for servicing a site, which corresponds to hover). The tool developed in [7] was used to determine the (upper bounds for the) durations of the transitions. The durations of the landing and take-off maneuvers at the charging stations were included in the durations of the transitions to and from the charging stations. Quadrotor feedback control laws were generated to follow the designed vector fields by using the framework developed in [25].

We used the same setup as described in Ex. II.1 (Fig. 1) with the specification from Ex. IV.3. We consider the MC mode and a time interval Δt of 6 sec. Four snapshots from a successful experimental trial are shown in Fig. 2. Consider the loop starting at Ch_3 and Ch_2 and ending at Ch_1 and Ch_3, respectively.

In this loop, the blue quadrotor visits site C, services C for at least 18 sec, visits site A, services A for at least 12 sec, and lands at Ch_1. After the first blue quadrotor lands, the red one takes-off and visits site B, services B for at least 12 sec, visits site C, services C for 12 sec, and lands at Ch_3. The actual transition and servicing durations were 21.47, 18.0, 5.55, 12.0, 23.9, 23.08, 12.0, 28.19, 12.01, 13.72 (all in sec and in the order described above). These durations are bounded above by the estimated durations that were used to compute the control policy, which were 24, 18, 18, 12, 24, 24, 12, 30, 12, 30. Note that specification ϕ_{tw} is satisfied, because: (1) A is serviced in 57.01sec, before its deadline of 72 sec; (2) C is serviced in 38.47 sec, before its deadline of 54 sec; (3) B followed by C are serviced in 156.2 sec, before the deadline of 192 sec; and (4) C is serviced in 42.2 sec after B, before the deadline of 48 sec.



(a) Initial state (b) Servicing site A

(c) Docking at Ch_1 (d) State at the end of the loop

Fig. 2: Two quadrotors in an environment with three sites and three charging stations. Fig. 2(a): the quadrotors are fully charged and docked at the start of the mission. Fig. 2(b): the blue quadrotor is servicing site A, while the red quadrotor is still docked at charging station Ch_2. The docking procedure is shown in Fig. 2(c). The blue quadrobot attempts to land on charging station Ch_1. At the end of the first loop (Fig. 2(d)), the quadrotors are docked at Ch_1 and Ch_3.

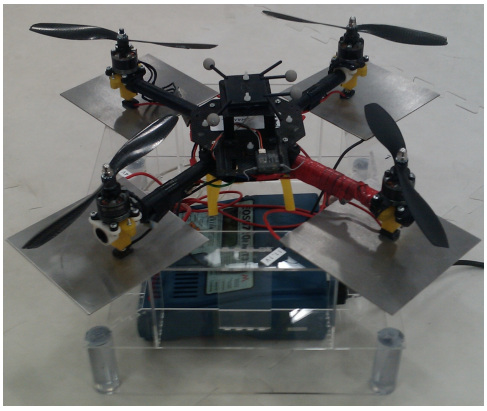


Fig. 3: Quadrotor docked at a charging station.

Acknowledgments

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

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

[2] C. Belta and L.C.G.J.M. Habets. Control of a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, 51(11):1749 – 1759, 2006.

[3] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *Robotics and Automation (ICRA), IEEE International Conference on*, pages 2689–2696. IEEE, 2010.

[4] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, Oct 1959.

[5] Xu Chu Ding, Marius Kloetzer, Yushan Chen, and Calin Belta. Formal Methods for Automatic Deployment of Robotic Teams. *IEEE Robotics and Automation Magazine*, 18:75–86, 2011.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractibility: a Guide to Theory of NP-completeness*. W.H. Freeman Co, New York, 1979.

[7] E. Aydin Gol and C. Belta. Time-constrained temporal logic control of multi-affine systems. *Nonlinear Analysis: Hybrid Systems*, 10:21–23, 2013.

[8] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[9] J.C.Beck, P.Prosser, and E.Selensky. Vehicle Routing and Job Shop Scheduling: What's the difference? In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, Trento, Italy, Jun 2003.

[10] 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.

[11] S. Karaman and E. Frazzoli. Sampling-based Motion Planning with Deterministic μ-Calculus Specifications. In *IEEE Conference on Decision and Control (CDC)*, Shanghai, China, December 2009.

[12] S. Karaman and E. Frazzoli. Linear temporal logic vehicle routing with applications to multi-UAV mission planning. *International Journal of Robust and Nonlinear Control*, 21(12):1372–1395, 2011.

[13] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[14] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. URL http://dx.doi.org/10.1007/BF01995674.

[15] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's Waldo? Sensor-based temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3116–3121, 2007.

[16] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Form. Methods Syst. Des.*, 19(3): 291–314, October 2001. ISSN 0925-9856.

[17] T. Latvala. Effcient model checking of safety properties. In *10th International SPIN Workshop*, Model Checking Software, pages 74–88. Springer, 2003.

[18] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[19] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *IEEE International Conference on Robotics and Automation*, pages 473–479, 1999.

[20] Stephen Smith, Jana Tumova, Calin Belta, and Daniela Rus. Optimal Path Planning for Surveillance with Temporal Logic Constraints. *International Journal of Robotics Research*, 30(14):1695–1708, 2011.

[21] 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.

[22] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-498-2.

[23] 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.

[24] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding Horizon Temporal Logic Planning for Dynamical Systems. In *Conference on Decision and Control (CDC) 2009*, pages 5997 –6004, 2009.

[25] D. Zhou and M. Schwager. Vector field following for quadrotors using differential flatness. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2014.