Specification-Based Maneuvering of Quadcopters Through Hoops

Christopher Banks, Kyle Slovak, Samuel Coogan and Magnus Egerstedt

***Abstract*— In this paper, we study the problem of navigating quadcopters through a sequence of hoops. The specification may be given directly or indirectly via a linear temporal logic (LTL) formula. We approach this problem in three phases. First, we introduce a planner that generates a path through a given sequence of hoops. Second, we augment our planner to leverage a given specification in linear temporal logic (LTL) and generate a sequence that satisfies this specification. Third, we implement cross-entropy optimization on this planner to enhance trajectory performance where quadcopter trajectories are modified within the solution space to optimize over a cost function. We implement this planner as a novel interaction modality between users and quadcopters on the Robotarium. Simulation and experimental results are provided.**

I. Introduction

Emerging unmanned vehicle applications such as farming [1] and reconnaissance [2] require navigating through a sequence of waypoints. With such applications, multiple objectives must be defined throughout the operation period of the vehicle. For example, a robot tasked with surveillance may be required to visit certain locations within a targeted area, report key information to users and periodically charge itself if its battery decreases below a threshold level. Despite a wealth of work on point-to-point motion planning[3], [4], [5], it is still challenging to provide end-to-end solutions for this waypoint problem on an actual platform that allows for reconfigurable waypoints and expressive task formulas.

Motivated by these applications and challenges, in this paper, we study the problem of flying quadcopters through a sequence of hoops. This problem is dynamically challenging and using hoops provides a rich set of specifications for creating end-to-end planners. We first consider the problem using a set of oriented hoops, meaning they have a front and a back. The hoop flying problem can then be solved by a user directly specifying a desired sequence of hoops. We develop a sequence-of-hoops algorithm that ensures a quadcopter flies through this exact sequence when given as hoop/direction pairs by an external user. This paper also considers use cases that involve complex tasks where explicit sequences may be difficult to create.

For complex tasks, linear temporal logic formula can be used to characterize hoops rather than specifying them directly. For example, a user may specify "visit $hoop_0$ or $hoop_1$ before $hoop_2$" which can be represented by the LTL formula $\zeta = \neg hoop_2 \text{ U } (hoop_0 \vee hoop_1) \wedge \Diamond hoop_2$ or a user may indicate "avoid $hoop_1$ at all times" which can be captured in the formula $\zeta = \Box\neg hoop_1$. Work in [6] proposed a sampling-based technique for agents to satisfy a global linear temporal logic specification, and [7] utilized a hybrid controller approach to design continuous trajectories using the dynamics of an autonomous system to satisfy LTL specifications. In addition to these prior works, we address the issue of autonomous systems satisfying global LTL specifications; however, our algorithm identifies more closely with the latter work as we consider the continuous dynamics of quadcopters in addition to considering tasks specified in linear temporal logic. Moreover, given a LTL specification, we model it as a Nondeterministic Büchi Automaton (NBA) and find a satisfying run [8]. The satisfying run corresponds to an infinite sequence of hoops that we parse through the planner as inputs. The planner then defines a trajectory via hand-picked control points around each hoop in the continuous space for quadcopters. We leverage the differentially flat dynamics of quadcopters [9], [10] to generate continuous trajectories using spline interpolation. With a solution to generating feasible trajectories, we turn to optimality.

This work follows [11] and [12] in applying cross-entropy optimization. We use cross entropy as a way to avoid discretizing the action space of the quadcopters, thus reducing computational complexity by not generating a transition system for the quadcopters. Cross-entropy optimization also improves upon feasible trajectories by minimizing over a user defined cost function.

A. Contributions

Our work contributes to the application and development of controllers and algorithms designed to provide abstraction-free planning methods for autonomous vehicles subject to complex specifications given as a temporal logic formula. The main contributions of this paper include: i) an end-to-end solution to the hoop flying problem via a planner given any explicitly labeled sequence of oriented hoops; ii) we utilize linear temporal logic without constructing a finite abstraction of the dynamics of the autonomous system to augment the expressive capabilities of our planner; iii) and we use cross-entropy optimization on nominal trajectories to optimize via user provided cost functions.

We provide this planner as a novel framework for external users to interact with quadcopters in the Robotarium, a remotely accessible multi-robot testbed housed at the Georgia Institute of Technology [13]. To date, the Robotarium has only supported ground robots for remote-access experimentation due to the ground robots ability to be easily

This work is supported in part by the National Science Foundation through the Graduate Research Fellowship grant #DGE-1650044 and through grant #1836932, and in part through the grant ARL DCIST CRA W911NF-17-2- 0181.

All authors are affiliated with Georgia Institute of Technology, Atlanta, GA 30332, USA, {cbanks34, kgslovak, sam.coogan, magnus}@gatech.edu

accessed and controlled at different levels of abstraction (trajectories, paths, behaviors, way points). For aerial robots, these abstractions are not as clear and this paper provides such an interaction modality, i.e., users can engage with the quadcopters at the level of discrete locations (hoops) and mission specifications (LTL). As such, the contribution in this paper should be understood partially as a technical development, partially as an approach to making quadcopters fly through hoops, and partially as providing a novel interaction modality for users engaging with the Robotarium.

B. Paper Overview

The paper proceeds in the following manner. We present a brief overview of the quadcopter model and controller design in Section II. We develop the mathematical foundation for the hoop flying problem and present three scenarios and approaches in Section III. Finally, we discuss experimental results in Section IV.

II. Quadcopter Model and Controller

In this section, we give a brief overview of quadcopter dynamics and the controller used in experiments.

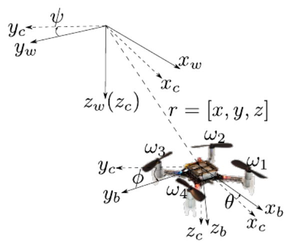


Fig. 1. A quadcopter with respect to the world frame (F_w), intermediate frame (F_c) and body frame (F_b). Four motors ($\omega_{1:4}$) produce torques and thrust for the system.

The rotation matrix, $R(\epsilon)$, that translates between the world frame (F_w) and body frame (F_b) is given by

$$R(\epsilon) = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{1}$$

where $s\theta$ and $c\theta$ stand for $\sin(\theta)$ and $\cos(\theta)$, respectively. The angles θ, ψ, and ϕ are the angles between the axes of the quadcopter in the body frame and the axes of the world frame. The input (μ) to the system consists of $\mu = [f_z, \omega_{bw}^T]^T$ with f_z as the thrust and $\omega_{bw} = [\omega_x, \omega_y, \omega_z]^T$ as the body rotational rates of the quadcopter in the world frame. We use the quadcopter model from [14] to describe the dynamics that generate trajectories for quadcopters:

$$\ddot{r} = g z_w + \frac{1}{m} R(\epsilon) z_w f_z \tag{2}$$

$$\dot{\epsilon} = \Gamma(\epsilon)\omega_{bw} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi sc\theta & c\phi sc\theta \end{bmatrix} \omega_{bw}, \tag{3}$$

where $z_w = [0\,0\,1]^T$ is the z-direction vector for force in F_w and $sc\theta$ and $t\theta$ are $\sec(\theta)$ and $\tan(\theta)$, respectively. The position of the center of mass (r) in the world frame (F_w) is $r = [r_x, r_y, r_z]^T$; m, and g are the mass and acceleration of gravity, respectively. We represent the Euler angles as $\epsilon = [\phi, \theta, \psi]^T$, and $\Gamma(\epsilon)$ is the transformation matrix from body rotational rates in F_b to euler angles in F_w. These dynamics give the 12-dimensional state (ξ) of the quadcopters where $\xi = [r^T, \dot{r}^T, \theta, \phi, \psi, \omega_{bw}^T]^T$ and input (μ).

A. Controller

We define desired trajectories (η_d) as parametric curves in $\mathbb{R}^3$ that are three-times differentiable such that $\eta_d(t) \in C^3$. The state (ξ) and inputs (μ) are generated from trajectories using differential flatness [10], [15] to develop the controller. This allows us to represent the entire state vector (ξ) of the quadcopter and its inputs (μ) as algebraic functions of the outputs. These outputs are called flat outputs, and are selected as $\eta = [r^T, \ \psi]^T$. Through differential flatness, we control the quadcopters using a feedforward term μ_{ff} adapted from the differentially flat outputs and a feedback term μ_{fb}. The μ_{ff} is derived by inverting (2) and (3) to get the feedforward thrust ($f_{z,ff}$) and body rotation rates ($\omega_{bw,ff}$). From [9],

$$f_{z,ff} = \gamma_(\eta, \dot{\eta}, \ddot{\eta}) = -m||\ddot{r} - g z_w||$$

$$\omega_{bw,ff} = \Gamma(\epsilon)^{-1}\dot{\epsilon} = \begin{bmatrix} 1 & 0 & -s\theta_d \\ 0 & c\phi_d & s\phi_d c\theta_d \\ 0 & -s\phi_d & c\phi_d c\theta_d \end{bmatrix} \dot{\epsilon},$$

where $\theta_d = atan2(\beta_a, \beta_b), \phi_d = atan2(\beta_c, \sqrt{\beta_a^2 + \beta_b^2})$ and the d stands for desired. The functions β_a, β_b, and β_c are defined as: $\beta_a = -\ddot{x}_d \cos(\psi_d) - \ddot{y}_d \sin(\psi_d)$, $\beta_b = -\ddot{z}_d + g$ and $\beta_c = -\ddot{x}_d \sin(\psi_d) + \ddot{y}_d \cos(\psi_d)$.

For the feedback term μ_{fb}, the feedback thrust ($f_{z,fb}$) and body rotation rates ($\omega_{bw,fb}$) terms take the form,

$$f_{z,fb} = K_p \langle R(\epsilon) z_w, r_d - r \rangle + K_d \langle R(\epsilon) z_w, \dot{r}_d - \dot{r} \rangle,$$

$$\omega_{bw,fb} = K_p \begin{bmatrix} \phi_d - \phi \\ \theta_d - \theta \\ \psi_d - \psi \end{bmatrix} + K_d \begin{bmatrix} \dot{\phi}_d - \dot{\phi} \\ \dot{\theta}_d - \dot{\theta} \\ \dot{\psi}_d - \dot{\psi} \end{bmatrix} + K_q \begin{bmatrix} y_d - y \\ x_d - x \\ 0 \end{bmatrix}$$

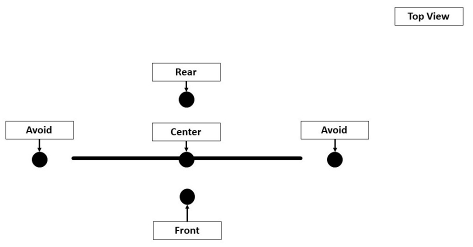
with gains K_p, K_d, and $K_q \in \mathbb{R}_{>0}$ where our total input to the system is $\mu = \mu_{ff} + \mu_{fb}$. We next define the hoop flying problem and our proposed approach.

III. Problem Formulation and Approach

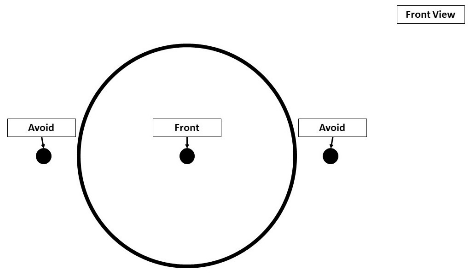
Given a quadcopter with dynamics described in Section II, we are interested in developing a fully autonomous planner capable of navigating through suspended hoops. We define a hoop in $\mathbb{R}^3$, with the following definition:

Definition 1: We define a hoop set $\mathcal{H}$=$\{\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_n\}$ according to $E_i(r) = \frac{(r_x - x_i)^2}{a^2} + \frac{(r_y - y_i)^2}{b^2} + \frac{(r_z - z_i)^2}{c^2}$ and $\mathcal{H}_i = \{r \in \mathbb{R}^3 \,|\, E_i(r) \leq 1\}$ where(r_x, r_y, r_z) is the pose of the quadcopter, (x_i, y_i, z_i) is the position of a hoop ($\mathcal{H}_i$) with index i and a, b, and c are the x-radius, y-radius and z-radius of the hoops, respectively. We define these three constant radii $(a, b, c) \in \mathbb{R}_{>0}$ for the hoops to represent the volume covered by each hoop in our experiments and note they are equivalent for all hoops.

Additionally, each hoop contains five control points (two avoid points, front, rear, and center) in $\mathbb{R}^3$, as illustrated in Fig. 2. We leverage these control points as anchor points defining locations of interest near hoops and use them to guide trajectories between hoops. A motion plan is provided



(a) A top-down view of the orientation of control points for a hoop where the straight line depicted represents a hoop as seen from above. The five labeled dots represent the five control points (two avoid points, rear, center, and front) for each hoop.



(b) A front view of the hoop, depicted as a circle in the above image, with the five control points (two avoid points, rear, center and front). Note, the center and rear control points are occluded in this view.

Fig. 2. The orientation of the control points around an example hoop. We define five control points as points in $\mathbb{R}^3$ at desired positions around hoops which are used as anchor points for trajectories generated between them.

to the *sequence-of-hoops planner*. This motion plan is a sequence of hoops given by name and direction indicating which labeled side of the hoop the quadcopter must fly through. For example, to fly through "$hoop_0$" in the forward direction followed by "$hoop_1$" in the rear direction, the sequence would be G = '$hoop_0$F $hoop_1$R'. We now follow with the description of our sequence-of-hoops planner.

A. Flying through Hoops in a Given Sequence

In this section, we describe our sequence-of-hoops trajectory planner in Algorithm 1. In particular, we consider the following scenario: **Scenario 4.1:** Given a quadcopter and a set of pre-positioned labeled hoops, design a trajectory to fly through a specific sequence of hoops.

The algorithm receives as input the current pose of the robot (r), a set of n hoops ($\mathcal{H}$) (each defined by the control points previously mentioned), the input sequence ($\mathcal{G}$) of hoop/direction pairs, and the previous control points visited by the quadcopter ($c_{1:m}$). These control points are used to generate four distinct curves based on the distance of the current pose and future control points along with the direction of the quadcopter and generates segments ($segment$) via spline interpolation that joins control points together. We first define the current (pos_curr) and previous (pos_prev) pose which are found based on the order of the input sequence. In Line 3, the past direction (dir_{past}) of the last control point transition is recovered from the list of previous control points.

In Lines 5 - 7 we define the next control point (pos_next) and the distance vector ($dist_{curr,next}$) to the next control point. In Line 5, the function NEXT_CONTROL_POINT uses the current sequence to determine which control point on a labeled hoop is next. The function DIRECTION in Line 7 gets the direction that the quadcopter is heading. This will inform our algorithm which type of trajectory, out of four pre-determined curves, will be chosen for a particular path segment. Starting at Line 8, hoops that are in the state space but not in the input sequence ($\mathcal{G}$) are avoided using the AVOID function. The AVOID function receives as input the current position, a hoop from the list of hoops and the desired next position. If a hoop is between these two points, the function will choose the closest avoid control point of a particular hoop without crossing through that hoop and that resulting $segment$ from the avoid control point to the current position will be added to the total trajectory $\eta(t)$.

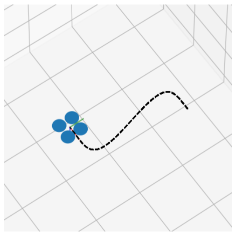
Algorithm 1: Sequence-of-Hoops Planner

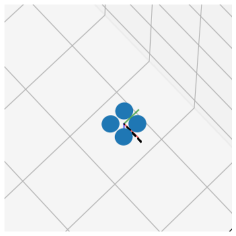
```
input : pose of quadcopter $r$, hoops $\mathcal{H}$, input sequence $\mathcal{G}$,
        control_points_prev $c_{1:m}$
output: trajectory $\eta(t)$
1  $pos\_curr \leftarrow r$
2  $pos\_prev \leftarrow c_m$
3  $dir_{past} \leftarrow$ DIRECTION($pos\_curr, pos\_prev$)
4  for $i \leftarrow 0$ to $len(\mathcal{G})$ do
5      $pos\_next \leftarrow$ NEXT_CONTROL_POINT($\mathcal{G}_i$)
6      $dist_{curr,next} \leftarrow$ DISTANCE($pos\_curr, pos\_next$)
7      $dir_{curr,next} \leftarrow$ DIRECTION($pos\_curr, pos\_next$)
8      for $hoop$ in $\mathcal{H}$ do
9          if $pos\_next$ != $hoop$ then
10             $segment$
                 $\leftarrow$ AVOID($pos\_curr, hoop, pos\_next$)
11         end
12     end
13     if $dist^{yz}_{curr,next}$ == 0 then
14         $segment \leftarrow$ STRAIGHT($pos\_curr, pos\_prev$)
15     end
16     if $dir_{curr,next} == dir_{past}$ then
17         if $dist^{x}_{curr,next} \leq dist^{yz}_{curr,next}$ then
18             $segment \leftarrow$ U_TURN($dist_{curr,next}$)
19         else
20             $segment \leftarrow$ S_CURVE($dist_{curr,next}$)
21         end
22     else
23         $segment \leftarrow$ TURN($dist_{curr,next}$)
24     end
25     $pos\_prev \leftarrow pos\_curr$
26     $pos\_curr \leftarrow$ RETURN_CONTROL_POINT($segment$)
27     $\eta(t) = \eta(t) + segment$
28 end
29 return $\eta(t)$
```

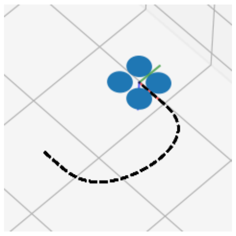
Characterizing a Trajectory Segment: For the four types of trajectories that can be generated, all are parameterized by time and are described by our algorithm in Lines 14 - 24. We describe each function in detail here.

STRAIGHT: If $dist_{yz}$ == 0, The function STRAIGHT in Line 14 generates a trajectory segment length $||\eta_s(t)|| = |r^c_x - r^w_x|$ where r^c_x is the $x - component$ of the current pose and r^w_x indicates the $x - component$ of the previous pose.

U_TURN: A trajectory the length of two quarter arcs joined by a straight segment is returned by the U_TURN function in


(a) S_CURVE: an arc produced when $dist_x > dist_{yz}$ and quadcopter is not changing direction


(b) STRAIGHT: a straight line curve generated in the x-direction when $dist_{yz}$ is zero


(c) U_TURN: curve produced when quadcopter is changing direction

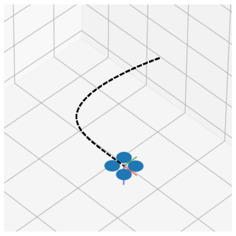

(d) TURN: curve produced when $dist_x \geq dist_{yz}$ and quadcopter is not changing direction

Fig. 3. Trajectory segments that are generated in the sequence-of-hoops planner.

Line 18 where $||\eta_s(t)|| = (dist_{yz} - dist_x) + (dist_x \cdot \frac{\pi}{2})$.

S_CURVE: In Line 20, the length of the trajectory segment is created by first defining a right triangle leg (rt), such that $rt = \delta\sqrt{dist_x^2 - dist_{yz}^2}$. Then, theta ($\theta$) and the hypotenuse ($h$) of the triangle are defined as $\theta = atan2(dist_{yz}, dist_x)$ and $h = \frac{rt}{\sin\theta}$, respectively. The total length of the trajectory results in $||n_s(t)|| = 2(2h\theta)$ which generates two equal length tangent arcs or an "s-curve" in function S_CURVE.

TURN: If the direction is not the same as last, the trajectory length becomes a straight segment followed by a half-circle that changes the direction of the quadcopter such that $||\eta_s(t)|| = dist_x + (dist_y \cdot \frac{\pi}{2})$, the function TURN generates this segment.

In Fig. 3 we show each type of trajectory segment that our planner can generate. The path lengths are then transformed into time segments via $t_s = \frac{||\eta_s(t)||}{v}$, where v is the desired speed. These time segments (t_s) along with the waypoints ($w_{1:m}$) given by the input sequence are used to generate smooth trajectories ($\eta(t)$) via spline interpolation.

Scenario 4.1 Example: Using the sequence-of-hoops planner, we provide the following sequence $\mathcal{G}$ = '$hoop_0$F $hoop_1$F $hoop_2$F $hoop_1$R'. The provided sequence requires a quadcopter to navigate through 4 unique hoop direction pairs in a specified order. In Fig. 4, we provide an illustrated trajectory following the specified sequence passed to this algorithm. A path is planned through all desired hoops and the quadcopter navigates to a pre-specified final waypoint. In the depicted simulation, our trajectories are constrained with respect to predefined sequences. In the next section, we propose using the expressive capacity of LTL to enhance our algorithm. Moreover, we implement an improvement on the sequence-of-hoops planner that can utilize LTL specifications to generate sequences that are not explicitly given by a user.

B. Finding Satisfying Sequences Given an LTL Specification

In Section III-A, we proposed a planner that can generate trajectories through explicitly defined sequences of hoops. However, suppose instead of satisfying explicit sequences, we utilized LTL as a method to convey high-level user input into trajectories that satisfy these specifications. Consider the following scenario: **Scenario 4.2:** Given a quadcopter and

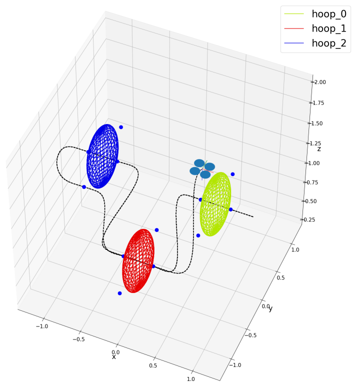


Fig. 4. A sample sequence $\mathcal{G}$ = '$hoop_0$F $hoop_1$F $hoop_2$F $hoop_1$R' in simulation using the sequence-of-hoops planner.

a set of pre-positioned labeled hoops, design a trajectory that satisfies an LTL specification. For example, consider the following specification: "always ensure flying through $hoop_0$ implies $hoop_2$ is flown through before $hoop_1$ and eventually reach $hoop_1$". This specification can be represented by the LTL formula $\phi = \Box(\Diamond hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \Diamond hoop_1)$. In this section, we define LTL and modify our sequence-of-hoops planner to accommodate LTL specifications.

Defining LTL_{-X} Over Continuous Space: LTL [8] is a logic formalism that is suited for specifying linear time properties [16], [17], [18]. Because our formula is constrained to the dynamics of a quadcopter, we use a subclass of LTL known as LTL_{-X} that is similar to LTL except it is defined without the X(next) operator. This subclass of LTL is well-known for formal verification of trajectories for robotic systems [19] and we will refer to it as LTL.

Definition 2: Let Π be a finite set of atomic propositions. Atomic propositions can be used to define state space properties like if certain regions in the state space are occupied (eg. "Is robot in area A?"). We define hoops ($hoop_n$) as atomic propositions so that Π=$\{hoop_0, hoop_1, \ldots, hoop_n\}$.

We check that propositions $hoop_i$ are satisfied by mapping the hoop set ($\mathcal{H}_i$) in Definition 1, to the propositions through the labeling function k, where $k(r) = \{hoop_i \in \Pi : r \in \mathcal{H}_i\}$. In other words, we map the position of a quadcopter ($r \in \mathbb{R}^3$) to a set of corresponding hoop propositions. For example, $k(r) = \{hoop_0\}$ iff r belongs to $\mathcal{H}_0$. For continuous quadcopter trajectories $\eta_c(t)$, we use a slight overload of notation for the following definition.

Definition 3: Let us define k over a continuous trajectory where $k(\eta_c) = \{hoop_{i_1}, hoop_{i_2}, hoop_{i_3}, \ldots, hoop_{i_j}\}$ is the sequence of hoops visited by a quadcopter and $hoop_i \in \Pi$ and j indicates the j^{th} hoop in the sequence.

This labeling function generates a sequence of propositions from the continuous trajectory $\eta_c(t)$ for $t \geq 0$. For example, a sequence could have the form $k(\eta_c) = \{hoop_0, hoop_1, hoop_0\}$. If the continuous trajectory $\eta_c(t)$ is created such that all propositions ($hoop_i$) generated from the trajectory satisfy an LTL formula ζ, then we have successfully found a trajectory that satisfies a given LTL formula. Therefore, the trajectory η_c satisfies the LTL formula ζ iff $k(\eta_c) \models \zeta$.

Generating Büchi Automata from LTL: After defining discrete propositions in continuous space, we can plan trajectories in the discrete space and map the trajectories back into the continuous domain using our sequence-of-hoops planner. From [8], any LTL formula can be represented as a Büchi Automaton, which are defined as:

Definition 4: The tuple $\mathcal{B} = (\mathcal{Q}, \Pi, \delta, \mathcal{Q}_0, \mathcal{F})$ is a nondeterministic Büchi Automaton (NBA) where $\mathcal{Q}$ denotes a finite set of states, Π denotes the input alphabet, $\delta : \mathcal{Q} \times \Pi \rightarrow 2^{\mathcal{Q}}$ is the transition function, $\mathcal{Q}_0 \subseteq \mathcal{Q}$ represents the set of initial states, and $\mathcal{F} \subseteq \mathcal{Q}$ is the set of final states.

A run $q = q_0 q_1 \ldots$ of a Büchi Automaton where $q_i \in \mathcal{Q}$ is an *accepting run* if $q_i \in \mathcal{F}$ for infinitely many indices. Associated with run q is a sequence of propositions ($\pi \in \Pi$) such that an infinite word $\sigma = \pi_0, \pi_1, \cdots \in \Pi$ is accepted if there is an accepting run for σ. Using efficient tools to translate LTL to nondeterministic Büchi Automaton (NBA) [20], we represent the NBA as a directed graph. We then search through the graph for runs σ of the prefix-suffix form $\sigma = (q_0, \ldots q_n)(q_{n+1}, \ldots)^{\omega}$ where the suffix portion of the run contains at least one state q_i within the accepting set. The n^{th} state indicates the last state in the prefix of the run. Once a satisfying run has been found, the satisfying word (i.e., a sequence of hoops) has been found and we apply our sequence-of-hoops planner to find a continuous trajectory.

Scenario 4.2 Example: In regards to our proposed scenario, we augment the sequence-of-hoops planner to accept LTL specifications. We introduce a function `LTL_TO_SEQUENCE` which receives an LTL specification, generates an equivalent NBA and searches for a satisfying sequence. In order to determine which direction (front or rear) the quadcopter must fly through the hoop from, we use the Euclidean distance between each proposed hoop to find the closest control point between two consecutive hoops in the sequence. The corresponding directions are appended to the hoops in the sequence and the planner executes as before. From the proposed scenario, we get the sequence: $\Sigma_{prop} = (hoop_1)(hoop_2)^{\omega}$ in prefix-suffix form. We note that Σ_{prop} is used to denote the accepting word for the NBA generated from our sample LTL specification and ω indicates the hoop, or set of hoops, that can be visited infinitely often during the execution of the algorithm. The resulting trajectory is shown in Figure 5.

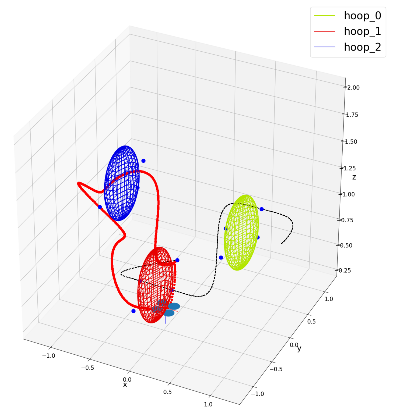


Fig. 5. A satisfying run from the LTL specification ($\phi = \Box(\Diamond hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \Diamond hoop_1)$). We show the prefix portion of the trajectory in black and the suffix portion in red. The suffix portion indicates the set of hoops that can be visited infinitely often.

C. *Optimizing Sequences Using Cross-Entropy*

Although the sequence-of-hoops planner is able to generate trajectories from user provided LTL specifications, the path is generally not optimal. We draw from the following scenario to motivate our problem: **Scenario 4.3:** Given a sequence-of-hoops planner for quadcopters, utilize the cross-entropy method to return trajectories that satisfy the sequence as well as minimize cost. Use this method to guarantee an optimal trajectory with respect to a cost function $\mathcal{J}(r, u)$, parameterized by the robot pose, r, and its control input u. We use the cross-entropy method as a stochastic optimization technique for choosing trajectories according to our cost function, which we define to minimize the total length of the trajectory, i.e., $\mathcal{J}(r, u) = ||\eta(\cdot)||$. In the following, a high-level overview of the cross-entropy method will be given followed by a proposed algorithm that further extends the augmented sequence-of-hoops planner in Section III-B to ensure optimality. For a more detailed description of the method see [21], [11].

The Cross Entropy Method: Cross-entropy optimization is a method of importance sampling for probablistically rare events. The algorithm design for using cross-entropy with motion planning [12] can be generalized as the following:

1) Generate a set of sample trajectories (J) from a distribution and calculate cost $\mathcal{J}(r, u)$ for each trajectory
2) Update the distribution using a subset of samples (κ), until the sampling distribution converges to a desired cost (Σ) and delta function over the optimal trajectory

The subset of samples is defined as $\kappa = \rho J$, where $\rho = \{\rho \in \mathbb{R} : 10^{-1} \leq \rho < 0.3\}$ and J is ordered from least cost to greatest. While this method may not generate a globally optimal solution, due to it being a non-convex optimization method, the entire state space ($\mathcal{X}$) will be explored during trajectory generation. In the next section, we describe our algorithm for minimizing the sequence-of-hoops trajectory using cross-entropy.

Optimizing the Sequence-of-Hoops Planner: From the augmented sequence-of-hoops planner in Section III-B, we apply cross-entropy optimization to reduce the cost of the sampled trajectories once they are generated. Our algorithm is adapted from [11] with modifications on sampling initial means. We sample from the sequence-of-hoops planner to generate initial means to ensure that only the subset of the state space relevant to our hoop sequence is sampled.

Algorithm 2: Cross-Entropy Sequence-of-Hoops Algorithm

```
1   input : LTL formula $\zeta$, hoop_propositions $i \to n$ $\mathcal{H}_{i:n}$,
            number of trajectories $T$, optimal cost $\Sigma$, elite set
            modifier $\rho$, sampling distribution $p(\mu_0, v)$,
            iteration number $N$
    output: best_path $\eta(t)$
2   n := initial iteration number
3   $\mu_0$ = Sequence-of-Hoops Planner($r$, $\mathcal{H}_{i:n}$, $\zeta$, $w_{1:m}$)
4   best_cost := $\infty$
5   while $best\_cost > \Sigma$ and $n < N$ do
6       for $i$ in $T$ do
7           $path\_samples \to p(\cdot, v)$
8           $\eta(t) \to \mathtt{PATH}(path\_samples)$
9           $path\_check \to \mathtt{LTL\_PARSER}(\zeta, \eta(t), \mathcal{H}_{i:n})$
10          if $path\_check$==$TRUE$ then
11      $sorted\_trajectories \to \eta_1(t) < \eta_2(t) \cdots < \eta_p(t)$
12      $best\_cost = sorted\_trajectories_0$
13      $elite\_set = \rho * sorted\_trajectories$
14      $p(\cdot, v) \to \mathtt{UPDATE}(sorted\_trajectories)$
15  return $\eta(t)$
```

Path samples are sampled from a multi-variate Gaussian distribution. The means (μ_0) are initialized to be n equidistant samples from the augmented Sequence-of-Hoops planner. As a result, we get path samples from Line 7. Line 8 defines a PATH function that receives as input samples from the distribution $p(\mu_0, v)$ and generates a trajectory via spline interpolation. This trajectory is then checked in Line 9 where it is monitored for inclusion in hoop sets ($\mathcal{H}$). The trajectory is then parsed in the syntax of the accepting set of hoops and checked for whether it satisfies the string received initially from the augmented planner generated from the LTL formula (ζ). If this check returns $TRUE$ the trajectory is returned, otherwise a new trajectory is sampled. Trajectories are collected and sorted from best cost to worst cost and an elite set is chosen corresponding to a subset of trajectories (T). We then update the probability distribution using the elite subset of trajectories.

Scenario 4.3 Example: Using the cross-entropy sequence-of-hoops planner, we optimize over the LTL formula $\phi = \square(\lozenge hoop_0 \to \neg hoop_1 \cup hoop_2 \wedge \lozenge hoop_1)$ provided in Section III-B. In Fig. 7, we show the optimized trajectory of the LTL formula with the constraint that the *each* trajectory segment (prefix and suffix) length should be less than 5 meters or $\mathcal{J}(r, u) \leq 5$. In Fig. 6 we show the sampled paths over an iteration of the algorithm.

IV. Demonstration of Planner

We implement the planner on the Robotarium at Georgia Tech where we use Crazyflie 2.0 quadcopters [22], [23]. The Robotarium uses a Vicon Tracking system which records

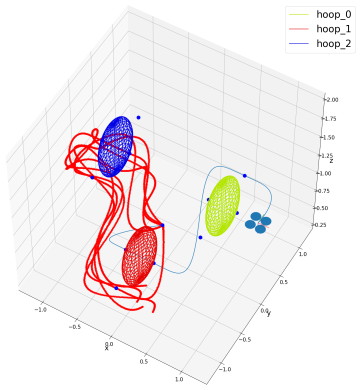


Fig. 6. Here we show a series of trajectories generated from Algorithm 2. The prefix portion of the trajectory, represented in blue, satisfied the cost function $\mathcal{J} = ||\eta(t)||$ initially therefore, only one sample was needed. Several samples of the suffix portion, in red, of the trajectory were sampled before a satisfying trajectory was found.

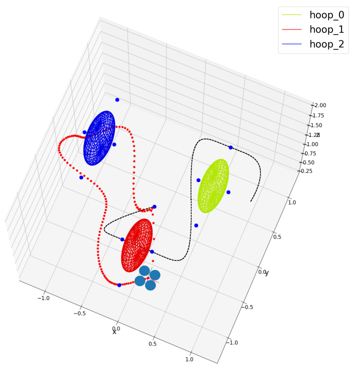


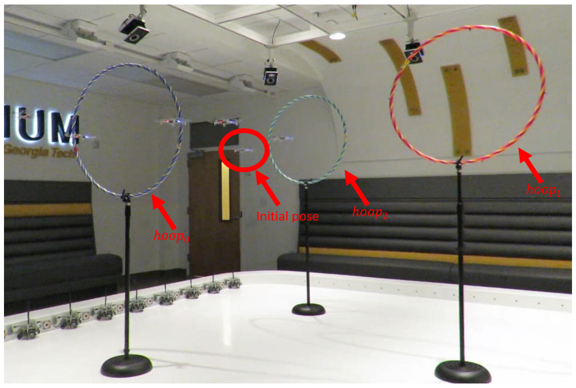
Fig. 7. A simulation snapshot of the LTL cross-entropy sequence-of-hoops planner after a satisfying run. The optimized trajectory of is shown here where each trajectory segment must satisfy the cost constraint $\mathcal{J}(r, u) \leq 5$.

real-time position of robots with a 100 Hz update rate. The algorithm was created in Python and sends control inputs to a PID controller in C++. Commands are sent via ROS messages to Crazyflies and a radio operating in the 2400 MHz range with a data rate of 2 Mbit/s sends these commands to the quadcopters. We orient hoops with vertical stands, as pictured in Fig. 8a, in the Robotarium and mark them with Vicon tracking points to record the center of the hoops and generate the other control points.

Using the methods we have described in the previous sections, a user can specify either an explicit sequence of hoops or an LTL formula via a command line interface (CLI). Depending on the input, the sequence is then parsed through the sequence-of-hoops planner as a sequence of hoops or LTL formula and generates trajectories that maneuver a quadcopter through hoops satisfying that specification. The user is also given the option to give a minimizing cost as well. Using the cross-entropy LTL algorithm, trajectories are minimized via cost functions ($\mathcal{J}$) provided a priori.

In Fig. 8b, a composite image of a quadcopter is shown completing a trajectory. We define the sequence ($\mathcal{G}$) for the sequence-of-hoops planner to execute as the same in Section III-A. We use spline interpolation to generate trajectories

(a) The hoops are mounted on adjustable stands and are tracked via Vicon markers.



(b) A composite image of the quadcopter executing the beginning of the automatically generated trajectory from sequence $\mathcal{G}$ = '0F 1F 2F 1R'.

Fig. 8. Hoops are placed in the Robotarium and users can give sequences/LTL specifications via a command line interface (CLI).

between waypoints and define a constant velocity of 0.45 m/s. By restricting the planning to discrete hoop-to-hoop transitions we argue we have provided a potential solution to the hoop flying problem with our sequence-of-hoops planner. Using LTL, specifications as hoops can naturally be defined as discrete objects and enhances the range of specifications that the planner can satisfy. Leveraging the dynamics of quadcopters allows us to generate continuous trajectories, implement these trajectories on actual aerial robotic systems and provide applicable use cases for LTL with robotic systems.

V. Conclusions

Through the methods developed and implemented in this paper, we design an end-to-end solution to the hoop flying problem. We show that given an explicitly labeled sequence of hoop/direction pairs the planner generates trajectories from four predefined trajectory segments to satisfy the sequence. We also show that for more complex specifications that may not be easily defined as sequences can be defined as LTL specifications and parsed through the planner as input. In addition, to ensure optimality, the cross-entropy optimization method is utilized on nominal trajectories to optimize via user provided cost functions. In order to enhance the Robotarium user experience with quadcopters, we have implemented this as a novel interaction modality for external users to engage with aerial vehicles in the Robotarium.

References

[1] L. García-Pérez, M. García-Alegre, A. Ribeiro, and D. Guinea, "An agent of behaviour architecture for unmanned control of a farming vehicle," *computers and electronics in agriculture*, vol. 60, no. 1, pp. 39–48, 2008.

[2] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[3] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[4] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 55–64, 2011.

[5] S. A. Bortoff, "Path planning for UAVs," in *American Control Conference, 2000. Proceedings of the 2000*, vol. 1, no. 6. IEEE, 2000, pp. 364–368.

[6] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," *CoRR*, vol. abs/1307.7263, 2013. [Online]. Available: http://arxiv.org/abs/1307.7263

[7] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," 2005.

[8] C. Baier and J.-P. Kateon, *Principles of Model Checking*. Cambridge, Massachusetts: The MIT Press, 2008.

[9] D. Zhou and M. Schwager, "Vector field following for quadrotors using differential flatness," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6567–6572.

[10] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2520–2525.

[11] S. C. Livingston, E. M. Wolff, and R. M. Murray, "Cross-entropy temporal logic motion planning," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, 2015, pp. 269–278.

[12] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[13] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The Robotarium: A remotely accessible swarm robotics research testbed," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1699–1706.

[14] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," *arXiv preprint arXiv:1710.05472*, 2017.

[15] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *ASME international mechanical engineering congress and exposition*. Citeseer, 1995.

[16] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[17] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison-Wesley Reading, 2004, vol. 1003.

[18] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 4, p. 14, 2011.

[19] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[20] A. Duret-Lutz and D. Poitrenaud, "Spot: an extensible model checking library using transition-based generalized bu/spl uml/chi automata," in *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings*. IEEE, 2004, pp. 76–83.

[21] P.-T. de Boer *et al.*, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005.

[22] B. AB. (2018). [Online]. Available: https://www.bitcraze.io/

[23] W. Hönig and N. Ayanian, "Flying multiple UAVs using ROS," in *Robot Operating System (ROS)*. Springer, 2017, pp. 83–118.