(Tech Report) Fly-by-Logic: Control of Multi-Drone Fleets with Temporal Logic Objectives

Yash Vardhan Pant, Houssam Abbas, Rhudii A. Quaye, Rahul Mangharam

***Abstract*—The problem of safe planning and control for multi-drone systems across a variety of missions is of critical importance, as the scope of tasks assigned to such systems increases. In this paper, we present an approach to solve this problem for multi-quadrotor missions. Given a mission expressed in Signal Temporal Logic (STL), our controller maximizes robustness to generate trajectories for the quadrotors that satisfy the STL specification in continuous-time. We also show that the constraints on our optimization guarantees that these trajectories can be tracked nearly perfectly by lower level off-the-shelf position and attitude controllers. Our approach avoids the oversimplifying abstractions found in many planning methods, while retaining the expressiveness of missions encoded in STL allowing us to handle complex spatial, temporal and reactive requirements. Through experiments, both in simulation and on actual quadrotors, we show the performance, scalability and real-time applicability of our method.**

I. Introduction

As the technology behind autonomous systems is starting to mature, they are being envisioned to perform greater variety of tasks. Fig. 1 shows a scenario where multiple quadrotors have to fly a variety of missions in a common air space, including package delivery, surveillance, and infrastructure monitoring. Drone A is tasked with delivering a package, which it has to do within 15 minutes and then return to base in the following 10 minutes. Drone B is tasked with periodic surveillance and data collection of the wildlife in the park, while Drone C is tasked with collecting sensor data from equipment on top of the white-and-blue building. All of these missions have complex spatial requirements (e.g. avoid flying over the buildings highlighted in red, perform surveillance or monitoring of particular areas and maintain safe distance from each other), temporal requirements (e.g., a deadline to deliver package, periodicity of visiting the areas to be monitored) and reactive requirements (like collision avoidance). The *safe* planning and control of multi-agent systems for missions like these is becoming an area of utmost importance. Most existing work for this either lacks the expressiveness to capture such requirements (e.g. [1], [2]), relies on simplifying abstractions that result in conservative behavior ([3]), or do not take into account explicit timing constraints ([4]). A more detailed coverage of existing methods can be found in Sec.I-A. In addition to these limitations, many of the planning methods are computationally intractable (and hence do not scale well or work in real-time), and provide guarantees only on a simplified abstraction of the system behavior ([3]).

Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, USA.
{yashpant, habbas, quayerhu, rahulm}@seas.upenn.edu

This work was supported by STARnet a Semiconductor Research Corporation program sponsored by MARCO and DARPA, NSF MRI-0923518 and the US Department of Transportation University Transportation Center Program.

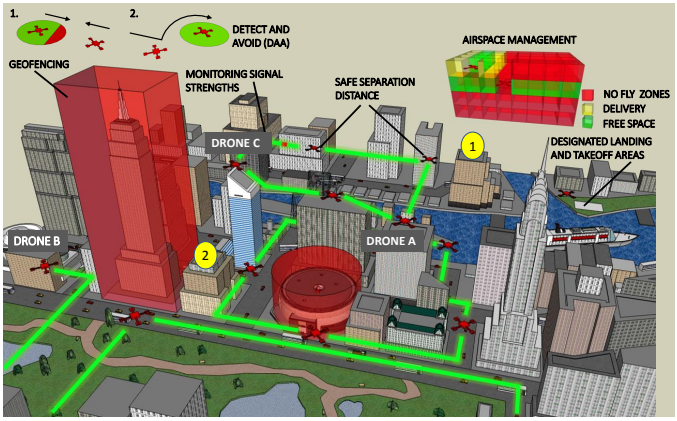

Fig. 1: Multiple autonomous drone missions in an urban environment. (Figure adapted from https://phys.org/news/2016-12-traffic-solutions-drones-singapore-airspace.html)

In this work, we aim to overcome some of those limitations, by focusing on a single class of dynamical systems, namely multi-rotor drones, such as quadrotors. By using Signal Temporal Logic (STL) as the mission specification language, we retain the flexibility to incorporate explicit timing constraints, as well as a variety of behaviors. Without relying on oversimplifying abstractions, we provide guarantees on the continuous-time behavior of the dynamical system. We also show through experiments that the resulting behavior of the drones is not conservative. The control problem formulation we present is aimed to be tractable, and through both simulations and experiments on actual drones we show that we can control up to two drones in real-time and up to 16 in simulation.

A. State of the art

The mission planning problem for multiple agents has been extensively studied. Most solutions work in an abstract grid-based representation of the environment [4], [5], and abstract the dynamics of the agents [6], [3]. As a consequence they have correctness guarantees for the discrete behavior but not necessarily for the underlying continuous system. Multi-agent planning with kinematic constraints in a discretized environment has been studied in [7] with application to ground robots. Planning in a discrete road map with priorities assigned to agents has been studied in [2] and is applicable to a wide variety of systems. Another priority-based scheme for drones using a hierarchical discrete planner and trajectory generator has been studied in [1]. Most of these use Linear Temporal Logic (LTL) as the mission specification language,

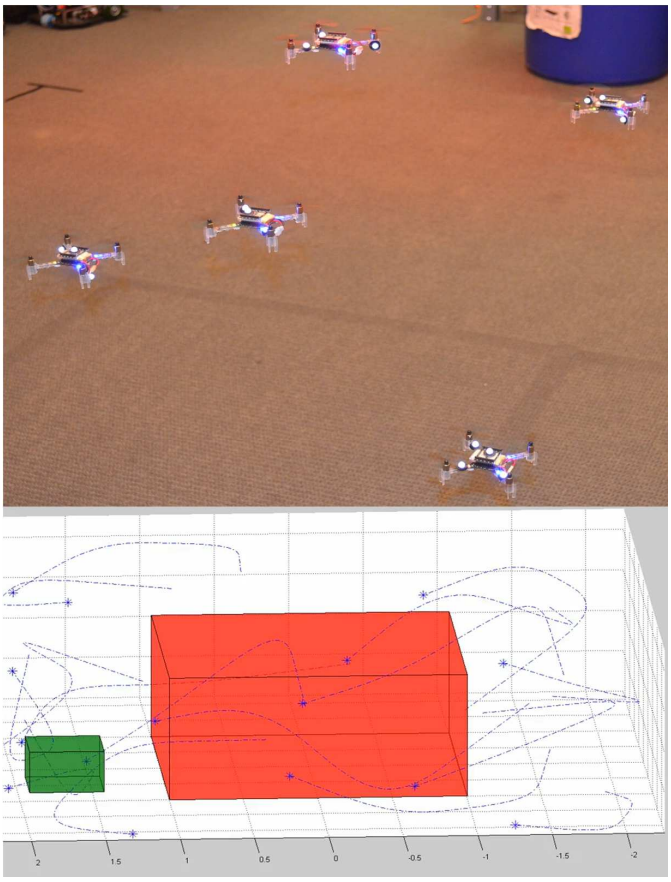

Fig. 2: (Top) Five Crazyflie 2.0 quadrotors executing a reach-avoid mission. (Bottom) A screenshot of a simulation with 16 quadrotors. In both cases, the quadrotors have to satisfy a mission given in STL.

which doesn't allow explicit time bounds on the mission objectives. The work in [3] uses STL. Other than [2] and [1], none of the above methods can run in real-time because of their computational requirements. While [2] and [1] are real-time, they can only handle the Reach-Avoid mission, in which agents have to reach a desired goal state while avoiding obstacles and each other.

In a more control-theoretic line of work, control of systems with STL or Metric Temporal Logic (MTL) specifications without discretizing the environment or dynamics has been studied in [8], [9], [10]. These methods are potentially computationally more tractable than the purely planning-based approaches discussed earlier, but are still not applicable to real-time control of complex dynamical systems like quadrotors (e.g. see [10]), and those that rely on Mixed Integer Programming based approaches [8] do not scale well. Stochastic heuristics like [11] have also been used for STL missions, but offer very few or no guarantees. Non-smooth optimization has also been explored, but only for safety properties [12].

In this paper, we focus on multi-rotor systems, and work with a continuous representation of the environment and take into account the behavior of the trajectories of the quadrotor. With the mission specified as a STL formula, we maximize a smooth version of the robustness ([3], [10]). This, unlike a majority of the work outlined above, allows us to take into account explicit timing requirements. Our method also allows us to use the full expressiveness of STL, so our approach is not limited to a particular mission type. Finally, unlike most of the work discussed above, we offer guarantees on the continuous-time behavior of the system to satisfy the spatio-temporal requirements. Through simulations and experiments on actual platforms, we show real-time applicability of our method for simple cases, as well as the scalability in planning for multiple quadrotors in a constrained environment for a variety of mission specifications.

B. Contributions

This paper presents a control framework for mission planning and execution for fleets of quadrotors, given a STL specification.

1) **Continuous-time STL satisfaction:** We develop a control optimization that selects waypoints by maximizing the robustness of the STL mission. A solution to the optimization is guaranteed to satisfy the mission in continuous-time, so trajectory sampling does not jeopardize correctness, while the optimization only works with a few waypoints.
2) **Dynamic feasibility of trajectories:** We demonstrate that the trajectories generated by our controller respect pre-set velocity and acceleration constraints, and so can be well-tracked by lower-level controllers.
3) **Real-time control:** We demonstrate our controller's suitability for online control by implementing it on real quadrotors and executing a reach-avoid mission.
4) **Performance and scalability:** We demonstrate our controller's speed and performance on real quadrotors, and its superiority to other methods in simulations.

The paper is organized as follows. Section II introduces STL and its robust semantics, while section III introduces the control problem we aim to solve in this paper. Section IV presents the proposed control architecture and the trajectory generator we use, and proves that the trajectories are dynamically feasible. The main control optimization is presented in Sec. V. Extensive simulation and experiments on real quadrotors are presented in Sections VI and VII, resp. All experiments are illustrated by videos available at the links in Table V.

II. Preliminaries on Signal Temporal Logic

Consider a continuous-time dynamical system $\mathcal{H}$ and its uniformly discretized version

$$\dot{x_c}(t) = f_c(x_c(t), u(t)), \quad x^+ = f(x, u) \tag{1}$$

where $x \in X \subset \mathbb{R}^n$ is the current state of the system, x^+ is the next state, $u \in U \subset \mathbb{R}^m$ is its control input and $f : X \times U \to X$ is differentiable in both arguments. The system's initial state x_0 takes values from some initial set $X_0 \subset \mathbb{R}^n$. In this paper we deal with trajectories of the same duration (e.g. 5 seconds) but sampled at different rates, so we introduce notation to make the sampling period explicit. Let $dt \in \mathbb{R}_+$ be a sampling period and $T \in \mathbb{R}_+$ be a trajectory duration. We write $[0 : dt : T] = (0, dt, 2dt, \ldots, (H-1)dt)$ for the sampled time interval s.t. $(H-1)dt = T$ (we assume T is divisible

by $H-1$). Given an initial state x_0 and a finite control input sequence $\mathbf{u} = (u_0, u_{t_1} \ldots, u_{t_{H-2}}), u_t \in U, t_k \in [0:dt:T]$, a *trajectory* of the system is the unique sequence of states $\mathbf{x} = (x_0, x_{t_1} \ldots, x_{t_{H-1}})$ s.t. for all $t \in [0:dt:T]$, x_t is in X and $x_{t_{k+1}} = f(x_{t_k}, u_{t_k})$. We also denote such a trajectory by $\mathbf{x}[dt]$. Given a time domain $\mathbb{T} = [0:dt:T]$, the signal space $X^{\mathbb{T}}$ is the set of all signals $\mathbf{x}: \mathbb{T} \to X$. For an interval $I \subset \mathbb{R}_+$ and $t \in \mathbb{R}_+$, set $t + I = \{t + a \mid a \in I\}$. The max operator is written $\sqcup$ and min is written $\sqcap$.

A. Signal Temporal Logic (STL)

The controller of $\mathcal{H}$ is designed to make the closed loop system (1) satisfy a specification expressed in Signal Temporal Logic (STL) [13], [14]. STL is a logic that allows the succinct and unambiguous specification of a wide variety of desired system behaviors over time, such as "The quadrotor reaches the goal within 10 time units while always avoiding obstacles" and "While the quadrotor is in Zone 1, it must obey that zone's altitude constraints". Formally, let $M = \{\mu_1, \ldots, \mu_L\}$ be a set of real-valued functions of the state $\mu_k : X \to \mathbb{R}$. For each μ_k define the *predicate* $p_k := \mu_k(x) \geq 0$. Set $AP := \{p_1, \ldots, p_L\}$. Thus each predicate defines a set, namely p_k defines $\{x \in X \mid f_k(x) \geq 0\}$. Let $I \subset \mathbb{R}$ denote a non-singleton interval, $\top$ the Boolean True, p a predicate, $\neg$ and $\wedge$ the Boolean negation and AND operators, respectively, and $\mathcal{U}$ the Until temporal operator. An STL formula φ is built recursively from the predicates using the following grammar:

$$\varphi := \top | p | \neg\varphi | \varphi_1 \wedge \varphi_2 | \varphi_1 \mathcal{U}_I \varphi_2$$

Informally, $\varphi_1 \mathcal{U}_I \varphi_2$ means that φ_2 must hold at some point in I, and *until* then, φ_1 must hold without interruption. The disjunction ($\vee$), implication ($\implies$), Always ($\Box$) and Eventually ($\Diamond$) operators can be defined using the above operators. Formally, the *pointwise semantics* of an STL formula φ define what it means for a system trajectory $\mathbf{x}$ to satisfy φ.

Definition 2.1 (STL semantics): Let $\mathbb{T} = [0:dt:T]$. The boolean truth value of φ w.r.t. the discrete-time trajectory $\mathbf{x}: \mathbb{T} \to X$ at time $t \in \mathbb{T}$ is defined recursively.

$$\begin{aligned}
(\mathbf{x}, t) &\models \top \quad \Leftrightarrow \quad \top \\
\forall p_k \in AP, (\mathbf{x}, t) &\models p \quad \Leftrightarrow \quad \mu_k(x_t) \geq 0 \\
(\mathbf{x}, t) &\models \neg\varphi \quad \Leftrightarrow \quad \neg(\mathbf{x}, t) \models \varphi \\
(\mathbf{x}, t) &\models \varphi_1 \wedge \varphi_2 \quad \Leftrightarrow \quad (\mathbf{x}, t) \models \varphi_1 \wedge (\mathbf{x}, t) \models \varphi_2 \\
(\mathbf{x}, t) &\models \varphi_1 \mathcal{U}_I \varphi_2 \quad \Leftrightarrow \quad \exists t' \in (t + I) \cap \mathbb{T}. (\mathbf{x}, t') \models \varphi_2 \\
&\qquad\qquad \wedge \forall t'' \in (t, t') \cap \mathbb{T}, \ \ (\mathbf{x}, t'') \models \varphi_1
\end{aligned}$$

We say $\mathbf{x}$ *satisfies* φ if $(\mathbf{x}, 0) \models \varphi$.

All formulas that appear in this paper have bounded temporal intervals: $0 \leq \inf I < \sup I < +\infty$. To evaluate whether such a *bounded* formula φ holds on a given trajectory, only a finite-length prefix of that trajectory is needed. Its length can be upper-bounded by the *horizon* of φ, $hrz(\varphi) \in \mathbb{N}$, calculable as shown in [8]. For example, the horizon of $\Box_{[0,2]}(\Diamond_{[2,4]} p)$ is 2+4=6: we need to observe a trajectory of, at most, length 6 to determine whether the formula holds.

B. Control using the robust semantics of STL

Designing a controller that satisfies the STL formula φ[1] is not always enough. In a dynamic environment, where the system must react to new unforeseen events, it is useful to have a margin of maneuverability. That is, it is useful to control the system such that we *maximize* our degree of satisfaction of the formula. When unforeseen events occur, the system can react to them without violating the formula. This degree of satisfaction can be formally defined and computed using the *robust semantics* of temporal logic [14], [15].

Definition 2.2 (Robustness[15], [14]): The *robustness* of STL formula φ relative to $\mathbf{x}: \mathbb{T} \to X$ at time $t \in \mathbb{T}$ is

$$\begin{aligned}
\rho_\top(\mathbf{x}, t) &= +\infty \\
\rho_{p_k}(\mathbf{x}, t) &= \mu_k(x_t) \, \forall p_k \in AP, \\
\rho_{\neg\varphi}(\mathbf{x}, t) &= -\rho_\varphi(\mathbf{x}, t) \\
\rho_{\varphi_1 \wedge \varphi_2}(\mathbf{x}, t) &= \rho_{\varphi_1}(\mathbf{x}, t) \sqcap \rho_{\varphi_2}(\mathbf{x}, t) \\
\rho_{\varphi_1 \mathcal{U}_I \varphi_2}(\mathbf{x}, t) &= \sqcup_{t' \in (t+I) \cap \mathbb{T}} \Big(\rho_{\varphi_2}(\mathbf{x}, t') \bigsqcap \\
&\qquad \sqcap_{t'' \in [t, t') \cap \mathbb{T}} \rho_{\varphi_1}(\mathbf{x}, t'') \Big)
\end{aligned}$$

When $t = 0$, we write $\rho_\varphi(\mathbf{x})$ instead of $\rho_\varphi(\mathbf{x}, 0)$.
The robustness is a real-valued function of $\mathbf{x}$ with the following important property.

Theorem 2.1: [15] For any $\mathbf{x} \in X^{\mathbb{T}}$ and STL formula φ, if $\rho_\varphi(\mathbf{x}, t) < 0$ then $\mathbf{x}$ violates φ at time t, and if $\rho_\varphi(\mathbf{x}, t) > 0$ then $\mathbf{x}$ satisfies φ at t. The case $\rho_\varphi(\mathbf{x}, t) = 0$ is inconclusive.

Thus, we can compute control inputs by *maximizing* the robustness over the set of finite input sequences of a certain length. The obtained sequence $\mathbf{u}^*$ is valid if $\rho_\varphi(\mathbf{x}^*, t)$ is positive, where $\mathbf{x}^*$ and $\mathbf{u}^*$ obey (1). The larger $\rho_\varphi(\mathbf{x}^*, t)$, the more robust is the behavior of the system: intuitively, $\mathbf{x}^*$ can be disturbed and ρ_φ might decrease but not go negative. In fact, the amount of disturbance that $\mathbf{x}^*$ can sustain is precisely ρ_φ: that is, if $\mathbf{x}^* \models \varphi$, then $\mathbf{x}^* + e \models \varphi$ for all disturbances $e: \mathbb{T} \to X$ s.t. $\sup_{t \in \mathbb{T}} \|e(t)\| < \rho_\varphi(\mathbf{x}^*)$.

If $\mathbb{T} = [0, T]$ then the above naturally defines satisfaction of φ by a *continuous-time* trajectory $\mathbf{y}_c : [0, T] \to X$.

III. Control Using a Smooth Approximation of STL Robustness

The goal of this work is to find a provably correct control scheme for fleets of quadrotors, which makes them meet a control objective φ expressed in temporal logic. So let $\epsilon > 0$ be a desired minimum robustness. We solve the following problem.

$$\begin{aligned}
P: \max_{\mathbf{u} \in U^{N-1}} \rho_\varphi(\mathbf{x}) & \qquad (2a) \\
\text{s.t. } x_{k+1} = f(x_k, u_k), \, \forall k = 0, \ldots, N-1 & \qquad (2b) \\
x_k \in X, u_k \in U \, \forall k = 0, \ldots, N & \qquad (2c) \\
\rho_\varphi(\mathbf{x}) \geq \epsilon & \qquad (2d)
\end{aligned}$$

Because ρ_φ uses the non-differentiable functions max and min (see Def. 2.2), it is itself non-differentiable as a function of the trajectory and the control inputs. A priori, this necessitates the use of Mixed-Integer Programming solvers [8], non-smooth optimizers [16], or stochastic heuristics [11] to

[1] Strictly, a controller s.t. the closed-loop behavior satisfies the formula.

solve P_ρ. However, it was recently shown in [10] that it is more efficient and more reliable to instead approximate the non-differentiable objective ρ_φ by a smooth (infinitely differentiable) function $\tilde{\rho}_\varphi$ and solve the resulting optimization problem $\widetilde{P}$ using Sequential Quadratic Programming. The approximate smooth robustness is obtained by using smooth approximations of min and max in Def. 2.2. In this paper, we also use the smoothed robustness $\tilde{\rho}_\varphi$ and solve $\widetilde{P}$ instead of P. The lower bound on robustness (2d) is used to ensure that if $\tilde{\rho}_\varphi \geq \epsilon$ then $\rho_\varphi \geq 0$. In [10] it was shown that an ϵ can be computed such that $|\rho_\varphi - \tilde{\rho}_\varphi| \leq \epsilon$. This approach was called Smooth Operator [10].

Despite the improved runtime of Smooth Operator, our experiments have shown that it is not possible to solve $\widetilde{P}$ in real-time using the full quadrotor dynamics. Therefore, in this paper, we develop a control architecture that is guaranteed to produce a correct and dynamically feasible quadrotor trajectory. By 'correct', we mean that the *continuous-time, non-sampled* trajectory satisfies the formula φ, and by 'dynamically feasible', we mean that it can be implemented by the quadrotor dynamics. This trajectory is then tracked by a lower-level MPC tracker. The control architecture and algorithms are the subject of the next section.

IV. Quadrotor Control Architecture

Fig. 3 shows the control architecture used in this paper, and its components are detailed in what follows. The overall idea is that we want the *continuous-time* trajectory $\mathbf{y}_c : [0, T] \to X$ of the quadrotor to satisfy the STL mission φ, but can only compute a discrete-time trajectory $\mathbf{x} : [0 : dt : T] \to X$ sampled at a low rate $1/dt$. So we do two things, illustrated in Fig.3:
A) to guarantee continuous-time satisfaction from discrete-time satisfaction, we ensure that a discrete-time high-rate trajectory $\mathbf{q} : [0 : dt' : T] \to X$ satisfies a suitably *stricter* version φ_s of φ. This is detailed in Section V.
B) To compute, in real-time, a sufficiently high-rate discrete-time $\mathbf{q}[dt']$ that satisfies φ_s, we perform a (smooth) robustness maximization over a low-rate sequence of waypoints $\mathbf{x}$ with sampling period $dt >> dt'$. In the experiments (sections VI and VII) we used $dt = 1s$ and $dt' = 50ms$. The optimization problem is such that the optimal low-rate trajectory $\mathbf{x} : [0 : dt : T] \to X$ and the desired high-rate $\mathbf{q}$ are related analytically: $\mathbf{q} = L(\mathbf{x})$ for a known $L : \mathbb{R}^{(T/dt)} \to \mathbb{R}^{(T/dt')}$. So the robustness optimization maximizes $\tilde{\rho}_{\varphi_s}(L(\mathbf{x}))$, automatically yielding $\mathbf{q}[dt']$. Moreover, we must add constraints to ensure that $\mathbf{q}$ is dynamically feasible, i.e., can be implemented by the quadrotor dynamics. Thus, qualitatively, the optimization problem we solve is

$$\begin{aligned} &\max_{\mathbf{x}[dt]} \ \tilde{\rho}_{\varphi_s}(L(\mathbf{x}[dt])) \\ &\text{s.t. } L(\mathbf{x}[dt]) \text{ obeys quadrotor dynamics and is feasible} \\ &\quad \mathbf{x} \text{ and } L(\mathbf{x}[dt]) \text{ are in the allowed air space} \\ &\quad \tilde{\rho}_{\varphi_s}(L(\mathbf{x}[dt])) \geq \varepsilon \end{aligned} \tag{3}$$

The mathematical formulation of the above problem, including the trajectory generator L, is given in Sec. IV-A. But first, we end this section by a brief description of the position and attitude controllers that take the high-rate $\mathbf{q}[dt']$ and provide motor forces to the rotors, and a description of the quadrotor dynamics. The state of the quadrotor consists of its 3D position p and 3D linear velocity $v = \dot{p}$. A more detailed version of the quad-rotor dynamics is in Sec. IX-A.

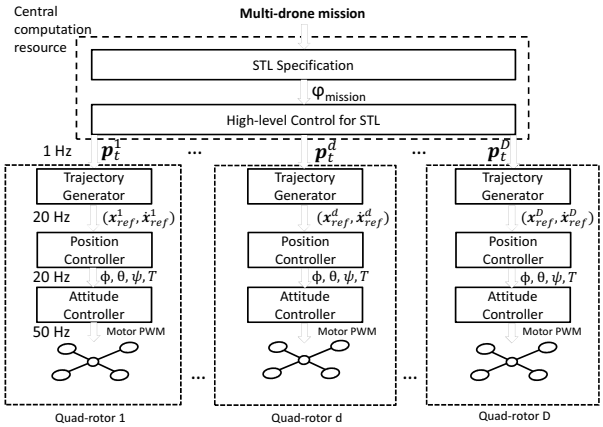

Fig. 3: The control architecture. Given a mission specification in STL, the high-level control optimization (centralized) generates a sequence of waypoints. These waypoints are sent over to the drones, and through a hierarchical control on-board control architecture, the resulting trajectories are tracked near perfectly, with the continuous time behavior of the system satisfying the STL specification.

Position controller To track the desired positions and velocities from the trajectory generator, we consider a Model Predictive Controller (MPC) formulated using the dynamics of (17) linearized around hover. Given desired position and velocity commands in the fixed-world x, y, z co-ordinates, the controller outputs a desired thrust, roll, and pitch command (yaw fixed to zero) to the attitude controller. This controller also takes into account bounds on positions, velocities and the desired roll, pitch and thrust commands.

Attitude controller Given a desired angular position and thrust command generated by the MPC, the high-bandwidth $(50 - 100$ Hz) attitude controller maps them to motor forces. In our control architecture, this is implemented as part of the preexisting firmware on board the Crazyflie 2.0 quadrotors. An example of an attitude controller can be found in [17].

A. The trajectory generator

The mapping L between low-rate $\mathbf{x}[dt]$ and high-rate $\mathbf{y}[dt']$ is implemented by the following trajectory generator, adapted from [18]. It takes in a motion duration $T_f > 0$ and a pair of position, velocity and acceleration tuples, called *waypoints*: an *initial* waypoint $q_0 = (p_0, v_0, a_0)$ and a *final* waypoint $q_f = (p_f, v_f, a_f)$. It produces a continuous-time minimum-jerk (time derivative of acceleration) trajectory $q(t) = (p(t), v(t), a(t))$ of duration T_f s.t. $q(0) = q_0$ and $q(T_f) = q_f$. In our control architecture, the waypoints are the elements of the low-rate $\mathbf{x}$ computed by solving (3). The generator of [18] formulates the quadrotor dynamics of (17) in terms of 3D jerk and this allows a decoupling of the equations along three orthogonal jerk axes. By solving three independent optimal control problems, one along each axis, it

obtains three minimum-jerk trajectories, each being a spline $q^* : [0, T_f] \to \mathbb{R}^3$ of the form:

$$\begin{bmatrix} p^*(t) \\ v^*(t) \\ a^*(t) \end{bmatrix} = \begin{bmatrix} \frac{\alpha}{120}t^5 + \frac{\beta}{24}t^4 + \frac{\gamma}{6}t^3 + a_0t^2 + v_0t + p_0 \\ \frac{\alpha}{24}t^4 + \frac{\beta}{6}t^3 + \frac{\gamma}{2}t^2 + a_0t + v_0 \\ \frac{\alpha}{6}t^3 + \frac{\beta}{2}t^2 + \gamma t + a_0 \end{bmatrix} \quad (4)$$

Here, α, β, and γ are scalar *linear* functions of the initial q_0 and final q_f. Their exact expressions depend on the desired type of motion:

1. Stop-and-go motion. [18] This type of motion yields straight-line position trajectories $p(\cdot)$. These are suitable for navigating tight spaces, since we know exactly the robot's path between waypoints. For Stop-and-Go, the quadrotor starts from rest and ends at rest: $v_0 = a_0 = v_f = a_f = 0$. I.e. the quadrotor has to come to a complete stop at each waypoint. In this case, the constants are defined as follows:

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \frac{1}{T_f^5} \begin{bmatrix} 720(p_f - p_0) \\ -360T_f(p_f - p_0) \\ 60T_f^2(p_f - p_0) \end{bmatrix} \quad (5)$$

2. Trajectories with free endpoint velocities [18] Stop-and-go trajectories have limited reach, since the robot must spend part of the time coming to a full stop at every waypoint. In order to get better reach, the other case from [18] that we consider is when the desired initial and endpoint velocities, v_0 and v_f, are free. Like the previous case, we still assume $a_0 = a_f = 0$. The constants in the spline (4) are then:

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \frac{1}{T_f^5} \begin{bmatrix} 90 & -15T_f^2 \\ -90T_f & 15T_f^3 \\ 30T_f^2 & -3T_f^4 \end{bmatrix} \begin{bmatrix} p_f - p_0 - v_0T_f \\ v_f - v_0 \end{bmatrix} \quad (6)$$

In this case, the trajectories between waypoints are not restricted to be on a line, allowing for a wider range of maneuvers, as will be demonstrated in the simulations of Sec. VI. An example of such a spline (planar) is shown in Fig. 4.

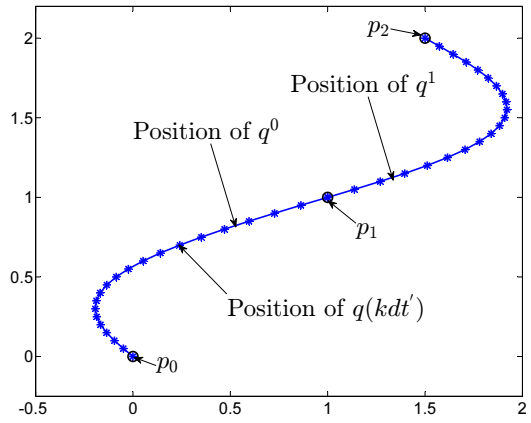

Fig. 4: Planar splines connecting position waypoints p_0, p_1 and p_2. q^0 is the continuous spline (positions, velocities and accelerations) connecting p_0 and p_1 and q_1 is the spline from p_1 to p_2. $q(kdt')$ is the k^{th} sample of q^0, with sampled time dt'. Note, unlike the stop-go case, non-zero end point velocities mean that the resulting motion is not simply a line connecting the way points.

B. Constraints for dynamically feasible trajectories

The splines (4) that define the trajectories come from solving an unconstrained optimal control problem, so they are not guaranteed to respect any state and input constraints, and thus might not be *dynamically feasible*. By dynamically feasible, we mean that the quadrotor can be actuated (by the motion controller) to follow the spline. Typically, feasibility requires that the spline velocity and acceleration be within certain bounds. E.g. a sharp turn is not possible at high speed, but can be done at low speed. Therefore, we formally define dynamic feasibility as follows.

Definition 4.1 (Dynamically feasible trajectories): Let $[\underline{v}, \bar{v}]$ be bounds on velocity and $[\underline{a}, \bar{a}]$ be bounds on acceleration. A trajectory $q : [0, T_f] \to \mathbb{R}^3$, with $q(t) = (p(t), v(t), a(t))$, is *dynamically feasible* if $v(t) \in [\underline{v}, \bar{v}]$ and $a(t) \in [\underline{a}, \bar{a}]$ for all $t \in [0, T_f]$ for each of the three axes of motion.

Assumption 4.1: We assume that dynamically feasible trajectories, as defined here, can be tracked almost perfectly by the position (and attitude) controller. *This assumption is validated by our experiments on physical quadrotor platforms.* See Sec. VII.

In this section we derive constraints on the desired end state (p_f, v_f, a_f) such that the resulting trajectory $q(\cdot)$ computed by the generator [18] is dynamically feasible.

Since the trajectory generator works independently on each jerk axis, we derive constraints for a one-axis spline given by (4). An identical analysis applies to the splines of other axes. Since a quadrotor can achieve the same velocities and accelerations in either direction along an axis, we take $\underline{v} < 0 < \bar{v} = -\underline{v}$ and $\underline{a} < 0 < \bar{a} = -\underline{a}$. We derive the bounds for the two types of motion described earlier.

Stop-and-go trajectories: $\mathbf{v_f = v_0 = 0 = a_f = a_0 = 0}$ Since the expressions for the splines are linear in p_f and p_0 ((4), (5)), without loss of generality we assume $p_0 = 0$. By substituting (5) in (4), we get:

$$p_t^* = (6\frac{t^5}{T_f^5} - 15\frac{t^4}{T_f^4} + 10\frac{t^3}{T_f^3})p_f \quad (7a)$$

$$v_t^* = \underbrace{(30\frac{t^4}{T_f^5} - 60\frac{t^3}{T_f^4} + 30\frac{t^2}{T_f^3})}_{K_1(t)} p_f \quad (7b)$$

$$a_t^* = \underbrace{(120\frac{t^4}{T_f^5} - 180\frac{t^2}{T_f^4} + 60\frac{t^2}{T_f^4})}_{K_2(t)} p_f \quad (7c)$$

Fig. 11 shows the functions K_1 and K_2 for $T_f = 1$. The following lemma is proved by examining the first two derivatives of K_1 and K_2.

Lemma 4.1: The function $K_1 : [0, T_f] \to \mathbb{R}$ is non-negative and log-concave. The function $K_2 : [0, T_f] \to \mathbb{R}$ is anti-symmetric around $t = T_f/2$, concave on the interval $t \in [0, T_f/2)$ and convex on the interval $[T_f/2, T_f]$.

Let $\max_{t\in[0,T_f]} K_1(t) = K_1^*$ and $\max_{t\in[0,T_f]} |K_2(t)| = K_2^*$. These are easily computed thanks to Lemma 4.1. We can now state the feasibility constraints for Stop-and-Go motion. See the appendix for a proof sketch.

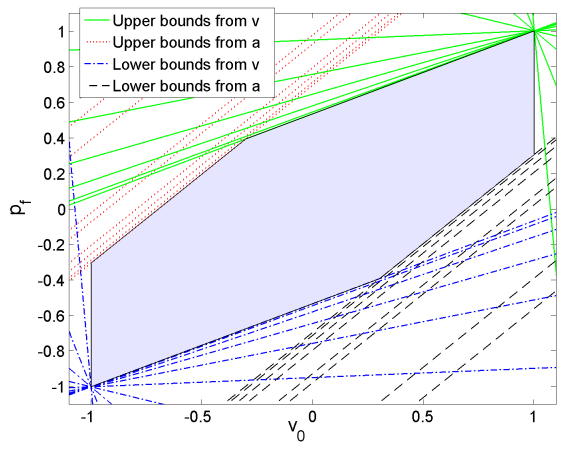

Fig. 5: The upper and lower bounds on p_f due to the acceleration and velocity constraints. Shown as a function of v_0 for $t = 0, 0.1, \ldots, T_f = 1$. The shaded region shows the feasible values of p_f as a function of v_0.

Theorem 4.1 (Stop-and-go feasibility): Given an initial position p_0 (and $v_0 = a_0 = 0$), a maneuver duration T_f, and desired bounds $[\underline{v}, \bar{v}]$ and $[\underline{a}, \bar{a}]$, if $\underline{v}/K_1^* \le p_f - p_0 \le \bar{v}/K_1^*$ and $\underline{a}/K_2^* \le p_f - p_0 \le \bar{a}/K_2^*$ then $v_t^* \in [\underline{v}, \bar{v}]$ and $a_t^* \in [\underline{a}, \bar{a}]$ for all $t \in [0, T_f]$.

Since $\underline{v}, \bar{v}, \underline{a}, \bar{a}, K_1^*, K_2^*$ are all available offline, they can be used as constraints if solving problem (3) offline.

Free end velocities: $\mathbf{a_f = a_0 = 0}$, free v_f. Here too, without loss of generality $p_0 = 0$. Substituting (6) in (4) and re-arranging terms yields the following expression for the optimal translational state:

$$\begin{aligned}
p_t^* &= \left(\frac{90t^5}{240T_f^5} - \frac{90t^4}{48T_f^4} + \frac{30t^3}{12T_f^3}\right)p_f - \left(\frac{90t^5}{240T_f^4} - \frac{90t^4}{48T_f^3} + \frac{30t^3}{12T_f^2} - t\right)v_0 \\
v_t^* &= \underbrace{\left(\frac{90t^4}{48T_f^5} - \frac{90t^3}{12T_f^4} + \frac{30t^2}{4T_f^3}\right)}_{K_3(t)} p_f - \left(\frac{90t^4}{48T_f^4} - \frac{90t^3}{12T_f^3} + \frac{30t^2}{4T_f^2} - 1\right)v_0 \\
a_t^* &= \underbrace{\left(\frac{90t^3}{12T_f^5} - \frac{90t^2}{4T_f^4} + \frac{30t^t}{2T_f^3}\right)}_{K_4(t)} p_f - \left(\frac{90t^3}{12T_f^4} - \frac{90t^2}{4T_f^3} + \frac{30t^t}{2T_f^2}\right)v_0
\end{aligned} \tag{8}$$

Applying the velocity and acceleration bounds $\underline{v} \le v^* \le \bar{v}$ and $\underline{a} \le a^* \le \bar{a}$ to (8) and re-arranging terms yields:

$$\frac{(\underline{v} - (1 - T_f K_3(t))v_0)}{K_3(t)} \le p_f \le \frac{(\bar{v} - (1 - T_f K_3(t))v_0)}{K_3(t)} \; \forall t \in [0, T_f] \tag{9a}$$

$$\underline{a}/K_4(t) + T_f v_0 \le p_f \le \bar{a}/K_4(t) + T_f v_0 \; \forall t \in [0, T_f] \tag{9b}$$

The constraints on p_f are linear in v_0, but parametrized by functions of t. Since t is continuous in $[0, T_f]$, (9) is an infinite system of linear inequalities. Fig. 5 shows these linear bounds for $t = 0, 0.1, 0.2, \ldots, 1 = T_f$ with $\bar{v} = 1 = -\underline{v}, \bar{a} = 2 = -\underline{a}$.

The infinite system can be reduced to 2 inequalities only, as proved in the appendix.

Lemma 4.2: p_f satisfies (9) if it satisfies the following

$$\begin{aligned}
\frac{\underline{v} - (1 - T_f K_3(T_f))v_0}{K_3(T_f)} \le \quad & p_f \quad \le \frac{\bar{v} - (1 - T_f K_3(T_f))v_0}{K_3(T_f)} \\
T_f v_0 + \underline{a}/K_4(t') \le \quad & p_f \quad \le T_f v_0 + \bar{a}/K_4(t')
\end{aligned} \tag{10}$$

where t' is a solution of the quadratic equation $\frac{dK_4(t)}{dt} = 0$, such that $t' \in [0, T_f]$.

The main result follows:

Theorem 4.2 (Free endpoint velocity feasibility): Given an initial translational state $p_0, v_0 \in [\underline{v}, \bar{v}], a_0 = 0$, and a maneuver duration T_f, if p_f satisfies

$$\begin{aligned}
&\frac{\underline{v} - (1 - T_f K_3(T_f))v_0}{K_3(T_f)} \le p_f - p_0 \le \frac{\bar{v} - (1 - T_f K_3(T_f))v_0}{K_3(T_f)} \\
&T_f v_0 + \underline{a}/K_4(t') \le p_f - p_0 \le T_f v_0 + \bar{a}/K_4(t')
\end{aligned} \tag{11}$$

with t' defined as in Lemma 4.2, then $v^*(t) \in [\underline{v}, \bar{v}]$ and $a_t^* \in [\underline{a}, \bar{a}]$ for all $t \in [0, T_f]$ and $p^*(T_f) = p_f$.

V. Control of Quadrotors for Satisfaction of STL Specifications

We are now ready to formulate the mathematical robustness maximization problem we solve for temporal logic planning. We describe it for the Free Endpoint Velocity motion; an almost-identical formulation applies for the Stop-and-Go case with obvious modifications.

Recall the notions of low-rate trajectory $\mathbf{x}$ and high-rate discrete-time trajectory $\mathbf{q}$ defined in Section IV. Consider an initial translational state $x_I = (p_I, v_I)$ and a desired final position p_f to be reached in T_f seconds, with free end velocity and zero acceleration. Given such a pair, the generator of Section IV-A computes a trajectory $q = (p, v, a) : [0, T_f] \to \mathbb{R}^9$ that connects p_I and p_f. By (4), for every $t \in [0, T_f]$, $q(t)$ is a linear function of p_I, p_f and v_I. If the spline $q(\cdot)$ is uniformly sampled with a period of dt', let $H = T_f/dt'$ be the number of discrete samples in the interval $[0, T_f]$. Every $q(dt')$ (sampled point along the spline) is a linear function of p_I, v_I, p_f. Hereinafter, we use $x_I \overset{T_f}{\to} x_f$ as shorthand for saying that x_I is the initial state, and $x_f = (p_f, v_f)$ is the final state with desired end position p_f and end velocity $v_f = v(T_f)$ computed using the spline.

More generally, consider a *sequence* of low-rate waypoints $(x_0 \overset{T_f}{\to} x_1, x_1 \overset{T_f}{\to} x_2, \ldots, x_{N-1} \overset{T_f}{\to} x_N)$, and a sequence $(q^k)_{k=0}^{N-1}$ of splines connecting them and their high-rate sampled versions $\hat{q}^k$ sampled with a period $dt' << T_f$. Then every sample $q^k(i \cdot dt')$ is a linear function of p_{k-1}, v_{k-1} and p_k.

We now put everything together. Write $\hat{\mathbf{q}} = (q^0(0), \ldots, q^{N-1}(Hdt)) \in \mathbb{R}^{9N(H-1)}$, $\mathbf{x} = ((p_0, v_0), \ldots, (p_{N-1}, v_{N-1})) \in \mathbb{R}^{6N}$, and let $L : \mathbb{R}^{6N} \to \mathbb{R}^{9N(H-1)}$ be the linear map between them. In the Stop-and-Go case, this uses all velocities to 0 and uses (7) for positions, and in the free velocity case, L uses (8). The robustness maximization problem is finally:

$$\max_{\mathbf{x}} \quad \tilde{\rho}_{\varphi_s}(L(\mathbf{x})) \tag{12a}$$

$$\text{s.t. } \mathrm{LB}_v(v_{k-1}) \le p_k - p_{k-1} \le \mathrm{UB}_v(v_{k-1}) \forall k = 1, \ldots, N, \tag{12b}$$

$$\mathrm{LB}_a(v_{k-1}) \le p_k - p_{k-1} \le \mathrm{UB}_a(v_{k-1}) \forall k = 1, \ldots, N, \tag{12c}$$

$$\tilde{\rho}_{\varphi_s}(L(\mathbf{x})) \ge \varepsilon \tag{12d}$$

where (12b) and (12c) are the constraints from (11) in Free Endpoint Velocity motion, and Thm. 4.1 in Stop-and-Go motion, with $p_I = p_{k-1}$ and $p_f = p_k$.

Since the optimization variables are only the waypoints $\mathbf{p}$, and not the high-rate discrete-time trajectory, this makes the optimization problem much more tractable. In general, the number N of low-rate waypoints $\mathbf{p}$ is a design choice that requires some mission specific knowledge. The higher N is, the more freedom of movement there is, but at a cost of increased computation burden of the optimization (more constraints and variables). A very small N on the other hand will restrict the freedom of motion and might make it impossible for the resulting trajectory to satisfy the STL specification.

A. Strictification for Continuous time guarantees

In general, if the *sampled* trajectory $\mathbf{q}$ satisfies φ, this does not guarantee that the continuous-time trajectory q also satisfies it. For that, we use [19, Thm. 5.3.1], which defines a *strictification operator* str:$\varphi \mapsto \varphi_s$ that computes a syntactical variant of φ having the following property.

Theorem 5.1: [19] Let dt be the sampling period, and suppose that there exists a constant $\Delta_g \geq 0$ s.t. for all t, $\|q(t) - q(t+dt)\| \leq \Delta_g dt$. Then $\rho_{\varphi_s}(\mathbf{q}) > \Delta_g \Rightarrow (q,0) \models \varphi$. Intuitively, the stricter φ_s tightens the temporal intervals and the predicates μ_k so it is 'harder' to satisfy φ_s than φ. See [19, Ch. 5]. For the trajectory generator g of Section IV-A, Δ_g can be computed given T_f, $\underline{v}, \bar{v}, \underline{a}$ and $\bar{a}$.

We need the following easy-to-prove result, to account for the fact that we optimize a smoothed robustness:

Corollary 5.1.1: Let ε be the worst-case approximation error for smooth robustness. If $\tilde{\rho}_{\varphi_s}(\mathbf{q}) > \Delta_g + \varepsilon$ then $(q,0) \models \varphi$

B. Robust and Boolean modes of solution

The control problem of (12) can be solved in two modes [8]: the *Robust (R) Mode*, which solves the problem until a maximum is found (or some other optimizer-specific criterion is met). And a *Boolean (B) Mode*, in which the optimization (12) stops as soon as the smooth robustness value exceeds ε.

C. Implementation of the control

The controller can be implemented in one of two ways:

One-shot: The optimization of (12) is solved once at time 0 and the resulting trajectories are then used as a plan for the quadrotors to follow. In our simulations, where there are no disturbances, this method is acceptable or when any expected disturbances are guaranteed to be less than $\tilde{\rho}^*$, the robustness value achieved by the optimization.

Shrinking horizon: In practice, disturbances and modeling errors necessitate the use of an online feedback controller. We use a shrinking horizon approach. At time 0, the waypoints are computed by solving (12) and given to the quadrotors to track. Then every T_f seconds, estimates for p, v, a are obtained, and the optimization is solved again, while fixing all variables for previous time instants to their observed/computed values, to generate new waypoints for the remaining duration of the trajectory. For the k^{th} such optimization, we re-use the $k-1^{st}$ solution as an initial guess. This results in a faster optimization that can be run online, as will be seen in the experiments.

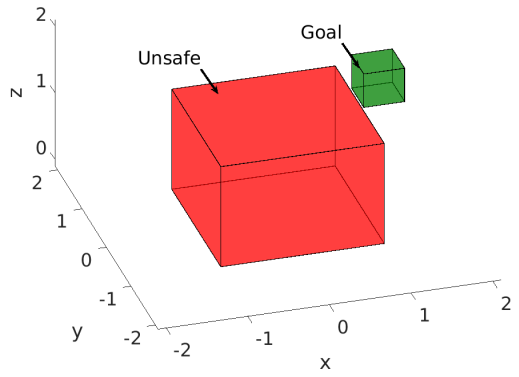

Fig. 6: The environment for the reach-avoid problem.

VI. Simulations Results

We demonstrate the efficiency and the guarantees of our method through multiple examples, in simulation and in real experiments. For the simulations, we consider two case studies: a) A multi-drone reach-avoid problem in a constrained environment, and b) A multi-mission example where several drones have to fly one of two missions in the same environment. We assume a disturbance-free environment, and solve the problem in one shot, i.e. the entire trajectory that satisfies the mission is computed in one go (aka open-loop). Links to the videos for all simulations are in table V in the appendix.

A. Simulation setup

The following simulations were done in MATLAB 2016b, with the optimization formulated in Casadi [20] with Ipopt [21] as the NLP solver. HSL routines [22] were used as internal linear solvers in Ipopt. All simulations were run on a laptop with a quadcore i7-7600 processor (2.8 Ghz) and 16Gb RAM running Ubuntu 17.04. For all simulations, waypoints are separated by $T_f = 1$ second.

B. Multi drone reach-avoid problem

The objective of the drone d is to reach a goal set Goal, while avoiding an unsafe set Unsafe, in the 3D position space. With p denoting the drone's 3D position, the mission for a single drone d is:

$$\varphi^d_{\text{sra}} = \Box_{[0,T]} \neg (p \in \text{Unsafe}) \wedge \Diamond_{[0,T]} (p \in \text{Goal}) \tag{13}$$

This says that for all times in $[0,T]$ the drone must avoid the Unsafe set, and sometime during the interval $[0,T]$, it must visit the Goal set. See Fig. 6.

The Multi drone Reach-Avoid problem adds the requirement that every two drones d, d' must maintain a minimum separation $\delta_{\text{min}} > 0$ from each other: $\varphi^{d,d'}_{\text{sep}} = \Box_{[0,T]}(||p^d - p^{d'}|| \geq \delta_{\text{min}}$. Assuming the number of drones is D, the specification reads:

$$\varphi_{\text{mra}} = \wedge_{d=1}^{D} \varphi^d_{\text{sra}} \bigwedge \wedge_{d=1}^{D} (\wedge_{d' \neq d} \varphi^{d,d'}_{\text{sep}}) \tag{14}$$

The horizon of this formula is $hrz(\varphi_{\text{mra}}) = T$. The robustness of φ_{mra} is upper-bounded by 0.25, which is achievable

if all drones visit the center of the goal set Goal = $[1.5, 2] \times [1.5, 2] \times [1, 1.5]$ - at that time, they would be 0.25m away from the boundaries of Goal - and maintain a minimum distance of 0.25 to Unsafe and each other. We analyze the runtimes and achieved robustness of our controller by running a 100 simulations, each one from different randomly-chosen initial positions of the drones in the free space $X/(\text{Goal} \cup \text{Unsafe}))$.

1) Stop and go trajectories: We show the results of controlling using (12) to satisfy φ_{mra} for the case of Stop-and-Go motion (see Sec.IV-A). with $T = 6$ seconds. Videos of the computed trajectories are available at link 1 (in Boolean mode) and at link 2 (in Robust Mode) in Table V.

Table I shows the run-times for an increasing number of drones D in the Robust and Boolean modes. It also shows the robustness of the obtained optimal trajectories in Robust mode. (We maximize smooth robustness, then compute true robustness on the returned trajectories). As D increases, the robustness values decrease. Starting from the upper bound of 0.25 with 1 drone, the robustness decreases to an average of 0.122 for $D = 5$. This is expected, as more and more drones have to visit Goal within the same time [0,6], while maintaining a pairwise minimum separation of $\delta_{\min}$. As a result, the drones cannot get deep into the goal set, and this lowers the robustness value.

Up to 5 drones, the controller is successful in accomplishing the mission every time. By contrast, for $D > 5$, the controller starts failing, in up to half the simulations. We conclude that for $T = 6$, the optimization can only handle up to 5 drones for this mission, in this environment.

Comparison to MILP-based solution. The problem of maximizing $\rho_{\varphi_{\text{mra}}}(\mathbf{y})$ to satisfy φ_{mra} can be encoded as a Mixed Integer Linear Program (MILP) and solved. The tool BluSTL [8] implements this approach. We compare our runtimes to those of BluSTL for the case $D = 1$. The robustness maximization in BluSTL took $1.2 \pm 0.3s$ (mean $\pm$ standard deviation) and returned a robustness value of 0.25 for each simulation, while the Boolean mode took $0.65 \pm 0.2s$ seconds. (Note we do not include the time it takes BluSTL to encode the problem in these numbers.) Thus our approach out-performs MILP in both modes. For $D > 1$, BluSTL could *not* return a solution with positive robustness. The negative-robustness solutions it did return required several minutes to compute. It should however be noted that BluSTL is a general purpose tool for finding trajectories of dynamical systems that satisfy a given STL specification, while our approach is tailored to the particular problem of controlling multi-rotor robots for satisfying a STL specification.

TABLE I: Stop-and-Go motion. Mean $\pm$ standard deviation for runtimes (in seconds) and robustness values from 100 runs of the reach-avoid problem.

D	Boolean mode	Robust mode	ρ^* (Robust mode)
1	0.078 ± 0.004	0.40 ± 0.018	0.244 ± 0
2	0.099 ± 0.007	1.313 ± 0.272	0.198 ± 0.015
3	0.134 ± 0.015	2.364 ± 0.354	0.176 ± 0.018
4	0.181 ± 0.024	3.423 ± 0.370	0.160 ± 0.031
5	0.214 ± 0.023	7.009 ± 3.177	0.122 ± 0.058

2) Trajectories with free end point velocities: We also solved the multi-drone reach-avoid problem for Free Velocity motion (Section IV-A), with $T = 6$. An instance of the resulting trajectories are available in links 3 and 4 for Boolean mode and at links 5 and 6 for Robust mode in Table V. Table II shows the runtimes for an increasing number of drones D in the Robust and Boolean modes. It also shows the robustness of the obtained optimal trajectories in Robust mode. As before, the achieved robustness value decreases as D increases. Unlike the stop-and-go case, *positive robustness solutions are achieved for all simulations, up to $D = 16$ drones.* This is due to the added freedom of motion between waypoints. This matches the intuition that a wider range of motion is possible when the quadrotors do not have to come to a full stop at every waypoint.

For this case, we did not compare with BluSTL as the there is no easy way to incorporate this formulation in BluSTL.

TABLE II: Free Endpoint Velocity motion. Mean $\pm$ standard deviation for runtimes (in seconds) and robustness values from 100 runs of the reach-avoid problem.

D	Boolean mode	Robust mode	ρ^* (Robust mode)
1	0.081 ± 0.005	0.32 ± 0.18	0.247 ± 0
2	0.112 ± 0.016	0.86 ± 0.15	0.188 ± 0.026
4	0.244 ± 0.017	1.88 ± 0.17	0.149 ± 0.040
5	0.307 ± 0.056	3.48 ± 0.56	0.137 ± 0.018
6	0.439 ± 0.073	9.08 ± 0.85	0.102 ± 0.032
8	0.651 ± 0.042	15.86 ± 2.15	0.0734 ± 0.018
10	0.843 ± 0.077	16.64 ± 1.30	0.051 ± 0.017
12	1.123 ± 0.096	23.99 ± 5.81	0.033 ± 0.003
16	1.575 ± 0.114	32.21 ± 6.25	0.028 ± 0.005

Discussion The simulations show the performance and scalability of our method, finding satisfying trajectories for φ_{mra} for 16 drones in less than 2 seconds on average, while maximizing robustness in 35 seconds. On the other hand, the MILP-based approach does not scale well, and for the cases where it does work, is considerably slower than our approach.

Since the high-level control problem is solved at 1 Hz, the runtimes (in the boolean mode) suggest that we can control up to 2 drones online in real-time without too much computation delay: Stop-and-Go takes an average of 0.099s for 2 drones (Table I), and Free Endpoint Velocity takes an average of 0.11s for 2 drones (Table II). Moreover, when applied online, we solve the optimization in a shrinking horizon fashion, drastically reducing runtimes in later iterations.

C. Multi drone multi mission example

Our method can be applied to scenarios with multiple missions We illustrate this with the following 2-mission scenario: - Mission Pkg: A package delivery mission. The drone(s) d has to visit a Delivery region to deliver a package within the first 10 seconds and then visit the Base region to pick up another package, which becomes available between 10 and 20 seconds later. In STL,

$$\varphi^d_{\text{pkg}} = \Diamond_{[0,10]}(p^d \in \text{Deliver}) \wedge \Diamond_{(10,20]}(p^d \in \text{Base}) \tag{15}$$

- Mission Srv: A surveillance mission. The drone(s) d has to, within 20 seconds, monitor two regions sequentially. In STL,

$$\begin{aligned}\varphi^d_{\text{srv}} = \Diamond_{[0,5]}(p^d \in \text{Zone}_1) \wedge \Diamond_{(5,10]}(p^d \in \text{Zone}_2) \\ \wedge \Diamond_{[10,15)}(p^d \in \text{Zone}_1) \wedge \Diamond_{(15,20]}(p^d \in \text{Zone}_2)\end{aligned} \tag{16}$$

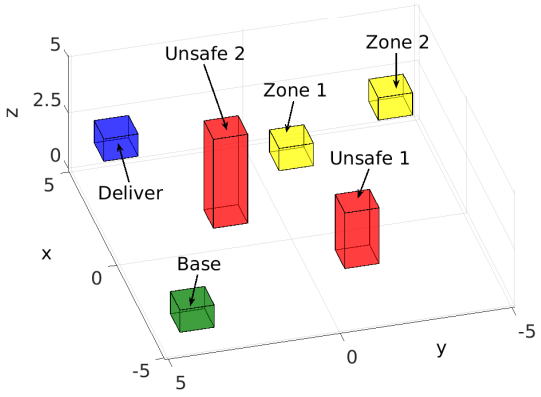

Fig. 7: The environment for the multi-mission problem.

In addition to these requirements, all the drones have to always maintain a minimum separation of $\delta_{\min}$ from each other ($\varphi_{\text{sep}}^{d,d'}$ above), and avoid two unsafe sets Unsafe_1 and Unsafe_2. Given an even number D of drones, the odd-numbered drones are flying mission Pkg, while the even-numbered drones are flying mission Srv. The overall mission specification over all D drones is:

$$\varphi_{\text{x-mission}} = \wedge_{d \text{ Odd}} \varphi_{\text{pkg}}^{d} \bigwedge \wedge_{d \text{ Even}} \varphi_{\text{srv}}^{d} \bigwedge \wedge_{d \le D} \wedge_{d' \neq d} \varphi_{\text{sep}}^{d,d'}$$
$$\bigwedge \wedge_{d \le D} \wedge_{i=1}^{2} \square_{[0,20]} \neg (p^d \in \text{Unsafe}_i)$$

The mission environment is shown in Fig. 7. Note that the upper bound on robustness is again 0.25.

TABLE III: Mean $\pm$ standard deviation for runtimes (in seconds) and robustness values for one-shot optimization. Obtained from 50 runs of the multi-mission problem with random initial positions.

D	Boolean mode	Robust mode	ρ^* (Robust mode)
2	0.33 ± 0.08	4.93 ± 0.18	0.2414 ± 0
4	0.65 ± 0.10	16.11 ± 4.05	0.2158 ± 0.0658
6	2.38 ± 0.28	24.83 ± 7.50	0.1531 ± 0.0497
8	20.82 ± 4.23	32.87 ± 2.26	0.0845 ± 0.0025

Results. We solved this problem for $D = 2, 4, 6, 8$ drones, again with randomly generated initial positions in both Robust and Boolean modes. Videos of the resulting trajectories are in links 7-10 of Table V. The runtimes increase with the number of drones, as shown in Table III.The runtimes are very suitable for offline computation. For online computation in a shrinking horizon fashion, they impose an update rate of at most 1/2 Hz (once every 2 seconds). In general, in the case of longer horizons, this method can serve as a one-shot planner.

For $D > 8$, the optimization could not return a solution that satisfied the STL specification. This is likely due to the small size of the sets that have to be visited by all drones in the same time interval, which is difficult to do while maintaining minimum separation (see Fig. 7). So it is likely the case that no solution exists, which is corroborated by the fact that maximum robustness decreases as D increases (Table III).

VII. Experiments on real quadrotors

We evaluate our method on Crazyflie 2.0 quadrotors(Fig. 2). Through these experiments we aim to show that: a) the velocity and acceleration constraints from Section IV-B indeed ensure that the high-rate trajectory $\mathbf{y}$ generated by robustness maximization is dynamically feasible and can be tracked by the MPC position controller, and b) our approach can control the quadrotors to satisfy their specifications in a real-time closed loop manner. A real-time playback of the experiments is in the links 11-16 of Table V in the appendix.

A. Experimental Setup

The Crazyflies are controlled by a single laptop running ROS and MATLAB. For state estimation, a Vicon motion capture system gave us quadrotor positions, velocities, orientations and angular velocities. In order to control the Crazyflies, we: a) implemented the robustness maximization using Casadi in MATLAB, and interfaced it to ROS using the MATLAB-ROS interface provided by the Robotics Toolbox, b) implemented a Model Predictive Controller (MPC) using the quadrotor dynamics linearized around hover for the position controller, coded in C using CVXGEN and ROS, c) modified the ETH tracker in C++ to work with ROS. The Crazyflie has its own attitude controller flashed to the onboard microcontroller. The robustness maximizer runs at 1Hz, the trajectory generator runs at 20Hz, the position controller runs at 20Hz and the attitude controller runs at 50Hz.

B. Online real-time control

We fly the reach-avoid problem (in both Stop-and-Go and Free Endpoint Velocity modes), for one and two drones, with $T = 6$ seconds, and with a maneuver duration T_f set to 1 second. The controller operates in Boolean mode.

The shrinking horizon approach of Sec. V is used with a re-calculation rate of 1 Hz. This approach can be implemented in an online manner in real-time when the computation time for the high-level optimization (12) is much smaller than the re-calculation rate of the optimization, as it is in the cases we consider here.

We repeat each experiment multiple times. For every run, the quadrotors satisfied the STL specification of (14). The runtimes are shown in Table IV. Using the optimal solution at the previous time step as the initial solution for the current time step results in very small average runtimes per time-step. This shows that our method can be easily applied in a real-time closed-loop manner.

TABLE IV: Average runtime per time-step (in seconds) of shrinking horizon robustness maximization in Boolean mode. These are averaged over 5 repetitions of the experiment from the same initial point, to demonstrate the reproducibility of the experiments.

D	Stop-and-Go	Free Endpoint Velocity)
1	0.052	0.065
2	0.071	0.088

We observed that for more than 2 quadrotors, the online delay due to the optimization and the MATLAB-ROS interface (the latter takes up to $10ms$ for receiving a state-estimate and publishing a waypoint command) is large enough that the quadrotor has significantly less than T_f time to execute the maneuver between waypoints, resulting in trajectories that sometimes do not reach the goal state. Videos are in links 11-13 in table V.

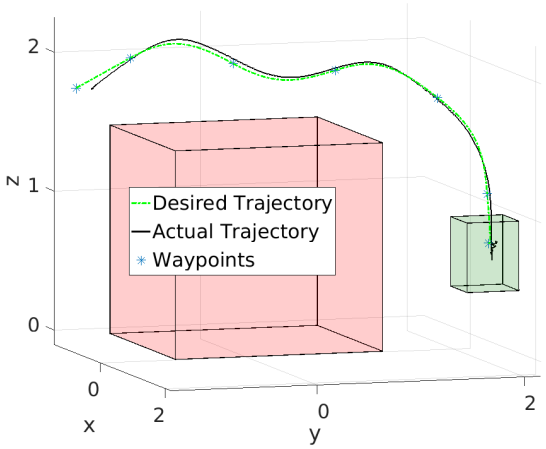

Fig. 8: The desired and actual trajectories for a Crazyflie 2.0 flying the reach-avoid problem with closed-loop control. The unsafe set is in red and the goal set is in green (Color in online version). The tracking is near-perfect, showing the dynamical feasibility of the selected waypoints.

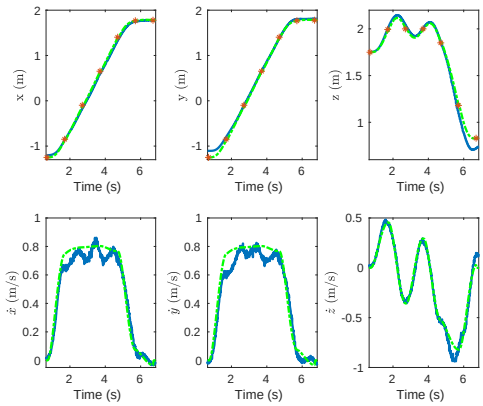

Fig. 9: The desired (dashed green) and actual (blue) positions and velocities along the 3 axes.

C. Validating the dynamic feasibility of generated trajectories.

Figs. 8 and 9 shows the tracking of the positions and velocities commanded by the spline. The near-perfect tracking in x,y axes and satisfactory tracking in the z axis shows that we are justified in assuming that imposing velocity and acceleration bounds on the robustness maximizer produces dynamically feasible trajectories that can be tracked by the position and attitude controllers. Note that the tracking in z is not perfect due to a combination of modeling error in the model for the MPC and the lack of thrust compensation as the batteries onboard the Crazyflies deplete.

D. Offline planning for multiple drones

Our approach can be used as an *offline* path planner. Specifically, we solve the problem (12) offline for Free Endpoint Velocity motion, and use the solution *low-rate* trajectory $\mathbf{x}$ as waypoints. Online, we run the trajectory generator of Sec. IV-A (and lower-level controllers) to navigate the Crazyflies between the waypoints in a shrinking horizon fashion. We did this for all 8 Crazyflies at our disposal, and we expect it is possible to support significantly more Crazyflies, since the online computation (for the individual position and attitude controllers of the drones) is completely independent for the various drones. A video of this experiment is available in the links of Table V.

VIII. Conclusions and Future Work

We presented a method for generating trajectories for multiple drones to satisfy a given STL requirement. The correctness of satisfaction is guaranteed for the continuous-time behavior of the resulting trajectories, and they can be tracked nearly perfectly by lower level position and attitude controllers. Through simulations, as well as experiments on actual quadrotors, we show the applicability of our method as a real-time high-level controller in a hierarchical control scheme. We also show that our method is computationally tractable and scales well for problems involving up to 16 drones.

While the examples in this paper show the good performance of our method, the method itself is not without limitations: a) The high-level optimization (12) at the heart of this approach is a non-convex problem and the solvers used guarantee convergence only to a local optima, which may have a negative robustness value. Parallel instances of the solver with different initial starting points can alleviate this problem in practice. b) The approach still cannot scale to control a large number of drones in an online and real-time manner. c) Finally, this method is only applicable to multi-rotor aerial robots (e.g. quad/hex rotors) and cannot be applied to non-holonomic ground robots (like autonomous cars) while obtaining the same guarantees on dynamic feasibility and satisfaction of specifications. Ongoing work is on extending our method to STL formulae with unbounded time horizons through a receding horizon approach, as well as addressing some of the existing limitations of this method.

Acknowledgements

The authors would like to acknowledge Bernd Pfrommer for his initial loan of the Crazyflies as well as help setting them up, Thejas Kesari for his help with the videos, and Jalaj Maheshwari for his help with figures.

References

[1] X. Ma, Z. Jiao, and Z. Wang, "Decentralized prioritized motion planning for multiple autonomous uavs in 3d polygonal obstacle environments," in *International Conference on Unmanned Aircraft Systems*, 2016.

[2] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[3] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *IEEE Conference on Decision and Control*, 2016.

[4] I. Saha, R. Rattanachai, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe ltl specifications," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[5] J. A. DeCastro, J. Alonso-Mora, V. Raman, and H. Kress-Gazit, "Collision-free reactive mission and motion planning for multi-robot systems," in *Robitics Research, Springer Proceedings in Advanced Robotics*, 2017.

[6] A. Desai, I. Saha, Y. Jianqiao, S. Qadeer, and S. A. Seshia, "Drona: A framework for safe distributed mobile robotics," in *ACM/IEEE International Conference on Cyber-Physical Systems*, 2017.

[7] W. Honig, T. K. Satish Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints," in *International Conference on Automated Planning and Scheduling*, 2016.

[8] V. Raman, A. Donze, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 81–87.

[9] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *Allerton conference*, September 2015.

[10] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," in *IEEE Conference on Control Technology and Applications,*, 2017.

[11] H. Abbas, G. E. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, 2013.

[12] H. Abbas and G. Fainekos, "Computing descent direction of MTL robustness for non-linear systems," in *American Control Conference*, 2013.

[13] O. Maler and D. Nickovic, *Monitoring Temporal Properties of Continuous Signals*. Springer Berlin Heidelberg, 2004.

[14] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proceedings of the International Conference on Formal Modeling and Analysis of Timed Systems*, 2010.

[15] G. Fainekos and G. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, 2009.

[16] H. Abbas and G. Fainekos, "Computing descent direction of mtl robustness for non-linear systems," in *American Control Conference (ACC)*, 2013, pp. 4411–4416.

[17] T. Luukkonen, "Modelling and control of quadcopter," in *Independent research project in applied mathematics*. Aalto University School of Science, 2011.

[18] M. W. Mueller, M. Hehn, and R. DÁndrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," in *IEEE Transactions on Robotics*, 2015.

[19] G. Fainekos, "Robustness of temporal logic specifications," Ph.D. dissertation, University of Pennsylvania, 2008. [Online]. Available: http://www.public.asu.edu/~gfaineko/pub/fainekos_thesis.pdf

[20] J. Andersson, "A General-Purpose Software Framework for Dynamic Optimization," PhD thesis, Arenberg Doctoral School, KU Leuven, 2013.

[21] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, 2006.

[22] "Hsl. a collection of fortran codes for large scale scientific computation." http://www.hsl.rl.ac.uk, accessed: 2017-10-05.

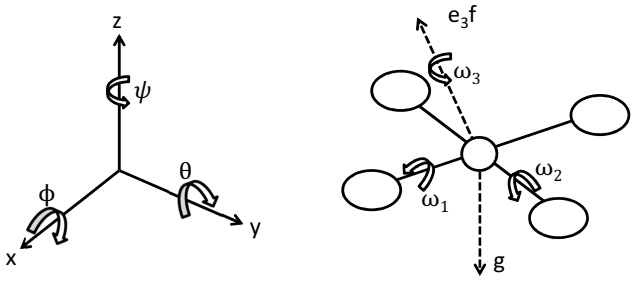

Fig. 10: World frame and rotation angles (left) and quadrotor frame with angular rates (right). g is gravitational force. Figure adapted from Fig. 2 in [18].

IX. Appendix

A. Quadrotor dynamics

Multi-rotor dynamics have been studied extensively in the literature [17], [18], and we closely follow the conventions of [18] Fig. 10 illustrates the following definitions. The quadrotor has 6 degrees of freedom. The first three, (x, y, z), are the linear position of the quadrotor in $\mathbb{R}^3$ expressed in the world frame. We write $p = (x, y, z)$ for position. The remaining three are the rotation angles (ϕ, θ, ψ) of the quadrotor body frame with respect to the fixed world frame. Their first time-derivatives, $\omega_1, \omega_2, \omega_3$, resp., are the quadrotor's angular velocities. We also write $v = \dot{p}$ for linear velocity and $a = \dot{v}$ for acceleration. If we let $\mathbf{R}$ denote the rotation matrix [17] that maps the quadrotor frame to the world frame at time t, $e_3 = [0, 0, 1]'$, and $h \in \mathbb{R}$ be the input to the system, which is the total thrust normalized by the mass of the quadrotor, then the dynamics are given by

$$\begin{aligned} \ddot{p} &= \mathbf{R} e_3 h + [0, 0, 9.81]' \\ \dot{\mathbf{R}} &= \mathbf{R} \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \end{aligned} \tag{17}$$

B. Links to videos

Table V has the links for the visualizations of all simulations and experiments performed in this work.

C. Dyanmic feasibility proofs for Section IV-B

Stop-and-go motion. Dynamical feasibility for velocities implies that $v_t^* \in [\underline{v}, \bar{v}] \, \forall t \in [0, T_f]$, so by (7), $\underline{v} \leq K_1(t) p_f \leq \bar{v}$. Similarly for accelerations, $\underline{a} \leq K_2(t) p_f \leq \bar{a}$. By non-negativity of K_1 and negativity of $\underline{v}$, the velocity constraints are equivalent to:

$$\underline{v}/K_1^* \leq p_f \leq \bar{v}/K_1^* \tag{18}$$

Similarly for acceleration, the constraints are

$$\underline{a}/K_2^* \leq p_f \leq \bar{a}/K_2^* \tag{19}$$

This establishes the result for $p_0 = 0$. For the general case, simply replace p_f by $p_f - p_0$ and apply the $p_0 = 0$ result. Through the decoupling of axes, this result holds for all 3 jerk axes.

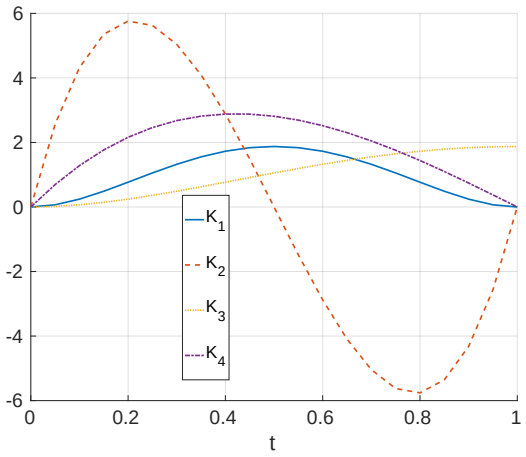

Fig. 11: The functions K_1 to K_4 for $T_f = 1$.

Free velocity motion. *Proof of Lemma 4.2.* We first prove the upper bound of the first inequality, derived from velocity bounds. The lower bound follows similarly. First, note that the upper bounds $v_0 \mapsto (\bar{v} - (1 - T_f K_3(t)) v_0)/K_3(t)$ are lines that intersect at $v_0 = \bar{v}$ for all t. Indeed, substituting $v_0 = \bar{v}$ in the upper bound yields $\bar{v} - (1 - T_f K_3(t))\bar{v})/K_3(t) = T_f \bar{v}$ regardless of t. See Fig. 5. Thus the least upper bound is the line with the smallest intercept with the y-axis. Setting $v_0 = 0$ in 9, the intercept is $\bar{v}/K_3(t)$. This is smallest when $K_3(t)$ is maximized. Since K_3 is monotonically increasing ($\frac{dK_3(t)}{dt} \geq 0$), $K_3(t)$ is largest at $t = T_f$. Thus the least upper bound on p_f is $(\bar{v} - (1 - T_f K_3(T_f)) v_0)/K_3(T_f)$.

We now prove the upper bound for the second inequality, derived from acceleration bounds. The lower bound follows similarly. The upper bounds $t \mapsto \bar{a}/K_4(t) + T_f v_0$ have the same slope, T. See Fig. 5. The least upper bound therefore has the smallest intercept with the y-axis, which is $\bar{a}/K_4(t)$. The smallest intercept, yielding the smallest upper bound, occurs at the t that maximizes K_4. Since $K_4(t)$ is concave in t in the interval $[0, T_f]$, it is maximized at the solution of $\frac{dK_4(t)}{dt} = 0$. This concludes the proof. Refer to Fig. 5 for the intuition behind this proof.

TABLE V: Links for the videos for simulations and experiments. Here, Sim. stands for Matlab simulations, CF2.0 for experiments on the Crazyflies. Stop-go and vel. free are the two modes of operation of the trajectory generator, and B (R) is the Boolean (Robust) mode of solving the control problem. Shr. Hrz. stands for the shrinking horizon mode for online control. The reader is advised to make sure while copying the link that special characters are not ignored when pasted in the browser.

Link number	Platform	Mode	Specification	Drones (D)	Link
1	Sim.	One-shot (B), stop-go	φ_{mRA}	1,2,4,5	http://bit.ly/RABstopgo
2	Sim.	One-shot (R), stop-go	φ_{mRA}	1,2,4,5	http://bit.ly/RARstopgo
3	Sim.	One-shot (B), vel. free	φ_{mRA}	1,2,4,5,6	http://bit.ly/RAB1to6varvel
4	Sim.	One-shot (B), vel. free	φ_{mRA}	8,10,12,16	http://bit.ly/RAB8to16varvel
5	Sim.	One-shot (R), vel. free	φ_{mRA}	1,2,4,5,6	http://bit.ly/RAR1to6varvel
6	Sim.	One-shot (R), vel. free	φ_{mRA}	8,10,12,16	http://bit.ly/RAR8to16varvel
7	Sim.	One-shot (R), vel. free	$\varphi_{x-mission}$	2	http://bit.ly/multi2mission
8	Sim.	One-shot (R), vel. free	$\varphi_{x-mission}$	4	http://bit.ly/multi4mission
9	Sim.	One-shot (R), vel. free	$\varphi_{x-mission}$	6	http://bit.ly/multi6mission
10	Sim.	One-shot (R), vel. free	$\varphi_{x-mission}$	8	http://bit.ly/multi8mission
11	CF2.0	Shr.Hrz (B), vel. free	φ_{sRA}	1	http://bit.ly/varvel1
12	CF2.0	Shr.Hrz (B), vel. free	φ_{mRA}	2	http://bit.ly/varvel2
13	CF2.0	Shr.Hrz (B), stop-go	φ_{sRA}	1	http://bit.ly/stopgo1
14	CF2.0	One-shot (R), vel. free	φ_{mRA}	4	http://bit.ly/varvel4
15	CF2.0	One-shot (R), vel. free	φ_{mRA}	6	http://bit.ly/varvel6
16	CF2.0	One-shot (R), vel. free	φ_{mRA}	8	http://bit.ly/varvel8