gatekeeper: Online Safety Verification and Control for Nonlinear Systems in Dynamic Environments

Devansh Ramgopal Agrawal, Ruichang Chen and Dimitra Panagou

***Abstract*—This paper presents the gatekeeper algorithm, a real-time and computationally-lightweight method that ensures that trajectories of a nonlinear system satisfy safety constraints despite sensing limitations. gatekeeper integrates with existing path planners and feedback controllers by introducing an additional verification step to ensure that proposed trajectories can be executed safely, despite nonlinear dynamics subject to bounded disturbances, input constraints and partial knowledge of the environment. Our key contribution is that (A) we propose an algorithm to recursively construct safe trajectories by numerically forward propagating the system over a (short) finite horizon, and (B) we prove that tracking such a trajectory ensures the system remains safe for all future time, i.e., beyond the finite horizon. We demonstrate the method in a simulation of a dynamic firefighting mission, and in physical experiments of a quadrotor navigating in an obstacle environment that is sensed online. We also provide comparisons against the state-of-the-art techniques for similar problems.**

***Index Terms*—Collision Avoidance, Motion and Path Planning, Aerial Systems: Applications, Safety-Critical Control**

I. INTRODUCTION

Designing autonomous systems with strict guarantees of safety is still a bottleneck to deploying such systems in the real world. Safety is often posed as requiring the system's trajectories to lie within a set of allowable states, called the safe set. In this paper, we consider the case where the safe set is not known a priori, but is rather built on-the-fly via the system outputs (sensor measurements). More specifically, we consider the problem where a robot with limited sensing capabilities (hence limited information about the environment) has to move while remaining safe under some mild assumptions on the evolution of the environment, to be stated in detail below.

Navigating within a non-convex safe set is often tackled by path planning techniques [1]–[4]. Typically a planner generates reference (or nominal) trajectories based on a simplified (e.g., linearized or kinematic) model of the system. However, the reference trajectories may not be trackable by the actual nonlinear system dynamics, and as a result safety constraints may be violated. Furthermore, when trajectories are planned over finite horizons, without recursive feasibility guarantees a planner may fail to find a trajectories, leading to safety violations. This is particularly relevant and challenging when operating in dynamic environments.

In this paper, we propose a technique to bridge path planners (that can solve the nonconvex trajectory generation problem) and controllers (that have robust stability guarantees) in a way that ensures safety. gatekeeper takes inspiration from [5] and [6], both of which also employ the idea of a backup planner/controller. Conceptually, a backup controller is a feedback controller that drives the system to a set of states that are safe (referred to as the backup safe set), and keeps the system in this set. For example, for a quadrotor navigating in an environment with static obstacles, a backup controller could be one that causes the quadrotor to hover in place.

In gatekeeper, the idea is that given a nominal trajectory generated by the path planner (potentially unsafe and/or not dynamically feasible) we construct a "committed trajectory" using a backup controller. To do this, at each iteration of gatekeeper, we simulate a controller that tracks the nominal trajectory upto some switching time, and executes the backup controller thereafter. The trajectory with the largest switching time that is valid (as defined in Def 9) becomes the committed trajectory. Thus, each committed trajectory is, by construction, guaranteed to be defined, feasible, and safe for all future time. The controller always tracks the last committed trajectory, thereby ensuring safety. This paper's key contribution is the algorithm to construct such committed trajectories, and a proof that the proposed approach ensures the closed-loop system remains safe. Furthermore, we explicitly account for robustness against disturbances and state-estimation error since naive approaches to robustification can lead to undesired deadlock. The overall algorithm is computationally efficient compared to similar methods, e.g., Model Predictive Control (MPC). In our simulations VI, gatekeeper was approximately 3-10 times faster than MPC.

In summary, this work has the following contributions:

- A framework to bridge path planners with tracking controllers in order to convert nominal/desired trajectories (generated by the path planner) into committed trajectories that the tracking controller can track safely.
- A formal proof that the robotic system will remain safe for all future time under the stated assumptions.

In particular, the new contributions of this version with respect to the conference paper [7] are:

- Theoretical: A robustification of the verification conditions in [7] to also account for errors introduced by

The authors would like to acknowledge the support of the National Science Foundation (NSF) under grant no. 1942907.

Devansh R Agrawal is with the Department of Aerospace Engineering, University of Michigan; Ruichang Chen is with the Department of Electrical and Computer Engineering; Dimitra Panagou is with the Department of Robotics and the Department of Aerospace Engineering, University of Michigan, Ann Arbor, USA. {devansh, chenrc, dpanagou}@umich.edu

state estimation. Furthermore, we have simplified the verification conditions.
- Experimental: A demonstration of the algorithm applied to quadrotors flying through an unknown environment, constructing a map of the environment online, and filtering human pilot commands to ensure collision avoidance.

A worked analytic example is provided in the appendix, to help illustrate the key concepts of the paper.

Paper Organization: In section II we review a few of the leading paradigms for safety-critical path planning and control. In section III, we describe the key idea underpinning `gatekeeper`. In sections IV, V we formally define the problem and describe our proposed solution. Finally, in section VI simulations and experiments are used to demonstrate the method, and specific implementation details are discussed.

Notation: Let $\mathbb{N} = \{0, 1, 2, ...\}$, and $\mathbb{R}, \mathbb{R}_{>0}, \mathbb{R}_{\geq 0}$ denote the set of reals, positive reals, and non-negative reals. Lowercase $t \in \mathbb{R}$ is used for specific time points, while uppercase $T \in \mathbb{R}$ is for durations. $\|\cdot\|$ refers to the vector 2-norm. Closed balls are denoted $\mathbb{B}(r) = \{x : \|x\| \leq r\}$. $A \ominus B$ and $A \oplus B$ are the Pontryagin set difference and the Minkowski sum of sets A, B. A function $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is class $\mathcal{K}$ if it is continuous, strictly increasing and $\alpha(0) = 0$. $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a class $\mathcal{KL}$ function if it is continuous, for each $t \geq 0$, $\beta(\cdot, t)$ is class $\mathcal{K}$, and for each $r > 0$, $\beta(r, \cdot)$ is strictly decreasing and $\lim_{t\to\infty} \beta(r, t) = 0$. See also Table I.

II. RELATED WORK

A wide range of architectures and approaches have been proposed to tackle safety-critical planning and control, especially when the environment is sensed online. A generic perception planning and control stack is depicted in Fig. 1a.

One approach is to encode the safety constraints in the path-planning module. In this case, the world is represented using a grid-world, or through simplified geometric primitives like obstacle points, or planes to depict the walls. From this representation, a path is generated to avoid obstacles using, for instance, grid-search techniques [8] or sampling [2]. These paths can then be modified to avoid the obstacles, e.g. [9]. However since the path was generated without considering the closed-loop behavior of the nonlinear dynamics of the system and the controller, the robot may not execute the planned path exactly. Therefore, safety may not be guaranteed.

A second approach is to encode the safety constraints at the controller. In recent years, methods based on Control Barrier Functions (CBFs) [10] have been developed to ensure that a system remains within a specified safe set while tracking a desired control input. These methods however require the safe set to be known apriori, represented by a scalar function $h : \mathcal{X} \to \mathbb{R}$ that is continuously differentiable, and satisfies an invariance condition (see for e.g. Def. 2 of [10]). For certain classes of systems and safe sets, constructive methods exist to design h, but these do not handle time-varying or multiple safety conditions well [11]–[15]. Alternatively, offline and computationally expensive methods based on Hamilton-Jacobi reachability [10], [16] or learning-based [17] can be used. However, when the environment is sensed online (and therefore the safe set is constructed online), the assumptions of a CBF cannot be verified, and in practice the controllers fail to maintain safety.

The third common approach is to encode the safety constraints jointly between the controller and the path planner. For example, Model Predictive Control (MPC) techniques plan trajectories considering the dynamics of the robotic system, and also determine a control input to track the trajectory. Various versions of this basic concept have been demonstrated, for example in [5], [18]–[20]. However, the nonlinearity of the robot dynamics and the nonconvexity of the environment, guranteeing convergence, stability or recursive feasibility is challenging. To handle the interaction between path planners and controllers, multirate controllers [15], [19] have also been proposed. These methods can exploit the differential flatness properties of the system to provide necessary theoretical guarantees, although the resulting mixed-integer problem can be expensive for clutter/complicated environments. In general, these methods solve the path planning problem and the control problem separately, but impose additional constraints on each to guarantee that the robot will remain safe. Unfortunately, this assumes some structure of each the path planner and the controller, limiting the applicability.

There is also a growing literature on end-to-end learning based methods for safe perception, planning, and control. See for e.g., [21], [22] and references within. These methods can perform well in scenarios that they have been trained on, but do not provide guarantees of performance or safety in scenarios beyond which they have been trained.

The idea of backup planners/controllers has been introduced recently to address some of the challenges listed above. In [5], a backup trajectory is constructed using a linear model to ensure the trajectory lies within the known safe set at any given time. However, since the backup trajectory was generated using simplified dynamics, the nonlinear system and controller may not be able to execute this trajectory. This can lead to safety violations. In [6], safety is guaranteed by blending the nominal control input with a backup control input. The mixing fraction is determined by numerically forward propagating the backup controller. However, since the nominal and backup control inputs are mixed, the nominal trajectory is never followed exactly, even when it is safe to do so. By combining elements from both methods in a novel manner, `gatekeeper` addresses the respective limitations of each, without requiring the path planner and controller to be co-designed.

III. MOTIVATING EXAMPLE AND METHOD OVERVIEW

We present an example to illustrate the key concepts in this paper, and challenges when dealing with dynamic environments and limited sensing. A common wildfire fire-fighting mission is the "firewatch" mission, where a helicopter is deployed to trace the fire-front, the outer perimeter of the wildfire. The recorded GPS trace is then used to create a map of the wildfire, which is then used to efficiently deploy appropriate resources. Today, the helicopters used in the firewatch mission are human-piloted, but in this example,

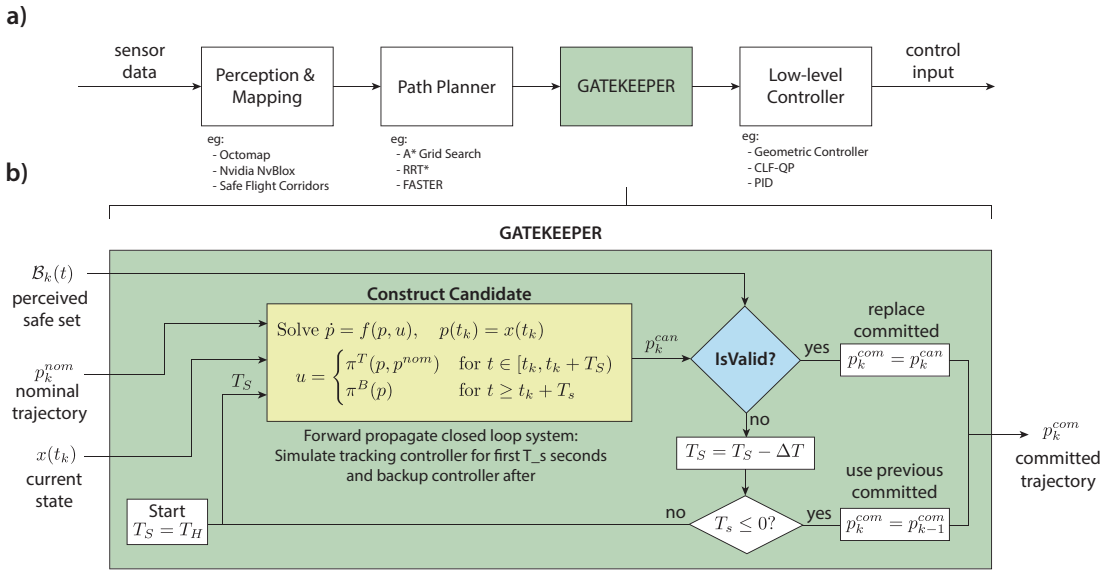

Fig. 1. Block Diagram describing the `gatekeeper` algorithm. (a) shows that `gatekeeper` is an additional module that fits within the common perception-planning-control stack of a robotic system. (b) is a pictorial representation of Algorithm 1.

we design an autonomous controller for a UAV to trace the fire-front without entering or being surrounded by the fire. Fig. 2 depicts the notation used in this paper.

The fire is constantly evolving, and expanding outwards. Thus the safe set, the set of states located outside the fire, is a time-varying set denoted $\mathcal{S}(t)$. Since the rate of spread of fire is different at each location, (it depends on various environmental factors like slope, vegetation and wind [23], [24]), the evolution of the safe set $\mathcal{S}(t)$ is unknown.

That said, it is often possible to bound the evolution of $\mathcal{S}(t)$. In this example, we assume the maximum fire spread rate is known. To operate in this dynamic environment, the UAV makes measurements, for example thermal images that detect the fire-front. However, due to a limited field-of-view, only a part of the safe set can be measured.

The challenge, therefore, is to design a controller for the nonlinear system that uses the on-the-fly measurements to meet mission objectives, while ensuring the system state $x(t)$ remains within the safe set at all times, i.e.,

$$x(t) \in \mathcal{S}(t), \ \forall t \geq t_0. \tag{1}$$

Since $\mathcal{S}$ is unknown, verifying (1) directly is not possible. We ask a related question: given the information available upto some time t_k, does a candidate trajectory $p_k^{can}(t)$ satisfy

$$p_k^{can}(t) \in \mathcal{B}_k(t), \ \forall t \geq t_k, \tag{2}$$

where $\mathcal{B}_k(t)$ is the *perceived* safe set for any time $t \geq t_k$ constructed using the sensory information available up to t_k only. If we assume the perception system provides a reliable estimate of a subset of the safe set, $\mathcal{B}_k(t) \subset \mathcal{S}(t) \ \forall t \geq t_k$, then any candidate trajectory satisfying (2) will also satisfy $p_k^{can}(t) \in \mathcal{S}(t)$. However, since the check in (2) needs to be performed over an infinite horizon $t \geq t_k$, it still cannot be implemented. *A key contribution of this paper is to show how we can perform this check by verifying only a finite horizon.*

We propose the following: at each iteration, we construct a *candidate trajectory* and check whether the candidate satisfies (2). If so, the candidate trajectory becomes a *committed trajectory*. The controller always tracks the last committed trajectory, thus ensuring safety.

Referring back to the firewatch mission, if the UAV is able to fly faster than the maximum spread rate of the fire, a safe course of action could be to simply fly perpendicular to the firefront, i.e., radially from the fire at a higher speed than the maximum fire spread rate. This maneuver is an example of a *backup controller*, since it encodes the idea that if the system state reaches a backup set $\mathcal{C}_k(t_{kB})$ at some time $t_{kB} \geq t_k$, then the backup controller π_B^k ensures that $x(t) \in \mathcal{C}_k(t)$ for all $t \geq t_{kB}$. Note, we use the notation $\mathcal{C}_k(t)$ to highlight that the backup set $\mathcal{C}_k$ could be a time-varying set.

In the firewatch mission, π_B^k is controller that makes the UAV fly perpendicular to the firefront, and $\mathcal{C}_k(t)$ is the set of states that are "sufficiently far from fire, with a sufficiently high speed perpendicular to the fire." A worked example with exact expressions for $\mathcal{S}(t), \mathcal{B}_k(t), \mathcal{C}_k(t)$ is provided in the appendix. Since the fire is constantly expanding, the $\mathcal{C}_k(t)$ set is also changing in time: the set of safe states needs to continue moving outwards radially. Furthermore, at each k, the backup controller and set can be a different, so we index these by k as well.

Using the notion of backup controllers, we can find a sufficient condition for (2) that only requires finite horizon trajectories:

$$\begin{cases} p_k^{can}(t) \in \mathcal{B}_k(t) & \text{if } t \in [t_k, t_{kB}) \\ p_k^{can}(t_{kB}) \in \mathcal{C}_k(t_{kB}) \end{cases} \tag{3}$$

$$\implies \begin{cases} p_k^{can}(t) \in \mathcal{S}(t) & \text{if } t \in [t_k, t_{kB}) \\ p_k^{can}(t) \in \mathcal{S}(t) & \text{if } t \in [t_{kB}, \infty) \end{cases} \tag{4}$$

$$\iff p_k^{can}(t) \in \mathcal{S}(t) \ \forall t \geq t_k \tag{5}$$

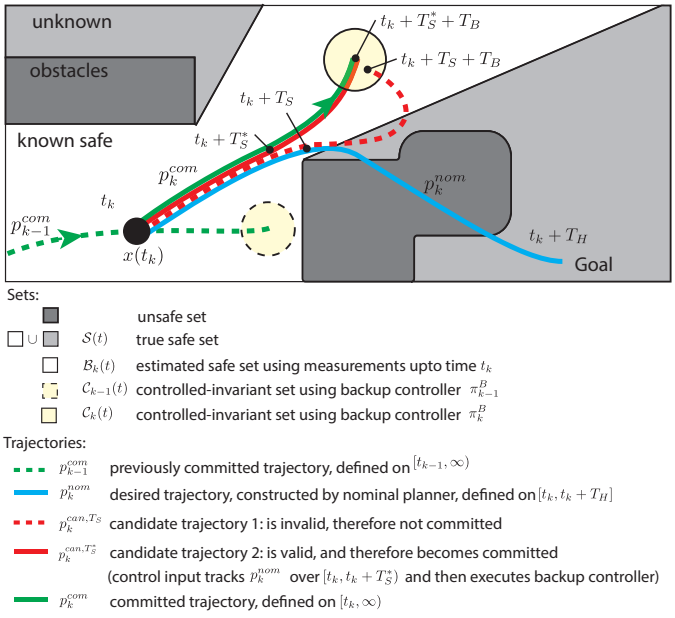

Fig. 2. Notation used in this paper. The nominal planner can plan trajectories into unknown spaces, but `gatekeeper` ensures the committed trajectory lies within the estimated safe sets, for all future time.

for any $t_{kB} \geq t_k$, provided (I) $\mathcal{B}_k(t) \subset \mathcal{S}(t)$, (II) $\mathcal{C}_k(t) \subset \mathcal{S}(t)\ \forall t \geq t_{kB}$, and (III) for $t \geq t_{kB}$ the control input to the candidate trajectory is π_B^k. These conditions can be verified easily: (I) *is the assumption that the perception system correctly identifies a subset of the safe set*, (II) is the defining property of a backup set, and (III) will be true based on how we construct the candidate trajectory.

Notice that in (3), we only need to verify the candidate trajectory over a finite interval $[t_k, t_{kB}]$, but this is sufficient to proving that the candidate is safe for all $t \geq t_k$.

In the following sections, we formalize the `gatekeeper` as a method to construct safe trajectories that balance between satisfying mission objectives and ensuring safety.

IV. PROBLEM FORMULATION

We consider two types of systems: (A) a nominal system, with perfect state information and without disturbances, and (B) a perturbed estimate-feedback system, where an observer is used to estimate the state from sensor measurements, and there are bounded disturbances on both the system dynamics and the measurements.

A. Nominal System Description

Consider a nonlinear system,

$$\dot{x} = f(x, u) \tag{6}$$

where $x \in \mathcal{X} \subset \mathbb{R}^n$ is the state and $u \in \mathcal{U} \subset \mathbb{R}^m$ is the control input. The function $f : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ is assumed locally Lipschitz.

Given a control policy $\pi : [t_0, \infty) \times \mathcal{X} \to \mathcal{U}$ and an initial condition $x(t_0) = x_0 \in \mathcal{X}$, the initial-value problem describing the (nominal) closed-loop system is:

$$\dot{x} = f(x, \pi(t, x)), \qquad x(t_0) = x_0. \tag{7}$$

Table I. Notation

Symbol	Definition
Indices:	
k	Planning iteration index
Time Points:	
t_k	Start time of iteration k
t_{kS}	Switch time $t_{kS} = t_k + T_S$
t_{kB}	Forecast time $t_{kB} = t_{kS} + T_B$
Sets:	
$\mathcal{X} \subset \mathbb{R}^n$	State space
$\mathcal{U} \subset \mathbb{R}^m$	Control input space
$\mathcal{S}(t) \subset \mathcal{X}$	Safe set at time t
$\mathcal{B}_k(t) \subset \mathcal{X}$	Perceived safe set at time t based on measurements upto time $t_k \leq t$
$\mathcal{C}_k(t) \subset \mathcal{X}$	k-th controlled-invariant set
Controllers:	
π_T	Trajectory tracking controller, $\pi_T : \mathcal{X} \times \mathcal{X} \to \mathcal{U}$
π_B	Backup controller, $\pi_B : \mathbb{R} \times \mathcal{X} \to \mathcal{U}$
Trajectories:	
p_k^{nom}	k-th nominal trajectory
p_k^{can}	k-th candidate trajectory
p_k^{com}	k-th committed trajectory

When π is piecewise continuous in t and Lipschitz wrt x, there exists an interval over which the solutions of (7) exist and are unique [25, Thm 3.1]. We assume this interval is $[t_0, \infty)$.

B. Perturbed System Description

Now consider a perturbed system without perfect state information. The perturbed system dynamics are

$$\dot{x} = f(x, u) + d(t), \tag{8a}$$
$$y = c(x) + v(t), \tag{8b}$$

where $y \in \mathbb{R}^p$ is the sensory output, and $c : \mathcal{X} \to \mathbb{R}^p$ is locally Lipschitz continuous. The additive disturbances $d : [t_0, \infty) \to \mathbb{R}^n$ and $v : [t_0, \infty) \to \mathbb{R}^p$ are bounded, $\sup_{t \geq t_0} \|d(t)\| = \bar{d} < \infty$, $\sup_{t \geq t_0} \|v(t)\| = \bar{v} < \infty$.

An observer-controller uses a state estimate $\hat{x} \in \mathcal{X}$ to determine the control input. The observer-controller is defined to be of the form

$$\dot{\hat{x}} = q(\hat{x}, y, u) \tag{9a}$$
$$u = \pi(t, \hat{x}) \tag{9b}$$

where $q : \mathcal{X} \times \mathbb{R}^p \times \mathcal{U} \to \mathbb{R}^n$ is locally Lipschitz in all arguments. The estimate-feedback controller $\pi : \mathbb{R}_{\geq 0} \times \mathcal{X} \to \mathcal{U}$ is assumed piecewise-continuous in t and Lipschitz in $\hat{x}$.

In this case, the closed-loop system dynamics are:

$$\dot{\hat{x}} = q(\hat{x}, y, \pi(t, \hat{x})), \qquad \hat{x}(t_0) = \hat{x}_0, \tag{10a}$$
$$\dot{x} = f(x, \pi(t, \hat{x})) + d(t), \qquad x(t_0) = x_0 \tag{10b}$$
$$y = c(x) + v(t) \tag{10c}$$

We assume that for each initial condition $(x_0, \hat{x}_0)$ and disturbance signals d, v, the solution exists and is unique for all time $t \in [t_0, \infty)$.

C. Set Invariance

Our method is based on concepts in set invariance.

Definition 1 (Controlled-Invariant Set)**.** *For the nominal system* (6)*, a controller* $\pi : [t_0, \infty) \times \mathcal{X} \to \mathcal{U}$ renders a set $\mathcal{C}(t) \subset \mathcal{X}$ controlled-invariant *on* t_0 *if, for the closed-loop system* (7) *and any* $\tau \geq t_0$*,*

$$x(\tau) \in \mathcal{C}(\tau) \implies x(t) \in \mathcal{C}(t), \ \forall t \geq \tau. \tag{11}$$

The concept of controlled invariance can be extended to the case with disturbances and an observer-controller [26].

Definition 2 (Robustly Controlled-Invariant Set)**.** *For the perturbed system* (8)*, an observer-controller* (9) renders a set $\mathcal{C}(t) \subset \mathcal{X}$ robustly controlled-invariant *on* t_0 *if, for the closed-loop system* (10) *and any bounded disturbance* d, v *with* $\sup_{t \geq t_0} \|d(t)\| \leq \bar{d}$*,* $\sup_{t \geq t_0} \|v(t)\| \leq \bar{v}$*, for any* $\tau \geq t_0$*,*

$$x(\tau) \in \mathcal{C}(\tau), \|\hat{x}(\tau) - x(\tau)\| \leq \delta \implies x(t) \in \mathcal{C}(t), \ \forall t \geq \tau. \tag{12}$$

for some $\delta > 0$*.*

Usually, the objective is to the find the largest controlled-invariant set $\mathcal{C}(t)$ for a given safe set $\mathcal{S}(t)$, referred to as the viability kernel [27]–[29]. However, these methods are difficult to apply when the safe set $\mathcal{S}(t)$ is unknown apriori, and instead is estimated online. The objective and approach of this paper is different, as described below.

D. Assumptions

Here, we formally state the assumptions that will be used to prove that `gatekeeper` renders a system safe. We assume the following modules are available, and explain the technical assumptions of each in the following paragraphs.

1) a model of the environment capable of bounding the evolution of the unsafe set,
2) a perception system that can sense (and forecast) the safe set,
3) a nominal planner that generates desired trajectories to satisfy mission requirements (for example reaching a goal state, or exploring a region), potentially using simplified dynamic models,
4) an input-to-state stable tracking observer-controller that can robustly track a specified trajectory,
5) a backup control policy that can stabilize the system to a control invariant set.

More specifically:

1) Environment Model: The time-varying safe set is denoted $\mathcal{S}(t) \subset \mathcal{X}$. While the full safe set may not be known at any given time, we assume that there are reasonable bounds on the evolution of the safe set. For example, in the firefighting scenario, an upper-bound on the fire's spread rate is known. Similarly, in an environment with dynamic obstacles, we assume that a reasonable upper-bound on the velocity or acceleration of the dynamic obstacles is known. As such, although we address safety in unknown environment, we still require some assumptions on the behavior of the environment to guarantee safety. The specific requirements of this model are described next.

2) Perception System: Let $\mathcal{S}(t) \subset \mathcal{X}$ be the time-varying set of safe states, which in general is unknown. We assume that the perception system provides *estimates* of the safe set that are updated as new information is acquired by the sensors. The information is available at discrete times t_k, $k \in \mathbb{N}$. Let $\mathcal{B}_k(t)$ denote the perceived safe set for time $t \geq t_k$ constructed using sensory information upto time t_k. We assume the following:

Assumption 1. The estimated safe set $\mathcal{B}_k(t)$ satisfies

$$\mathcal{B}_k(t) \subset \mathcal{S}(t) \qquad \forall k \in \mathbb{N}, t \geq t_k, \tag{13a}$$
$$\mathcal{B}_k(t) \subset \mathcal{B}_{k+1}(t) \qquad \forall k \in \mathbb{N}, t \geq t_{k+1}. \tag{13b}$$

This reads as follows. In (13a), we assume that any state perceived to be safe is indeed safe. In (13b), we assume that the perception system is conservative, i.e., new information acquired at t_{k+1} can only expand the perceived safe set at any time $t \geq t_{k+1}$.

This assumption (while stated more generally) is common in the literature on path planning in dynamic/unknown environments [20], [30]. Depending on the application, various methods can be used to computationally represent such sets, including SDFs [31] or SFCs [9].

Note, Assumption 1 *does not* require that if a state x is classified as safe at some time t_k, that x is safe for all time. Mathematically, we *do not* assume $x \in \mathcal{B}_k(t) \implies x \in \mathcal{B}_k(\tau) \ \forall \tau \geq t$. In the appendix, diagrams and a worked example with the firefighting mission is provided to help clarify Assumption 1 and the definitions of $\mathcal{S}(t), \mathcal{B}_k(t)$.

3) Nominal Planner: We assume that a nominal planner enforces the mission requirements by specifying the desired state of the robot for a short horizon T_H into the future.

Definition 3 (Trajectory)**.** *A* trajectory p *with horizon* T_H *is a piecewise continuous function* $p : \mathcal{T} \to \mathcal{X}$ *defined on* $\mathcal{T} = [t_k, t_k + T_H] \subset \mathbb{R}$*. A trajectory* p *is* dynamically feasible *wrt* (6) *if there exists a piecewise continuous control policy* $u : \mathcal{T} \to \mathcal{U}$ *s. t.*

$$p(t) = p(t_k) + \int_{t_k}^{t} f(p(\tau), u(\tau)) d\tau, \quad \forall t \in \mathcal{T}. \tag{14}$$

Denote the nominal trajectory available at the k-th iteration by the function $p_k^{nom} : [t_k, t_k + T_H] \to \mathcal{X}$. We do not require p_k^{nom} to be dynamically feasible wrt (6) or (8).

Note, although path planners (e.g. A*, RRT*) construct geometric paths, we assume the output of the path planner is a trajectory, i.e., is parameterized by time. Methods for time allocation of geometric paths is a well studied problem, see for e.g. [4], [5], [9].

4) Tracking Observer-Controller: We assume an estimate-feedback controller $\pi_T : \mathcal{X} \times \mathcal{X} \to \mathcal{U}$ that computes a control input $u = \pi_T(\hat{x}, p(t))$ to track a given trajectory p; we refer to this policy as the tracking observer-controller [32]–[34]. We assume that the tracking controller is input-to-state stable [26]:

Definition 4 (Input-to-State Stable Observer-Controller)**.** *Let $\mathcal{T} = [t_k, t_l] \subset \mathbb{R}_{\geq 0}$. A tracking observer-controller of the form*

$$u(t) = \pi_T(\hat{x}, p(t)) \tag{15a}$$
$$\dot{\hat{x}} = q(\hat{x}, y, u) \tag{15b}$$

is input-to-state stable *for the system (6), if, for any bounded disturbances $d : \mathcal{T} \to \mathbb{R}^n$ and $v : \mathcal{T} \to \mathbb{R}$, and any dynamically feasible trajectory $p : \mathcal{T} \to \mathcal{X}$, the following holds true:*

$$\begin{aligned} &\|x(t_k) - \hat{x}(t_k)\| \leq \delta, \text{ and } \quad p(t_k) = \hat{x}(t_k) \implies \\ &\quad \|x(t) - \hat{x}(t)\| \leq \beta(\delta, t - t_k) + \gamma(\bar{w}), \text{ and} \\ &\quad \|\hat{x}(t) - p(t)\| \leq \beta(\delta, t - t_k) + \gamma(\bar{w}), \text{ and} \\ &\quad \|x(t) - p(t)\| \leq \beta(\delta, t - t_k) + \gamma(\bar{w}), \ \forall t \in \mathcal{T}, \end{aligned} \tag{16}$$

where $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is class $\mathcal{KL}$, $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is class $\mathcal{K}$, and $\bar{w} = \max(\sup_{t\in\mathcal{T}} \|d(t)\|, \sup_{t\in\mathcal{T}} \|v(t)\|)$.

Note, we assumed the same β, γ for each of the three norms in (16) to simplify the notation. The presented methods can be re-derived using a different β, γ for each.

5) Backup Controller: In the case when a safe set $\mathcal{S}$ can not be rendered controlled invariant for given system dynamics, the objective reduces to finding a set $\mathcal{C} \subset \mathcal{S}$, and a controller $\pi : \mathcal{C} \to \mathcal{U}$ that renders $\mathcal{C}$ controlled invariant. For example, by linearizing (6) around a stabilizable equilibrium x_e, an LQR controller renders a (sufficiently small) set of states around x_e forward invariant [25, Thm. 4.13, 4.18]. This observation leads to the notion of backup safety [6], [35].

Definition 5 (Backup Controller)**.** *A controller $\pi_B^k : \mathcal{T} \times \mathcal{X} \to \mathcal{U}$ is a* backup controller *to a set $\mathcal{C}_k(t) \subset \mathcal{X}$ defined for $t \in \mathcal{T} = [t_k, \infty)$ if, for the closed-loop system*

$$\dot{x} = f(x, \pi_B^k(t, x)), \tag{17}$$

(A) there exists a neighborhood $\mathcal{N}_k(t) \subset \mathcal{X}$ of $\mathcal{C}_k(t)$, s.t. $\mathcal{C}_k(t)$ is reachable in fixed time T_B:

$$x(\tau) \in \mathcal{N}_k(\tau) \implies x(\tau + T_B) \in \mathcal{C}(\tau + T_B), \tag{18}$$

and (B) π_B^k renders $\mathcal{C}_k(t)$ controlled-invariant:

$$x(\tau + T_B) \in \mathcal{C}(\tau + T_B) \implies x(t) \in \mathcal{C}(t) \ \forall t \geq \tau + T_B. \tag{19}$$

We make the following assumption:

Assumption 2. At the k-th iteration, a set $\mathcal{C}_k(t)$ and a backup controller $\pi_B^k : [t_k, \infty) \times \mathcal{X} \to \mathcal{U}$ to $\mathcal{C}_k(t)$ can be found where

$$\mathcal{C}_k(t) \subset \mathcal{S}(t), \ \forall t \geq t_k. \tag{20}$$

Remark 1. Note that while we assume $\mathcal{C}_k(t) \subset \mathcal{S}(t)$, we *do not* assume the trajectory to reach $\mathcal{C}_k(t)$ is safe, nor that the set $\mathcal{C}_k(t)$ is reachable from the current state $x(t_k)$ within a finite horizon. This is in contrast to backward reachability based methods [13], [14], [36], [37]. Instead, we will ensure both of these conditions are satisfied through our algorithm.

E. Problem Statement

In summary, the problem statement is

Problem 1. Consider system (8) satisfying the assumptions listed in Section IV-D, i.e., a system with a perception system satisfying Assumption 1, a nominal planner that generates desired trajectories, an input-to-state stable tracking controller satisfying Definition 4, and a backup controller satisfying Assumption 2. Design an algorithm to track desired trajectories while ensuring safety, i.e., $x(t) \in \mathcal{S}(t)$ for all $t \geq t_0$.

V. PROPOSED SOLUTION

`gatekeeper` is a module that lies *between* the planning and control modules. It considers the nominal trajectories by the planner, modifies them as needed to what we call committed trajectories, and inputs these committed trajectories to the trajectory-tracking controller. In this section, we will demonstrate how to construct these committed trajectories. To aid the reader, the analysis is first presented for the nominal case, and later extended to the perturbed case. The various trajectories and times are depicted in Fig. 2. The algorithm is described in Algorithm 1 and depicted in Fig. 1.

A. Nominal Case

Suppose at the k-th iteration, $k \in \mathbb{N} \setminus \{0\}$, the previously committed trajectory is p_{k-1}^{com}. `gatekeeper` constructs a candidate trajectory p_k^{can,T_S} by forward propagating a controller that tracks p_k^{nom} over an interval $[t_k, t_k + T_S)$, and executes the backup controller for $t \geq t_k + T_S$. $T_S \in \mathbb{R}_{\geq 0}$ is a switching duration that `gatekeeper` will maximize, as described later. Formally,

Definition 6 (Candidate Trajectory)**.** *Suppose at $t = t_k$,*

- *the state is $x(t_k) = x_k$,*
- *the nominal trajectory is $p_k^{nom} : [t_k, t_k + T_H] \to \mathcal{X}$,*
- *π_B^k is a backup controller to the set $\mathcal{C}_k(t)$*

Given a $T_S \in [0, T_H]$, the candidate trajectory *$p_k^{can,T_S} : [t_k, \infty) \to \mathcal{X}$ is the solution to the initial value problem*

$$\dot{p} = f(p, u(t)), \tag{21a}$$
$$p(t_k) = x_k, \tag{21b}$$
$$u(t) = \begin{cases} \pi_T(p(t), p_k^{nom}(t)) & t \in [t_k, t_k + T_S) \\ \pi_B^k(t, p(t)) & t \geq t_k + T_S. \end{cases} \tag{21c}$$

Note, by construction the candidate trajectory is dynamically feasible wrt (6).

We say a candidate trajectory is *valid* if the following hold:

Definition 7 (Valid)**.** *A candidate trajectory $p_k^{can,T_s} : [t_k, \infty) \to \mathcal{X}$ defined by (21) is* valid *if the trajectory is safe wrt the estimated safe set over a finite interval:*

$$p_k^{can,T_S}(t) \in \mathcal{B}_k(t), \ \forall t \in [t_k, t_{k,B}], \tag{22}$$

and the trajectory reaches $\mathcal{C}_k(t)$ at the end of the horizon:

$$p_k^{can,T_S}(t_{k,B}) \in \mathcal{C}_k(t_{k,B}), \tag{23}$$

where $t_{k,B} = t_k + T_S + T_B$.

Notice checking whether a candidate is valid only requires the solution p_k^{can,T_S} over the finite interval $[t_k, t_k + T_S + T_B]$.

This means that the candidate can be constructed by numerical forward integration over a finite horizon.

Def. 8 defines how the k-th committed trajectory is constructed using the nominal trajectory p_k^{nom}, the backup controller π_k^B, and the previous committed trajectory p_{k-1}^{com}.

Definition 8 (Committed Trajectory)**.** *At the k-th iteration, define*

$$\mathcal{I}_k = \left\{ T_S \in [0, T_H] : p_k^{can,T_S} \text{ is valid} \right\} \subset \mathbb{R}, \tag{24}$$

where $p_k^{can,T_S} : [t_k, \infty) \to \mathcal{X}$ is as defined in (21), and Def. 7 is used to check validity. The committed trajectory is $p_k^{com} : [t_k, \infty) \to \mathcal{X}$, defined as follows:

If $\mathcal{I}_k \neq \emptyset$, let $T_S^* = \max \mathcal{I}_k$. The committed trajectory *is*

$$p_k^{com}(t) = p_k^{can,T_S^*}(t), \quad t \in [t_k, \infty). \tag{25}$$

If $\mathcal{I}_k = \emptyset$, the committed trajectory *is*

$$p_k^{com}(t) = p_{k-1}^{com}(t), \quad t \in [t_k, \infty). \tag{26}$$

Finally, we are ready to prove that the proposed strategy guarantees safety. First, we show that each committed trajectory is safe.

Theorem 1. *Suppose $p_0^{can,T_S} : [t_0, \infty) \to \mathcal{X}$ is a candidate trajectory that is dynamically feasible wrt (6) and valid according to Def. 7 for some $T_S \geq 0$.*

If, for every $k \in \mathbb{N}$, $p_k^{com} : [t_k, \infty) \to \mathcal{X}$ is determined using Def. 8, then for all $k \in \mathbb{N}$,

$$p_k^{com}(t) \in \mathcal{S}(t), \quad \forall t \in [t_k, \infty). \tag{27}$$

Proof. The proof is by induction.

Base Case: $k = 0$. Since p_0^{can} is a valid trajectory, it is committed, i.e., $p_0^{com} = p_0^{can,T_S}$. Then,

$$\begin{aligned} p_0^{com}(t) &\in \begin{cases} \mathcal{B}_0(t) & \text{for } t \in [t_0, t_{0,B}) \\ \mathcal{C}_0(t) & \text{for } t = t_{0,B} \end{cases} \\ \implies p_0^{com}(t) &\in \begin{cases} \mathcal{S}(t) & \text{for } t \in [t_0, t_{0,B}) \\ \mathcal{S}(t) & \text{for } t \geq t_{0,B} \end{cases} \\ \iff p_0^{com}(t) &\in \mathcal{S}(t) \text{ for } t \geq t_0 \end{aligned}$$

where $t_{0,B} = t_0 + T_S + T_B$.

Induction Step: Suppose the claim is true for some $k \in \mathbb{N}$. We will show the claim is also true for $k+1$. There are two possible definitions for p_k^{com}:

Case 1: When $\mathcal{I}_{k+1} \neq \emptyset$, p_{k+1}^{can,T_S^*} is a valid candidate, i.e.,

$$\begin{aligned} p_{k+1}^{com}(t) = p_{k+1}^{can,T_S^*}(t) \ \forall t \geq t_0 \\ \in \begin{cases} \mathcal{B}_{k+1}(t) & \text{for } t \in [t_{k+1}, t_{k+1,SB}) \\ \mathcal{C}_{k+1}(t) & \text{for } t \geq t_{k+1,SB} \end{cases} \\ \in \mathcal{S}(t) \text{ for } t \geq t_{k+1} \end{aligned}$$

Case 2: If $\mathcal{I}_{k+1} = \emptyset$, the committed trajectory is unchanged,

$$p_{k+1}^{com}(t) = p_k^{com}(t) \in \mathcal{S}(t), \ \forall t \geq t_{k+1}.$$

□

The following theorem demonstrates that `gatekeeper` ensures the system remains safe.

Theorem 2. *Under the assumptions of Theorem 1, if $x(t_0) = p_0^{com}(t_0)$, and for each $k \in \mathbb{N}$ the control input to the nominal system (6) is*

$$u(t) = \pi_T^k(x(t), p_k^{com}(t)), \forall t \in [t_k, t_{k+1}), \tag{28}$$

then the closed-loop dynamics (7) will satisfy

$$x(t) \in \mathcal{S}(t), \forall t \geq t_0. \tag{29}$$

Proof. We prove this by showing that $\forall k \in \mathbb{N}$, $x(t) = p_k^{com}(t)$ for $t \in [t_k, t_{k+1})$. Again, we use induction.

Base Case: Since we are considering the nominal system (6), if $x(t_0) = p_0^{com}(t_0)$, and the tracking controller is ISS (16),

$$\begin{aligned} \|x(t) - p_0^{com}(t)\| \leq \beta(0, t - t_0) + \gamma(0) = 0 \\ \therefore x(t) = p_0^{com}(t) \ \forall t \in [t_0, t_1) \end{aligned}$$

Induction Step: Suppose for some $k \in \mathbb{N}$, $x(t) = p_k^{com}(t)$ for $t \in [t_k, t_{k+1})$. There are two cases for p_{k+1}^{com}:

Case 1: a new candidate is committed, $\therefore p_{k+1}^{can,T_S}(t_{k+1}) = x(t_{k+1})$. Since the tracking controller is input-to-state stable, this implies $x(t) = p_{k+1}^{com}(t)$ for $t \in [t_{k+1}, t_{k+2})$.

Case 2: A new candidate is not committed, $\therefore p_{k+1}^{com}(t) = p_k^{com}(t)$ for $t \in [t_{k+1}, t_{k+2})$. Since $x(t_{k+1}) = p_k^{com}(t_{k+1})$, the tracking controller ensures $x(t) = p_{k+1}^{com}(t)$ for $t \in [t_{k+1}, t_{k+2})$.

Therefore, $x(t) = p_k^{com}(t) \in \mathcal{S}(t) \ \forall t \in [t_k, t_{k+1})$, for each $k \in \mathbb{N}$. Thus, $x(t) \in \mathcal{S}(t)$ for all $t \geq t_0$. □

Remark 2. In [6], [35], numerical forward propagation of the trajectory with a backup controller is also used to construct a safety filter. However, the resulting control input mixes the nominal control input with the backup control input at all times. In contrast, in `gatekeeper` we use a switching time to switch between implementing the nominal control input and the backup control input. This is desirable since it leads to less conservative controllers, as highlighted in section VI-A.

B. Perturbed Case

We now address the case with non-zero disturbances and state-estimation error.[1] The algorithm is identical to that presented above, except that the validation step will be redefined.

First, we highlight the problem that disturbances introduce. Consider the specific scenario visualized in Fig. 3. To account for the disturbances, we validate safety of a tube around the candidate trajectory: using the ISS bound (16), a tube of decreasing radius around the committed trajectory will always contain the true state of the system. Therefore, if instead of (22), we checked that the corresponding tube containing the candidate trajectory lies within the safe set (green tube in Fig. 3a), then indeed, the system will remain safe.

However, when a new candidate is proposed at the next iteration, the new tube (red tube) intersects with the unsafe

[1]Compared to the conference version [7], here we consider the additional uncertainty due to state estimation errors, and simplify the validation check.

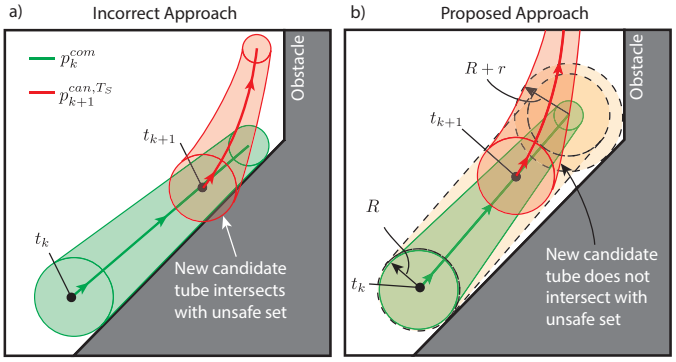

Fig. 3. Diagram depicting the challenge due to disturbances. (a) Green line shows the committed trajectory at iteration k, and the shaded region is the tube that contains the system trajectory. If the validation step only checks that the green tube lies within the safe set, a new candidate trajectory (red) cannot be committed, since the candidate tube (red shaded region) intersects with the unsafe set. (b) shows the proposed approach, where safety is checked wrt the yellow set, i.e., a tube of radius R along the trajectory and a ball of radius $R+r$ at the end. This allows for sufficient margin to commit a new trajectory at the next iteration.

set. Thus, the new candidate cannot be committed, and an undesired deadlock is reached: $x(t) \in \mathcal{C}_k(t)$ for all $t \geq t_{k,B}$.

To avoid this behavior, we propose a different validity check. First, we check that a tube of radius R is safe over the finite horizon, and second, we ensure $\mathcal{C}_k$ is a (larger) distance $R+r$ away from the unsafe set, where r, R are defined below. In Fig. 3a, this is depicted by the yellow sets. Note, the additional $+r$ term is used to avoid the described deadlock behaviour, and is not needed to guarantee safety.

Recall Def. 4 defines the controller's tracking error bounds. The validity check in Def. 7 is replaced by the following:

Definition 9 (Robustly Valid)**.** *Consider the dynamical system (8), where the disturbances are bounded by* $\sup_{t\geq t_k} \|d(t)\| \leq \bar{d}$, *and* $\sup_{t\geq t_k} \|v(t)\| \leq \bar{v}$. *Let* $\bar{w} = \max(\bar{d}, \bar{v})$.

Suppose $\|x(t_k) - \hat{x}(t_k)\| \leq r$ *for some* $k \in \mathbb{N}$. *Let* $R = \beta(r, 0) + \gamma(\bar{w})$.

A candidate trajectory $p_k^{can,T_s} : [t_k, \infty) \to \mathcal{X}$ *defined by (21) is* robustly valid *if*

- *the candidate trajectory coincides with the state estimate at the initial time:*

$$\hat{x}(t_k) = p_k^{can,T_s}(t_k), \tag{30}$$

- *the candidate trajectory is robustly safe over a finite interval:*

$$p_k^{can,T_S}(t) \in \mathcal{B}_k(t) \ominus \mathbb{B}(R) \ \forall t \in [t_k, t_{k,B}], \tag{31}$$

- *at the end of the interval, it reaches* $\mathcal{C}_k(t)$*:*

$$p_k^{can,T_S}(t_{k,SB}) \in \mathcal{C}_k(t_{k,B}), \tag{32}$$

- *and the set* $\mathcal{C}_k(t)$ *is* $(R+r)$ *away from the unsafe set:*

$$\mathcal{C}_k(t) \subset \mathcal{S}(t) \ominus \mathbb{B}(R+r) \ \forall t \geq t_k. \tag{33}$$

If a candidate trajectory is robustly valid, it can be committed. The following theorem proves that `gatekeeper` can render the perturbed system (8) safe.

Theorem 3. *Suppose* $p_0^{com} : [t_0, \infty) \to \mathcal{X}$ *is a committed trajectory on* $[t_0, \infty)$ *that is* robustly *valid by Def. 9 for some* $r > 0, T_S \geq 0$. *Suppose* $\|x(t_0) - \hat{x}(t_0)\| \leq r$, *and* $p_0^{com}(t_0) = \hat{x}(t_0)$.

If, for every $k \in \mathbb{N}\backslash\{0\}$, $p_k^{com} : [t_k, \infty) \to \mathcal{X}$ *is determined using Def. 8 (except that validity is checked using Def. 9), and the control input to the perturbed system (8) is*

$$u(t) = \pi_T^k(\hat{x}(t), p_k^{com}(t)) \quad \forall t \in [t_k, t_{k+1}]$$

then the closed-loop system (10) will satisfy

$$x(t) \in \mathcal{S}(t), \quad \forall t \geq t_0. \tag{34}$$

Proof. Based on the same arguments as in Thm. 1, we have that for any $k \in \mathbb{N}$,

$$p_k^{com}(t) \in \mathcal{S}(t) \quad \forall t \in [t_k, \infty). \tag{35}$$

We aim to prove the analog of Thm 2, i.e., that for any $k \in \mathbb{N}$, tracking the committed trajectory $p_k^{com}(t_k)$ for $t \geq t_k$ is safe. This is proved below.

Since p_k^{com} is robustly valid, $p_k^{com}(t_k) = \hat{x}(t_k)$. Therefore,

$$\|x(t_k) - p_k^{com}(t_k)\| = \|x(t_k) - \hat{x}(t_k)\| \leq r.$$

Using (16), this implies that for all $t \geq t_k$,

$$\begin{aligned} \|x(t) - p_k^{com}(t)\| &\leq \beta(r, t - t_k) + \eta(\bar{w}) \\ &\leq \beta(r, 0) + \eta(\bar{w}) \\ &= R \\ \therefore \|x(t) - p_k^{com}(t)\| &\leq R \end{aligned}$$

and so by (31),

$$\begin{aligned} & p_k^{com}(t) \in \mathcal{B}_k(t) \ominus \mathbb{B}(R) && \forall t \in [t_k, t_{k,SB}] \\ \implies & \{p_k^{com}(t)\} \oplus \mathbb{B}(R) \subset \mathcal{B}_k(t) && \forall t \in [t_k, t_{k,SB}] \\ \implies & x(t) \in \mathcal{B}_k(t) && \forall t \in [t_k, t_{k,SB}]. \end{aligned}$$

Furthermore, since for all $t \geq t_k + T_S$ the committed trajectory is generated the backup controller, and $p_k^{com}(t_{k,B}) \in \mathcal{C}_k(t_{k,B})$, we have $p_k^{com}(t) \in \mathcal{C}_k(t), \forall t \geq t_{k,B}$. Therefore,

$$x(t) \in \mathcal{C}_k(t) \oplus \mathbb{B}(R) \quad \forall t \geq t_{k,B}.$$

Putting these together,

$$\begin{aligned} x(t) &\in \begin{cases} \mathcal{B}_k(t) & \text{for } t \in [t_k, t_{k,B}] \\ \mathcal{C}_k(t) \oplus \mathbb{B}(R) & \text{for } t \geq t_{k,B} \end{cases} \\ \implies x(t) &\in \begin{cases} \mathcal{S}_k(t) & \text{for } t \in [t_k, t_{k,B}) \\ \mathcal{S}_k(t) & \text{for } t \geq t_{k,B} \end{cases} \\ \iff x(t) &\in \mathcal{S}(t) \quad \forall t \geq t_k. \end{aligned}$$

□

This proves that for any $k \in \mathbb{N}$, if p_k^{com} is the committed trajectory, the system will remain safe while it is tracking p_k^{com}. When a new candidate trajectory that is robustly valid (by Def. 9) is found, the committed trajectory can be updated, and the system will continue to remain safe.

Remark 3. The theorem provides certain parameters of the nominal planner. For instance, requiring trajectories to lie in

$\mathcal{B}(t_k) \ominus \mathbb{B}(R)$ corresponds to the common practice of inflating the unsafe sets by a radius R. The theorem shows that any $R \geq \beta(r, 0) + \gamma(\bar{w})$ is sufficient.

Remark 4. In (33), we checked that $\mathcal{C}_k(t)$ is at least $(R+r)$ away from $\mathcal{S}(t)$ at all $t \geq t_k$, even though the proof of safety only requires a margin R. The reason we check for $(R+r)$ is to prevent the deadlock scenario discussed before: under the stated assumptions, for $t \geq t_k$, $\|x(t) - \hat{x}(t)\| \leq \beta(t - t_k) + \gamma(\bar{w})$. Therefore, if $r \geq \gamma(\bar{w})$ there exists some time $\tau = t_k + T$ where $r = \beta(T, r) + \gamma(\bar{w})$ since class $\mathcal{KL}$ functions are strictly decreasing wrt t. Thus, for $t \geq \tau$, $p_k^{com}(t) \in \mathcal{C}_k(t)$. Thus,

$$x(t), \hat{x}(t) \in \mathcal{C}_k(\tau) \oplus \mathbb{B}(r)$$

and $\|x(t) - \hat{x}(t)\| \leq r$. Thus, when validating the new candidate trajectory $p_{k'}^{can,T_s}$ they will start at least R away from the boundary, i.e., there is sufficient margin for new trajectories to be committed.

Remark 5. The construction of committed trajectories is summarized in pseudo-code in Alg. 1. Determining $\max \mathcal{I}$ is not computationally expensive, since it is an optimization over a scalar variable in a bounded interval. In our implementations, we have used a simple grid search with N points. Therefore, upto N initial value problems need to be solved. Using modern differential equation solvers, e.g. [38], this can be done very efficiently. In our simulations, with $N = 10$, the median computation time was only 3.4 ms.

Algorithm 1: `gatekeeper`

Parameters: $N > 0 \in \mathbb{N}$
`// Do a grid search backwards over the interval` $[0, T_H]$`:`
for *i in range(0, N):* **do**
  Using $\mathcal{B}_k(t)$, identify $\mathcal{C}_k(t)$ satisfying assum. 2.
  $T_S = (1 - i/N)T_H$
  Solve the initial value problem (21) to determine $p_k^{can,T_S}(t)$ over the interval $[t_k, t_k + T_S + T_B]$
  if p_k^{can,T_S} *is robustly valid by Def. 9* **then**
    $p_k^{com} = p_k^{can,T_S}$
    return
`// no candidate is valid,` $\mathcal{I} = \emptyset$
$p_k^{com} = p_{k-1}^{com}$
return

VI. SIMULATIONS AND EXPERIMENTS

Code and videos are available here: [39].

A. Firewatch Mission

We simulate an autonomous helicopter performing the fire-watch mission, around a fire with an initial perimeter of 16 km. The helicopter begins 0.45 km from the fire front, and is tasked to fly along the perimeter, without entering the fire, while maintaining a target airspeed of 15 m/s. The helicopter is modeled as:

$$\dot{x}_1 = x_3 \cos x_4 \qquad \dot{x}_2 = x_3 \sin x_4$$
$$\dot{x}_3 = u_1 \qquad \dot{x}_4 = (g/x_3) \tan u_2$$

where x_1, x_2 are the cartesian position coordinates of the helicopter wrt an inertial frame, x_3 is the speed of the vehicle along its heading, x_4 is the heading, and g is the acceleration due to gravity. The control inputs are u_1, the acceleration along the heading, and u_2, the roll angle. The inputs are bounded, with $|u_1| < 0.5g$ and $|u_2| < \pi/4$ rad. This system models a UAV that can control its forward airspeed and makes coordinated turns. Notice the model has a singularity at $x_3 = 0$, and the system is *not* control affine.

The fire is modeled using level-set methods [40]. In particular, the fire is described using the implicit function $\phi : \mathbb{R} \times \mathbb{R}^2 \to \mathbb{R}$, where $\phi(t, p)$ is the signed distance to the firefront from location p at time t. Hence, the safe set is

$$\mathcal{S}(t) = \{x : \phi(t, [x_1, x_2]^T) \geq 0\}$$

where $[x_1, x_2]$ are the Cartesian coordinates of the UAV.

The evolution of the fire is based on the Rothermel 1972 model [23]. Given the Rate of Spread (RoS) function $\sigma : \mathbb{R}^2 \to \mathbb{R}$, the safe set evolves according to

$$\frac{\partial \phi}{\partial t}(t, p) + \sigma(p) \|\nabla \phi(t, p)\| = 0 \; \forall p \in \mathbb{R}^2 \tag{37}$$

The RoS depends on various environmental factors including terrain topology, vegetation type, and wind speeds [23], [24] but can be bounded [41]. The simulated environment was assigned an RoS function that the controllers did not have access to. The only information the controllers were allowed to use was the thermal image (to detect the fire within a ± 1 km range of the UAV) and assumption that the maximum rate of spread is 8 km/h.

We compare our approach against the nominal planner and two state of the art methods for similar problems, Fig. VI-A. In particular, we compare (A) a nominal planner (black), (B) FASTER [5] (purple), (C) Backup Filters [6] (blue) and (D) `gatekeeper` (green). Since these methods were not originally developed for dynamic environments with limited sensing, both methods (B, C) were modified to be applicable to this scenario. See [39] for details.

The simulation environment and each of the methods were implemented in `julia`, to allow for direct comparison, using `Tsit5()` [38] with default tolerances. Each run simulates a flight time of 50 minutes. The tracking controller was implemented as zero-order hold, updated at 20 Hz. Measurements of the firefront were available at 0.1 Hz, triggering the planners to update, intentionally slow to highlight the challenges of slow perception/planning systems. The measurements are a bitmask image, defining the domain where $\phi \leq 0$, at a grid resolution of 10 meters. These simulations were performed on a 2019 Macbook Pro (Intel i9, 2.3 GHz, 16 GB).

In the nominal planner, a linear MPC problem is solved to generate trajectories that fly along the local tangent 0.1 km away from firefront at 15 m/s. The planner uses a simplified dynamic model for the helicopter, a discrete-time double

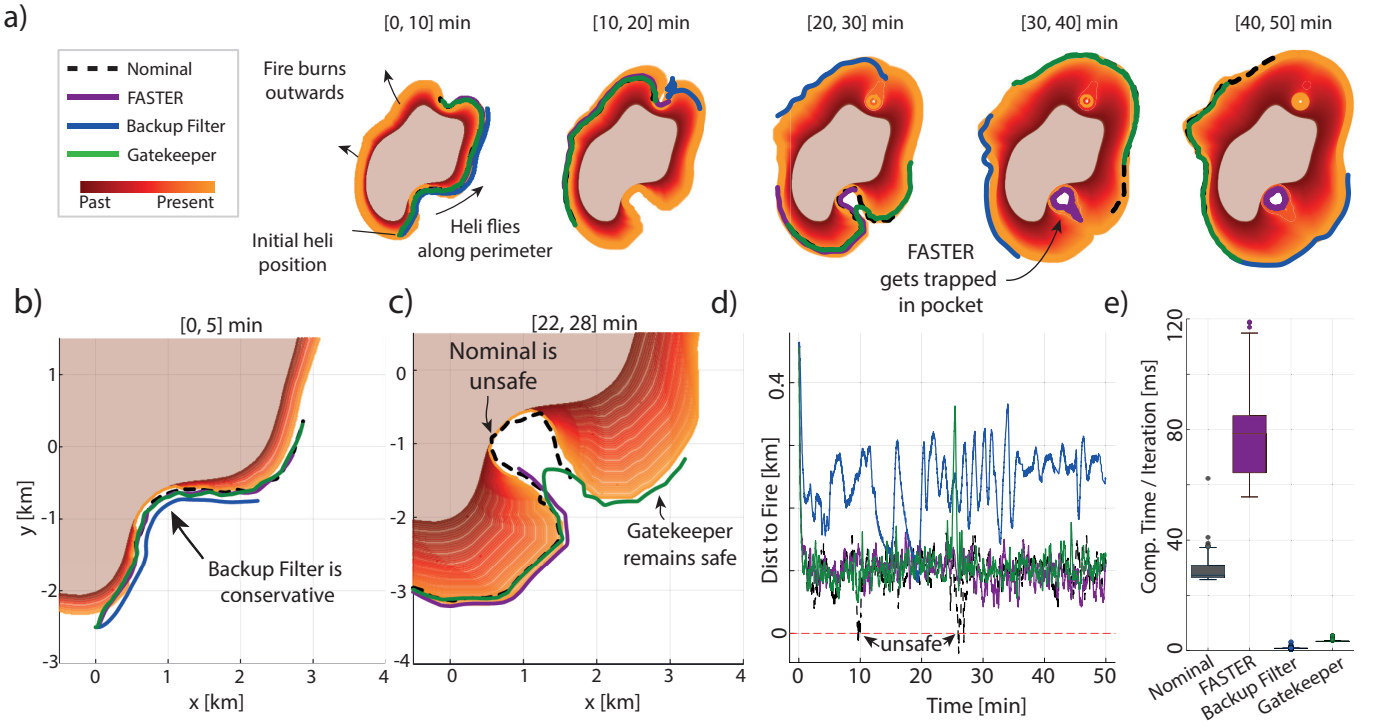

Fig. 4. Simulation results from Firewatch mission. (a) Snapshots of the fire and trajectories executed by each of three controller. The fire is spreading outwards, and the helicopters are following the perimeter. The black line traces the nominal controller, the blue line is based on the backup filter adapted from [6] and the green line shows the proposed controller. (b, c) show specific durations in greater detail. At $t = 0$, the `gatekeeper` controller behaves identically to the nominal controller, and makes small modifications when necessary to ensure safety. The backup filter is conservative, driving the helicopter away from the fire and slowing it down. (d) Plot of minimum distance to fire-front across time for each of the controllers. (e) The nominal controller becomes unsafe 3 times, while FASTER, the backup controller, and the `gatekeeper` controllers maintain safety. Animations are available at [39].

Table II. Comparison of `gatekeeper` (ours) with the nominal planner, FASTER [5], and backup filters [6]. The distance to the firefront, velocity of the helicopter, and computation time per iteration are reported for each method. IQR = interquartile range. *Since the backup filter is run at each control iteration instead of every planning iteration, it runs 20 times as often as `gatekeeper`, i.e., is 5 times as computationally expensive as `gatekeeper`.

	Distance to Fire [km]			Velocity [m/s]		Comp. time [ms]		
	Minimum	Mean	Std.	Mean	Std.	Median	IQR	
Target	≥ 0	0.100	-	15.0	-	-	-	
Nominal Planner	-0.032	0.098	0.032	15.14	0.73	27.32	4.37	Unsafe
FASTER [5]	0.040	0.101	0.030	12.60	2.08	78.50	20.64	Safe, but gets trapped in pocket
Backup Filters [6]	0.081	0.240	0.054	10.11	3.52	0.87*	0.05	Safe, but conservative and slow
Gatekeeper (proposed)	0.049	0.108	0.034	14.91	1.35	3.39	0.11	Safe

integrator. This problem is a convex quadratic program (QP), solved using `gurobi`. The median computation time is 27 ms, using $N = 40$ waypoints and a planning horizon of 120 seconds. The tracking controller is a nonlinear feedback controller that directly tracks trajectories of the double integrator, based on differential flatness [15], [34]. When the tracking controller directly tracks nominal trajectories, the system becomes unsafe, going as far as 32 m into the fire.

In FASTER, the same double integrator model is assumed, and a similar MPC problem is solved. We impose additional safety constraints, that the committed trajectory must lie within a safe flight corridor [9] based on the signed distance field to the fire, corrected based on the maximum fire spread rate. While this approach does keep the helicopter outside the fire, it gets surrounded by the fire (Fig. VI-Aa). This is ultimately due to the fact that FASTER only plans trajectories over a finite planning horizon, and is therefore unable to guarantee recursive feasibility in a dynamic environment. Due to the large number of additional constraints on the QP, FASTER is about 3 times slower than the nominal planner.

In the Backup Filters approach, the backup trajectory is numerically forward propagated on the nonlinear system over the same 120 second horizon, and can be computed efficiently, requiring less than 1 ms per iteration. While this approach keeps the system safe, it does so at the cost of performance: the mean distance to the fire is 0.24 km, more than twice the target value, and the average speed is 10 m/s, 33% less than the target. This behavior is because the desired flight direction is perpendicular to the backup flight direction, and therefore the executed trajectory is off-nominal.

In `gatekeeper`, the committed trajectories are constructed by maximizing the interval that the nominal trajectory

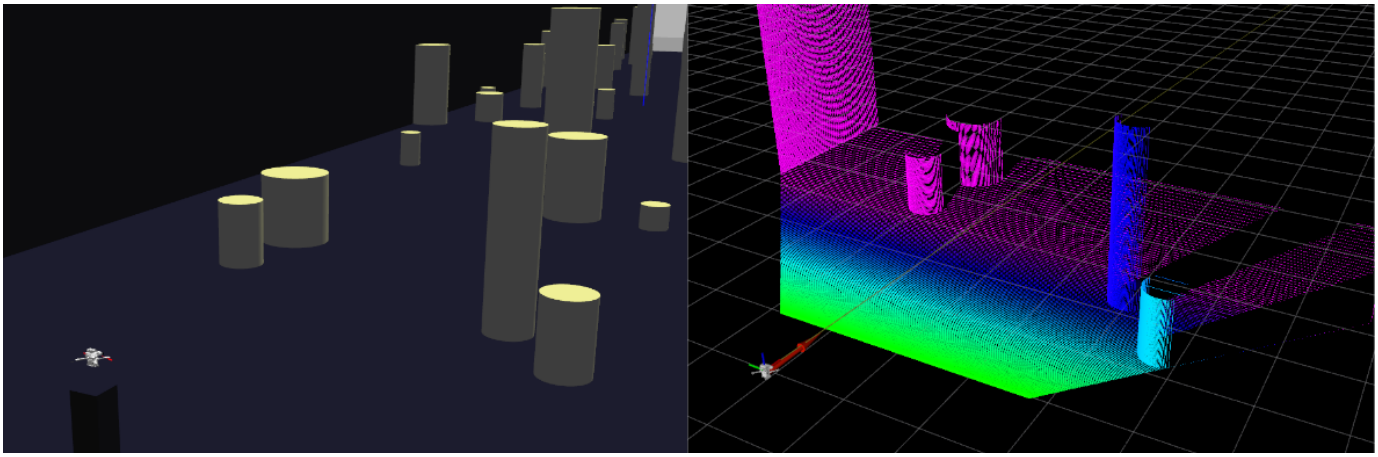

Fig. 5. (Left) Simulation environment comprising of a quadrotor navigating in a 50 m long corridor with randomly scattered cylindrical obstacles of various heights and radii. This picture depicts the "Easy 1" world. (Right) The point-cloud sensor data received by the quadrotor describing the environment. Using the point-cloud, a Signed Distance Field (SDF) representation of the environment is constructed. A Safe Flight Corridor (SFC), i.e., a convex polyhedron of obstacle-free space, centered on the quadrotor is extracted and used as the perceived safe set. The nominal planner treats unknown regions as free, while `gatekeeper` treats unknown regions as occupied.

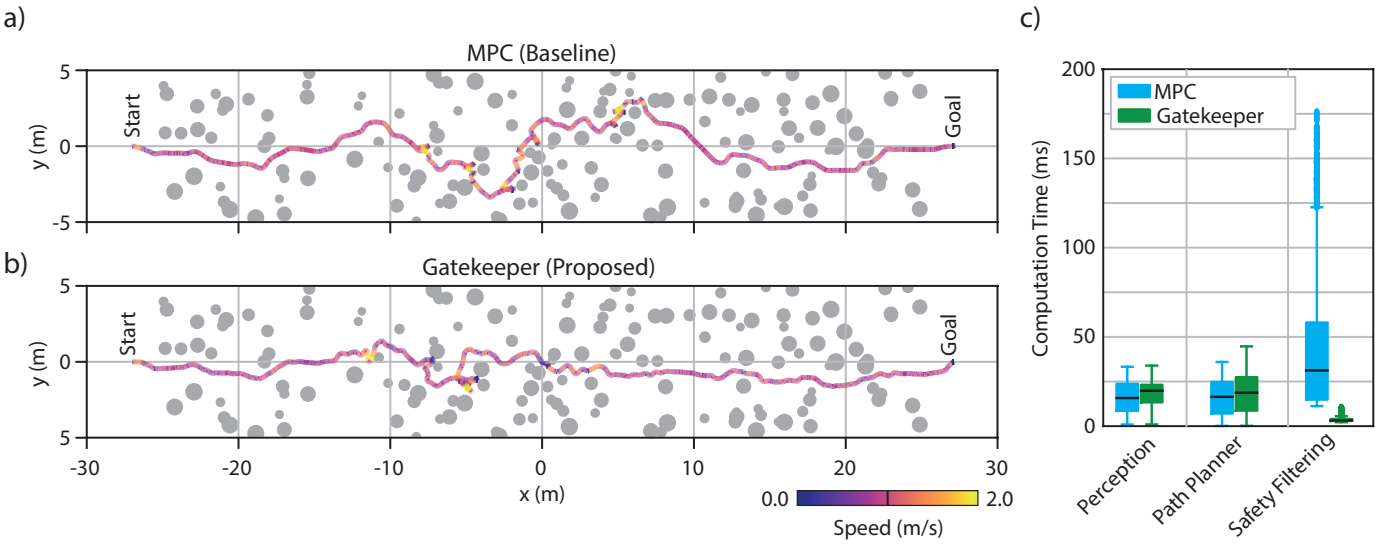

Fig. 6. Trajectories executed in the "Hard 1" world, using (a) MPC-based safety filter (baseline) and (b) the proposed `gatekeeper`-based safety filter. The gray circles indicate obstalces. Visually, the paths are similar, and are traversed with similar speeds. The color indicates that for most of the trajectories, the speed is at the target of 1 m/s, but near the obstacles (where there is greater replanning), the speeds vary more. (c) Box-and-whiskers plot showing the computation time for perception, planning, and safety filtering. The computation time for perception and path planning is similar with both safety filters, since both use the same perception and path planning implementation. However, `gatekeeper` is significantly faster than the MPC-based safety filter.

is tracked, before implementing the backup controller. This allows the system to follow the nominal, and deviate only when required to ensure safety: in Fig. VI-Ac, we see that `gatekeeper` chooses to not fly into the pocket, since it cannot ensure a safe path out of the pocket will exist in the future. `gatekeeper` is computationally lightweight, with a median run time of 3.4 ms, more than 20 times faster than FASTER. This is primarily because `gatekeeper` searches over a scalar variable in a bounded interval, instead of optimizing $\mathbb{R}^{4N+2N-2}$ variables as in the MPC problem.

B. Quadrotor Navigation (Simulations)

We demonstrate the efficacy of the `gatekeeper` algorithm for a quadrotor flying through a previously unobserved area, in both a high fidelity simulation, and hardware experiments. A desired goal location is specified by the human operator. The quadrotor must simultaneously sense the environment, build a local map of the obstacles, plan a path to the goal, filter the path using `gatekeeper`, and finally execute the committed trajectory. All of the processing happens onboard, in realtime, and by using `gatekeeper`, the quadrotor does not crash into any obstacles. Each step of the entire perception-planning-control stack is described next, followed by a comparison to state-of-the-art methods to accomplish this objective. All simulations were run using Gazebo and RotorS [42], on an AMD Ryzen 7 5800h CPU with 16 GB of memory, and an NVIDIA RTX 3050Ti GPU. All hardware experiments were performed on a 16GB Nvidia Xavier NX.

Table III. Summary of simulations in 15 different worlds with 3 difficulty levels, comparing the performance of an MPC-based safety filter against `gatekeeper`. `gatekeeper` is able to successfully reach the goal in more scenarios, and is an order of magnitude computationally faster.

	Goal Reached?		Median Comp. Time [ms]		Max Comp. Time [ms]		Average Speed [m/s]	
World	MPC	GK	MPC	GK	MPC	GK	MPC	GK
easy 1	True	True	34.71 ± 0.10	3.28 ± 0.04	168.89	10.78	0.91	0.81
easy 2	True	True	35.55 ± 0.11	3.42 ± 0.05	161.59	10.51	0.82	0.77
easy 3	True	True	33.18 ± 0.12	3.26 ± 0.05	151.12	12.48	0.83	0.67
easy 4	True	True	34.17 ± 0.25	3.33 ± 0.05	172.15	11.70	0.83	0.45
easy 5	True	True	35.40 ± 0.20	3.36 ± 0.05	180.94	11.94	0.90	0.35
medium 1	True	True	39.78 ± 0.20	3.27 ± 0.06	178.21	10.62	0.78	0.61
medium 2	False	True	47.62 ± 1.10	3.27 ± 0.05	217.97	12.19	0.57	0.76
medium 3	True	True	33.65 ± 0.08	3.18 ± 0.04	199.93	11.53	0.79	0.75
medium 4	False	False	44.93 ± 2.03	3.23 ± 0.06	199.74	9.25	0.44	0.43
medium 5	True	True	27.72 ± 0.10	3.23 ± 0.05	211.08	10.82	0.81	0.82
hard 1	True	True	31.22 ± 0.16	3.18 ± 0.07	201.54	9.65	0.68	0.82
hard 2	False	True	56.61 ± 0.61	3.41 ± 0.08	184.23	12.06	0.68	0.79
hard 3	False	False	44.75 ± 0.54	3.35 ± 0.06	213.06	9.43	0.34	0.54
hard 4	True	True	13.60 ± 0.09	3.25 ± 0.04	218.98	10.41	0.34	0.73
hard 5	False	True	56.57 ± 4.26	3.25 ± 0.06	208.30	10.15	0.50	0.68

A simulation environment with a forest of cylinders of random sizes, locations, and heights is generated in a corridor 50 m long, and 10 m wide. The start and goal locations are free, but a safe trajectory between these may not exist in the challenging map environments with large numbers of obstacles. Since the DMP is complete, the quadrotor will continue to explore to find a path until it successfully finds a path to the goal.

Perception: The quadrotor is equipped with a front-facing Intel Realsense D455 camera, which has a limited field of view of $87^\circ \times 58^\circ$, and a limited sensing range of 8 meters, operating at 30 FPS. The incoming depth maps are fused into a Euclidean-Signed Distance Field (ESDF) representation using the NvBlox package [43] at a resolution of 7.5 cm.

Path Planner: From the ESDF, a 2D slice of the obstacle geometry between 0.8-1.2 m height is extracted. The path to the goal location is planned using the Distance Map Planner (DMP) [9], a computationally efficient alternative to A* search which also pushes the path away from the obstacles. Unknown cells are treated as free cells. The planner takes less than 30 ms to replan trajectories, and is operated at 5 Hz. Given a desired linear travel speed $v = 2$ m/s, time is allocated to each leg of the returned path to construct the trajectory. This trajectory is not dynamically feasible for the quadrotor's nonlinear dynamics.

Safety Filtering: In our implementation of `gatekeeper`, a $4 \times 4 \times 2$ meter block centered on the quadrotor is extracted from the ESDF, and a convex polyhedron representing the safe region is constructed using the DecompUtil [9]. Unknown cells are treated as obstacles. This convex polyhedron is the safe set $\mathcal{B}_k$ used in `gatekeeper`. In these experiments, the environment is assumed static, but is unknown at the start of the run - as new regions are observed, the perceived safe set expands to include new regions.

Next, `gatekeeper` (as described in Algorithm 1) is used to convert the nominal trajectory into a dynamically feasible and safe trajectory for the quadrotor to follow. The backup controller used is a stopping controller, that drives the velocity and acceleration (and higher derivatives) of the quadrotor to zero. For any trajectory p^{can} to be valid, it must (A) lie within the safe polyhedron mentioned above (accounting for the grid resolution (7.5 cm), the quadrotor's radius (0.15 m), and a robustness margin $R = 0.1$ m), (B) terminate within the safe polyhedron accounting for the quadrotor radius and a robustness margin $R + r = 0.2$ m, (C) terminate with zero speed and zero control input. These conditions guarantee that the quadrotor can hover indefinitely at the terminal position.[2]

Note, due to the safety filtering, the nominal trajectory may not be traversable, for instance when attempting to traverse through a narrow passage. When this occurs, `gatekeeper` publishes a virtual obstacle, allowing the path planner to replan alternative routes.

Tracking Controller: The last committed trajectory is interpolated and tracked using a geometric tracking controller [33], running at 250 Hz.

Benchmark.: Our implementation of `gatekeeper` is compared with an MPC safety filter. In the MPC filter, the following optimization problem is solved:

$$\begin{aligned} \underset{x \in \mathcal{X}^{N+1}, u \in \mathcal{U}^N}{\operatorname{argmin}} & \sum_{i=0}^{N} \|x_i - [p_k^{nom}]_i\|_Q^2 + \sum_{i=0}^{N-1} \|u_i - [u_k^{nom}]_i\|_R^2 \\ \text{s.t. } & x_{i+1} = Ax_i + Bu_i \\ & x_i \in \mathcal{B}_k \\ & x_0 = \hat{x}(t_k) \\ & x_N = x_{N-1} \end{aligned}$$

[2]We do not consider the quadrotor's limited battery life as a constraint here. In [44] we consider this constraint. In this case, the backup controller plans a path back to the charging station, and `gatekeeper` ensures that the back-to-base trajectory is safe and feasible.

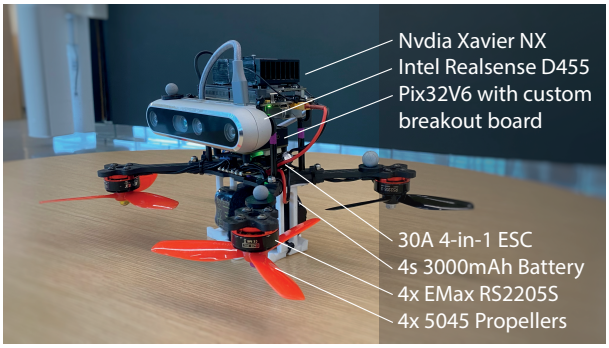

Fig. 7. Quadrotor used for experiments. A combination of off-the-shelf components and custom breakout boards is used to minimize weight and maximize performance.

where $[p_k^{nom}]_i = p_k^{nom}(t_k + i\Delta T)$ where $\Delta T = 0.02$ seconds is the discretization step size. A planning horizon of 2 seconds is considered, same as in our `gatekeeper` implementation. $[u_k^{nom}]_i$ the corresponding control input. To avoid solving a nonlinear program, the dynamics model assumed for the MPC safety filter is the linear double integrator. The set $\mathcal{B}_k$ is the same safe flight polyhedron used in `gatekeeper`. The initial condition is required to match with the estimated state, and the last constraint ensures that the quadrotor trajectory terminates within the horizon. The resulting problem is a quadratic program, and solved using OSQP [45].

Results: Fifteen different world environments were constructed with 3 difficulty levels, defined by the density of obstacle cylinders (Fig. 5). The quadrotors were tasked to fly a linear distance of 54 meters, at a desired speed of 1 m/s. Table III summarizes the performance with the MPC safety filter and using `gatekeeper`. Both the MPC and `gatekeeper` algorithms prevented collisions. In some cases neither MPC nor `gatekeeper` were able to reach the goal location, although `gatekeeper` was able to find a trajectory in more cases than MPC. MPC was consistently slower than the `gatekeeper`, requiring approximately 10x the computation time. Finally the average speed of the quadrotor was similar with both filters. Through this, we conclude that the performance of `gatekeeper` and MPC are similar, although `gatekeeper` computationally efficient, while additionally handling the nonlinear dynamics of the quadrotor.

C. Quadrotor Navigation (Experiments)

We also demonstrated the algorithm in experiments. A custom quadrotor was designed to optimize the payload, and maximize the flight time (Fig. 7). The quadrotor's wet weight is 820 g, and has a 15 min hover flight time. The perception, planning, and safety filtering steps were all performed on an onboard computer, the NVIDIA Xavier NX. The low-level control (a geometric tracking controller [33]) was performed on a Pix32V6, communicating with the Xavier through a serial connection. The goal destination and yaw angle was specified by a human operator.

As was the case for the simulations, the quadrotor uses the RGBD images from the Realsense D455 camera to construct a map of the environment using NvBlox, plans a trajectory using DMP, and filters the trajectory using `gatekeeper`.[3] The last committed trajectory is passed to the geometric tracking controller. Each of these steps were implemented in the same manner as described in Section VI-B.

Figure 8 shows snapshots of the top-down view of the map, and both the nominal and committed trajectories. See [39] for a video. Initially, the quadrotor tries to fly through a gap between the green and red obstacles towards the goal. However, since the gap is too small for `gatekeeper` to certify that it is safe to traverse through the gap,[4] new trajectories are not committed, and the quadrotor begins to execute its backup controller: stop and yaw. At this point, the nominal planner plans a new trajectory towards the right of the quadrotor, and `gatekeeper` allows a new trajectory to be committed. However, as the quadrotor approaches this gap, again it is too narrow to safely traverse. This repeats a few times before eventually the nominal planner finds the trajectory that indeed is safe to traverse, and the quadrotor reaches the goal destination.

In Fig. 9, the times at which nominal and committed trajectories are published are plotted. In our implementation using ROS2, when no new candidate trajectories are valid (as in (26)), the `gatekeeper` node does not publish a new committed trajectory, and therefore the controller continues to track the last committed trajectory. Therefore, in Fig. 9, the path-planner publishes at regular intervals, but there are gaps when `gatekeeper` is running but not publishing new committed trajectories. To allow the system to continue making progress towards the goal, we publish a virtual obstacle along the nominal trajectory when this happens, forcing the path planner to find a new trajectory to the goal. In this particular run, we observed 9 such instances.

The experiments also highlighted some limitations of `gatekeeper` that should form the basis for future study. In particular, suppose a nominal planner is poorly designed, and produces trajectories that are collision free, but not desirable. For example, suppose the nominal plan requires the quadrotor to jerk back and forth and yaw rapidly, at a sufficiently high frequency. Such a nominal trajectory could pass the validity check 9 (supposing the trajectory is collision-free), but could lead to a chattering behaviour of the quadrotor. In the future, we wish to investigate how to co-design the `gatekeeper` with planners and controllers to avoid such undesired trajectories. In our experiments, we have sometimes observed the nominal planner making large and abrupt changes in the nominal trajectory, but the quadrotor was able to track the committed trajectories.

Finally, further investigation into the design of backup controllers could yield interesting directions for future research. In our current implementation, the backup controller stops the quadrotor along the nominal trajectory. However, the position at which the quadrotor comes to a stop could be designed, for example, to maximize the visibility of the unknown regions.

[3] In hardware, the path planner replans once every 2 seconds. The ESDF is updated at 5 Hz, and `gatekeeper` is run at 20 Hz.

[4] The minimum gap required is the sum of (quadrotor diameter, 0.3 m) + (voxel size of map, 0.075 m) + (tracking radius r, 0.1 m) + (robustness radius R 0.1 m) = 0.58 m. The gap was measured to be 0.45 m across.

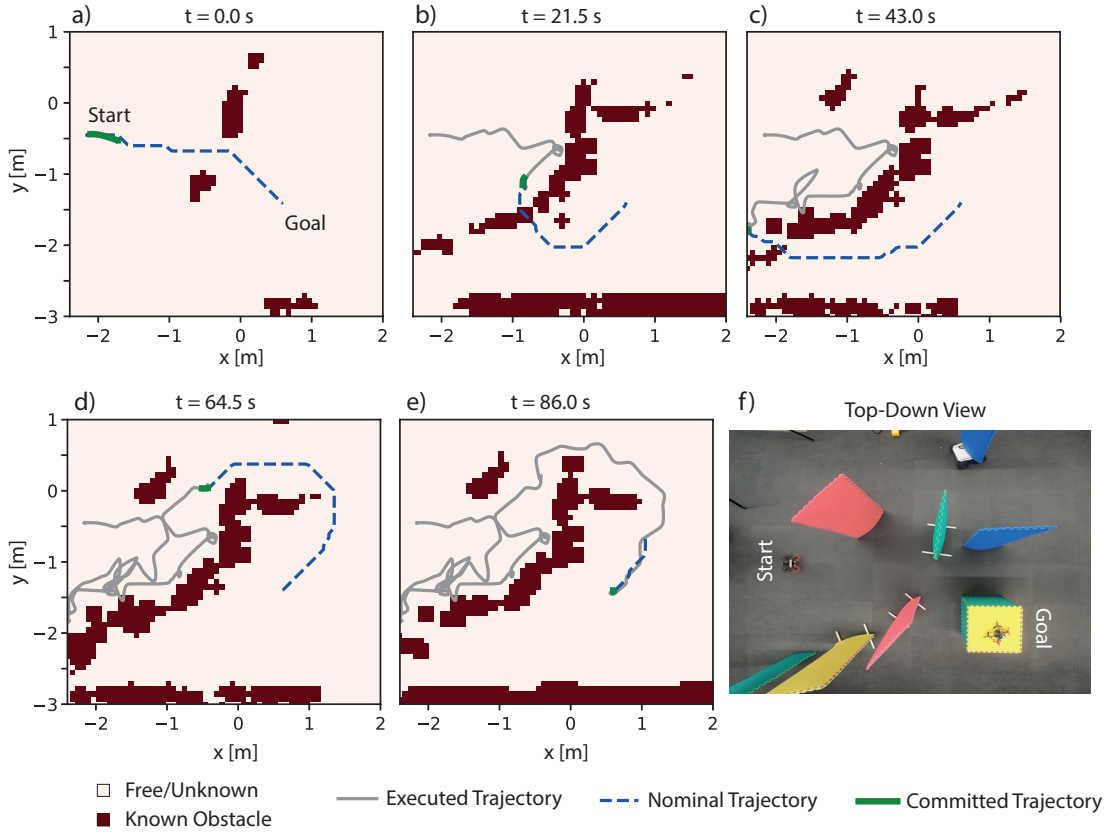

Fig. 8. (a-e) Snapshots of the map, nominal trajectory, committed trajectory, and executed path of the quadrotor. (f) Top-down view of the obstacle geometry.

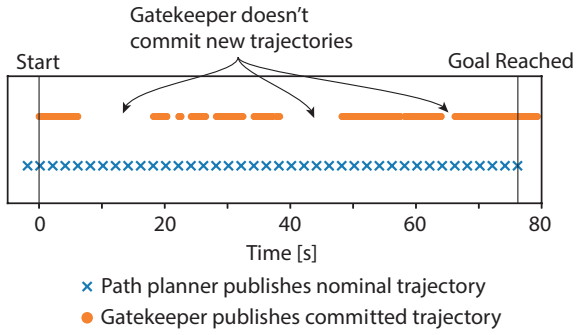

Fig. 9. Each dot (cross) represents a timepoint when the committed (nominal) trajectory is published. During the intervals where there are no new committed trajectories, `gatekeeper` is running but prevents unsafe trajectories from being committed. During these times, the controller continues to track the the last committed trajectory.

Such a backup controller would still maintain safety but also allow the quadrotor to reason more efficiently about the environment it is operating in. Furthermore, to operate this quadrotor in an environment with dynamic obstacles further attention will be needed on the design of backup controllers. If one were to assume a bounded speed at which the environment could move, the safe flight polyhedrons could quickly collapse to becoming non-empty. Using semantic maps, for example [46], might help to identify the dynamic parts of the environment, and overcome this issue.

VII. CONCLUSION

This paper proposes an algorithm ("`gatekeeper`") to safely control nonlinear robotic systems while information about dynamically-evolving safe states is received online. The algorithm constructs an infinite-horizon committed trajectory from a nominal trajectory using backup controllers. By extending a section of the nominal trajectory with the backup controller, `gatekeeper` is able to follow nominal trajectories closely, while guaranteeing a safe control input is known at all times. We have implemented the algorithm in a simulated aerial firefighting mission and on-board a real quadrotor, where we demonstrated `gatekeeper` is less conservative than similar methods, while remaining computationally lightweight. Various comparisons to state-of-the-art techniques are also provided. A key benefit of the `gatekeeper` approach is its applicability dynamic environments where the safe set is sensed online. This allows the method to be applied to a wide range of scenarios where only limited safety information is known, for instance, overtaking and merging scenarios for autonomous vehicles. Future directions involve developing more general methods to identify backup controllers, and understanding how the method can be applied in adversarial multi-agent settings.

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
[2] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE ICRA*. IEEE, 2011, pp. 1478–1483.

[3] D. J. Webb and J. Van Den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE ICRA*, 2013, pp. 5054–5061.

[4] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *16th Int. Symposium on Robotics Research*. Springer, 2016, pp. 649–666.

[5] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "Faster: Fast and safe trajectory planner for navigation in unknown environments," *IEEE TRO*, vol. 38, no. 2, pp. 922–938, 2021.

[6] A. Singletary, A. Swann, Y. Chen, and A. D. Ames, "Onboard safety guarantees for racing drones: High-speed geofencing with control barrier functions," *IEEE RAL*, vol. 7, no. 2, pp. 2897–2904, 2022.

[7] D. R. Agrawal, R. Chen, and D. Panagou, "Gatekeeper: Online safety verification and control for nonlinear systems in dynamic environments," in *2023 IEEE IROS*, 2023.

[8] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, no. 1, 2011, pp. 1114–1119.

[9] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE RAL*, vol. 2, no. 3, pp. 1688–1695, 2017.

[10] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference*, 2019, pp. 3420–3431.

[11] W. S. Cortez and D. V. Dimarogonas, "Correct-by-design control barrier functions for euler-lagrange systems with input constraints," in *2020 American Control Conference*, 2020, pp. 950–955.

[12] J. Breeden and D. Panagou, "Guaranteed safe spacecraft docking with control barrier functions," *IEEE Control Systems Letters*, vol. 6, pp. 2000–2005, 2021.

[13] M. Abate and S. Coogan, "Enforcing safety at runtime for systems with disturbances," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 2038–2043.

[14] C. Llanes, M. Abate, and S. Coogan, "Safety from in-the-loop reachability for cyber-physical systems," in *Proceedings of the Workshop on Computation-Aware Algorithmic Design for Cyber-Physical Systems*, 2021, pp. 9–10.

[15] D. R. Agrawal, H. Parwana, R. K. Cosner, U. Rosolia, A. D. Ames, and D. Panagou, "A constructive method for designing safe multirate controllers for differentially-flat systems," *IEEE Control Systems Letters*, vol. 6, pp. 2138–2143, 2021.

[16] T. Gurriet, M. Mote, A. Singletary, E. Feron, and A. D. Ames, "A scalable controlled set invariance framework with practical safety guarantees," in *2019 IEEE CDC*. IEEE, 2019, pp. 2046–2053.

[17] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Conference on Robot Learning*. PMLR, 2022, pp. 1724–1735.

[18] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[19] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 5115–5120.

[20] B. Zhou, J. Pan, F. Gao, and S. Shen, "Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight," *IEEE TRO*, vol. 37, no. 6, pp. 1992–2009, 2021.

[21] G. Chou, "Safe end-to-end learning-based robot autonomy via integrated perception, planning, and control," Ph.D. dissertation, 2022.

[22] P. Karkus, X. Ma, D. Hsu, L. P. Kaelbling, W. S. Lee, and T. Lozano-Pérez, "Differentiable algorithm networks for composable robot learning," *arXiv preprint arXiv:1905.11602*, 2019.

[23] R. C. Rothermel, *A mathematical model for predicting fire spread in wildland fuels*. Intermountain Forest & Range Experiment Station, Forest Service, US ..., 1972, vol. 115.

[24] P. L. Andrews, "The rothermel surface fire spread model and associated developments: A comprehensive explanation," *Gen. Tech. Rep. RMRS-GTR-371. Fort Collins, CO: US Dept. of Agriculture, Forest Service, Rocky Mountain Research Station. 121 p.*, vol. 371, 2018.

[25] H. K. Khalil, "Nonlinear systems, 3rd edition," *Prentice Hall*, 2002.

[26] D. R. Agrawal and D. Panagou, "Safe and robust observer-controller synthesis using control barrier functions," *IEEE Control Systems Letters*, vol. 7, pp. 127–132, 2022.

[27] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[28] T. Gurriet, A. Singletary, J. Reher, L. Ciarletta, E. Feron, and A. Ames, "Towards a framework for realizable safety critical control through active set invariance," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 98–106.

[29] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, "Robust control barrier–value functions for safety-critical control," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 6814–6821.

[30] J. Tordesillas and J. P. How, "Mader: Trajectory planner in multiagent and dynamic environments," *IEEE TRO*, vol. 38, no. 1, pp. 463–476, 2021.

[31] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ IROS*. IEEE, 2017, pp. 1366–1373.

[32] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *IEEE ICRA*. IEEE, 1990, pp. 384–389.

[33] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE CDC*. IEEE, 2010, pp. 5420–5425.

[34] P. Martin, R. M. Murray, and P. Rouchon, "Flat systems, equivalence and trajectory generation," *CDS Technical Report*, 2003.

[35] Y. Chen, M. Jankovic, M. Santillo, and A. D. Ames, "Backup control barrier functions: Formulation and comparative study," in *2021 60th IEEE Conference on Decision and Control*, 2021, pp. 6835–6841.

[36] S. Tonkens and S. Herbert, "Refining control barrier functions through Hamilton-Jacobi reachability," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022, pp. 13 355–13 362.

[37] T. Gurriet, P. Nilsson, A. Singletary, and A. D. Ames, "Realizable set invariance conditions for cyber-physical systems," in *2019 IEEE ACC*. IEEE, 2019, pp. 3642–3649.

[38] C. Rackauckas and Q. Nie, "`Differentialequations.jl`–a performant and feature-rich ecosystem for solving differential equations in `julia`," *J. Open Research Software*, vol. 5, no. 1, 2017.

[39] D. Agrawal and R. Chen, "Gatekeeper," https://github.com/dev10110/gatekeeper, 2022.

[40] A. Alessandri, P. Bagnerini, M. Gaggero, and L. Mantelli, "Parameter estimation of fire propagation models using level set methods," *Applied Mathematical Modelling*, vol. 92, pp. 731–747, 2021.

[41] M. G. Cruz and M. E. Alexander, "The 10% wind speed rule of thumb for estimating a wildfire's forward rate of spread in forests and shrublands," *Annals of Forest Science*, vol. 76, no. 2, pp. 1–11, 2019.

[42] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, 2016, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26054-9_23

[43] NVIDIA, "NvBlox." [Online]. Available: https://github.com/nvidia-isaac/nvblox

[44] K. B. Naveed, D. Agrawal, C. Vermillion, and D. Panagou, "Eclares: Energy-aware clarity-driven ergodic search," *arXiv preprint arXiv:2310.06933*, 2023.

[45] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[46] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020. [Online]. Available: https://github.com/MIT-SPARK/Kimera

APPENDIX

A. Worked Example for the Firewatch Scenario

This example demonstrates how the sets $\mathcal{S}(t), \mathcal{B}_k(t), \mathcal{C}_k(t)$ are related, using the firewatch mission. For simplicity, consider a double integrator system,

$$\dot{x} = Ax + Bu \tag{38}$$

where $x_{pos} = [x_1, x_2]^T$ is the position of the helicopter, and $x_{vel} = [x_3, x_4]^T$ is the velocity.

Say the fire starts at $t = t_0$, at location $p = [0, 0]^T$. The fire expands radially, with rate of spread $\sigma : \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$, i.e., $\sigma(p)$ is the rate of spread at a location $p \in \mathbb{R}^2$. To simplify the algebra, assume the RoS depends only on $\|p\|$, i.e, $\sigma(p_1) = \sigma(p_2)$ for any $\|p_1\| = \|p_2\|$. This means that the fire always spreads out uniformly in a circle.

Therefore, the safe set is time-varying, described by

$$\mathcal{S}(t) = \left\{ x : \|x_{pos}\| \geq \int_0^t \sigma(r(\tau))d\tau \right\} \ \forall t \geq 0 \tag{39}$$

where $r(t)$ is the radius of the fire at time $t \geq t_0$.

Since we don't know σ, we don't know $\mathcal{S}(t)$. Instead, we assume a reasonable upper bound: $\sigma(r) \leq 2$ m/s for all $r \geq 0$.

Therefore, at $t = t_0$, we can define an *perceived safe set*:

$$\mathcal{B}^0(t) = \{x : \|x_{pos}\| \geq 2(t - t_0)\} \ \forall t \geq t_0 \tag{40}$$

and clearly $\mathcal{B}^0(t) \subset \mathcal{S}(t) \ \forall t \geq 0$. Notice that $\mathcal{B}^0(t)$ is *not* a controlled invariant set for the double integrator.[5]

Suppose the system can directly measure the fire's radius. Let the k-th measurement be $r_k = r(t_k)$. This allows us to define the k-th *perceived* safe set:

$$\mathcal{B}_k(t) = \{x : \|x_{pos}\| \geq r_k + 2(t - t_k)\} \ \forall t \geq t_k \tag{41}$$

One can verify

$$\begin{aligned}
\mathcal{B}_k(t) &= \{x : \|x_{pos}\| \geq r_k + 2(t - t_k)\} \\
&= \left\{ x : \|x_{pos}\| \geq \int_{t_0}^{t_k} \sigma(r(\tau))d\tau + 2(t - t_k) \right\} \\
&\subset \left\{ x : \|x_{pos}\| \geq \int_{t_0}^{t_k} \sigma(r(\tau))d\tau + \int_{t_k}^{t} \sigma(r(\tau))d\tau \right\} \\
&= \left\{ x : \|x_{pos}\| \geq \int_{t_0}^{t} \sigma(r(\tau))d\tau \right\} \\
&= \mathcal{S}(t)
\end{aligned}$$

i.e. $\mathcal{B}_k(t) \subset \mathcal{S}(t)$ for all $t \geq t_k$.

Similarly, we can verify that for any $k \geq 0$,

$$\begin{aligned}
\mathcal{B}_{k+1}(t) &= \{x : \|x_{pos}\| \geq r_{k+1} + 2(t - t_{k+1})\} \\
&= \left\{ x : \|x_{pos}\| \geq r_k + \int_{t_k}^{t_{k+1}} \sigma(r(\tau))d\tau + 2(t - t_{k+1}) \right\} \\
&\supset \{x : \|x_{pos}\| \geq r_k + 2(t_{k+1} - t_k) + 2(t - t_{k+1})\} \\
&= \{x : \|x_{pos}\| \geq r_k + 2(t - t_k)\} \\
&= \mathcal{B}_k(t)
\end{aligned}$$

[5]Technically, a higher-order CBF could be used to design a QP controller that renders a subset of $\mathcal{B}^0(t)$ forward invariant, but this is only possible since $\mathcal{B}^0(t)$ is a sufficiently smooth function that we can analyze analytically.

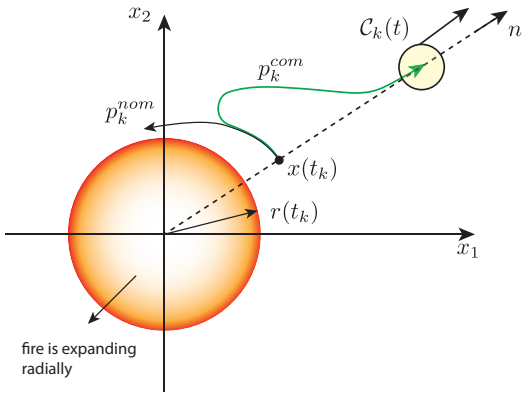

Fig. 10. Depiction of the scenario in the worked example.

i.e., $\mathcal{B}_k(t) \subset \mathcal{B}_{k+1}(t)$ for all $t \geq t_k$.

This proves that Assumption 1 is satisfied. Next, we define the backup controllers.

For any $k \in \mathbb{N}$, suppose the state is $x_k = x(t_k)$. The backup controller should drive the system radially away from the fire. Define n_k as the unit vector pointed at $x(t_k)$:

$$n_k = x_{pos}(t_k)/\|x_{pos}(t_k)\| \tag{42}$$

Notice that if the position followed the reference

$$p_{ref}(t) = (1 + r_k + 2(t - t_k))n_k \tag{43}$$

then the reference is moving radially at a speed of 2 m/s, and therefore faster than the maximum spread rate of the fire. Thus $p_{ref}(t)$ is a safe trajectory for all $t \geq t_k$.

This leads to the following backup controller:

$$\pi_k^B(t, x) = -K \left(\begin{bmatrix} x_{pos} \\ x_{vel} \end{bmatrix} - \begin{bmatrix} p_{ref}(t) \\ 2n_k \end{bmatrix} \right) \tag{44}$$

where $K \in \mathbb{R}^{2\times 4}$ is a stabilizing LQR gain for the double integrator. This controller stabilizes the system to $\mathcal{C}_k(t)$, where

$$\mathcal{C}_k(t) = \left\{ x : \left\| x - \begin{bmatrix} p_{ref}(t) \\ 2n_k \end{bmatrix} \right\| \leq 1 \right\} \tag{45}$$

This set is controlled invariant using the backup controller π_k^B. Geometrically, $\mathcal{C}_k(t)$ is a unit norm ball that is moving radially at 2 m/s in the n_k direction. Therefore, $\mathcal{C}_k(t) \subset \mathcal{S}(t)$ for all $t \geq t_k$, since the set is moving outwards radially at a speed higher than the maximum spread rate.

This example demonstrates how $\mathcal{S}(t)$, $\mathcal{B}_k(t)$, $\mathcal{C}_k(t)$ can be defined for a given problem. The main validation step in `gatekeeper`, will confirm whether after following the nominal trajectory over $[t_k, t_k + T_S)$, the system is able to safely reach $\mathcal{C}_k(t)$ using the backup controller π_k^B.

While the sets were described analytically here, in simulations they were represented numerically using SDFs.