Robotics and Autonomous Systems 115 (2019) 174–193

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance

Ross E. Allen [a,*,1,2], Marco Pavone [b,3]

[a] MIT Lincoln Laboratory, Lexington, MA, 02421, USA
[b] Stanford University, Stanford, CA, 94305, USA

HIGHLIGHTS

- Full-stack framework for path planning and obstacle avoidance for agile robots.
- Demonstrated on a quadrotor capable of dodging a fencing blade while flying indoors.
- Fusion of sampling-based planning, machine learning, and reactive control.

ARTICLE INFO

Article history:
Received 2 December 2017
Received in revised form 22 October 2018
Accepted 21 November 2018
Available online 31 December 2018

Keywords:
Motion planning
Kinodynamic
Real-time
Obstacle avoidance
Quadrotor
Unmanned aerial vehicle
Machine learning
Human–robot interaction

ABSTRACT

The objective of this paper is to present a full-stack, real-time motion planning framework for kinodynamic robots and then show how it is applied and demonstrated on a physical quadrotor system operating in a laboratory environment. The proposed framework utilizes an offline–online computation paradigm, neighborhood classification through machine learning, sampling-based motion planning with an optimal cost distance metric, and trajectory smoothing to achieve real-time planning for aerial vehicles. This framework accounts for dynamic obstacles with an event-based replanning structure and a locally reactive control layer that minimizes replanning events. The approach is demonstrated on a quadrotor navigating moving obstacles in an indoor space and stands as, arguably, one of the first demonstrations of full-online kinodynamic motion planning, with execution cycles of 3 Hz to 5 Hz. For the quadrotor, a simplified dynamics model is used during the planning phase to accelerate online computation. A trajectory smoothing phase, which leverages the differentially flat nature of quadrotor dynamics, is then implemented to guarantee a dynamically feasible trajectory.

1. Introduction

Due to their ease of use and development along with their wide range of applications in commercial, military, and recreational settings, quadrotor helicopters have become the focus of intense research in the last decade [1–3]. A standing problem in the field of quadrotor control is the achievement of real-time, high-velocity obstacle avoidance, as conceptually represented in Fig. 1. More generally, using the robotic motion planning nomenclature, this problem is referred to as *real-time kinodynamic motion planning* ("kinodynamic" meaning that system dynamics are taken into account during the trajectory planning process), which is an open challenge in robotics, not just for quadrotor control [4]. The challenge of real-time kinodynamic planning can be formulated into two questions that serve as the motivation for the work presented in this paper:

Motivating Question 1: Can we develop an algorithm/framework that provides real-time solutions to the kinodynamic planning problem for general dynamical systems?

Motivating Question 2: Given such a generalized approach to kinodynamic planning, can it be shown to work effectively on a real-world, physical system, such as a quadrotor?

In response to such motivating questions, this paper presents a *full-stack* approach for kinodynamic motion planning which includes: an offline–online computation paradigm, sampling-based

* Corresponding author.
E-mail addresses: ross.allen@ll.mit.edu (R.E. Allen), pavone@stanford.edu (M. Pavone).
1 PhD Candidate, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, USA (*affiliation during development of this work*).
2 Technical Staff, Controls and Autonomous Systems Engineering, MIT Lincoln Laboratory, Lexington, MA 02421, USA (*current affiliation. Note: no funding from MIT Lincoln Laboratory was used for this research*).
3 Assistant Professor, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, 94305, USA.

https://doi.org/10.1016/j.robot.2018.11.017
0921-8890/

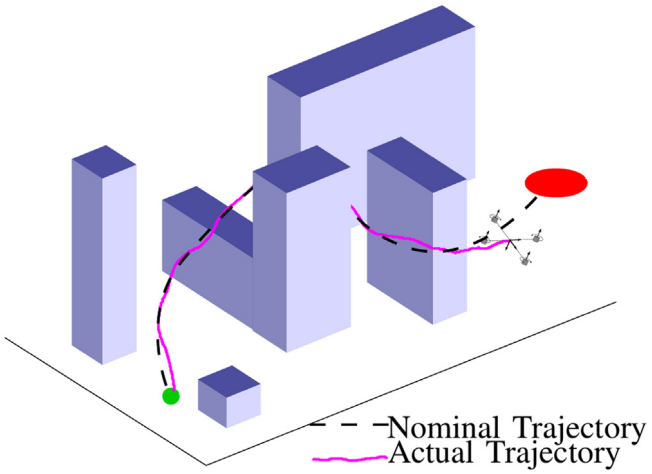

Fig. 1. Conceptual diagram of a quadrotor tracking a kinodynamic motion plan through an obstructed environment.

optimal motion planning, machine learning of reachable sets, trajectory smoothing, trajectory control, and event-based replanning. To further address the second motivating question we provide validating experiments of a quadrotor navigating static and dynamic obstacles. This is arguably one of the first – if not the first – demonstrations of truly real-time kinodynamic planning on a quadrotor system navigating a dynamic environment.

Related work

Throughout this paper we will detail each component of the full-stack planning framework and discuss its relation to the two motivating questions. First, however, let us build a foundation of prior work that sought to answer similar questions. There are two bodies of complementary, yet distinct, literature that are relevant to the work presented here: those works that address real-time motion planning in a general sense and those that focus on planning and control for quadrotors, specifically. We begin by discussing generalized planning and then move onto quadrotor-specific works.

Frazzoli et al. provided some of the pioneering work on real-time kinodynamic motion planning [5]. This work implemented the RRT algorithm with node connections achieved by concatenating a small set of motion primitives or "trim trajectories" between dynamic equilibrium points. Demonstrated on simulations of a small ground robot and a nonlinear helicopter model, the approach was successful in finding feasible trajectories through sparse obstacle sets in 10 s of milliseconds. The theory was even applied to dynamic obstacles; however computation times inflated to 10 s of seconds. The major shortcoming of this approach is the restrictive nature of "trim trajectories" that prevents the motion planner from achieving *completeness*, requires that subsequent trajectories are tied together at dynamic equilibrium points, and is highly reliant on the user to select appropriate motion primitives. For the helicopter example in the work of Frazzoli et al. only 25 different trim trajectories are used for node connections, all of which being constant speed, level or turning flight. Indeed a helicopter is capable of much more complex maneuvers than those considered. For any given set of motion primitives, it is argued that a pathological obstacle set could be devised that confounds this planning process. This effect is likely to blame for the significant increase in computation time for the dynamic obstacle sets: the motion primitives are inadequate for this specific case. The work presented in our paper includes a notion of time optimality and does not require the user to select specifically tailored motion primitives, therefore remaining more applicable to arbitrary obstacle sets. Furthermore, our work does not require a motion plan to be stitched together at dynamic equilibrium points, thus allowing for arbitrarily complex/acrobatic motion plans.

Leven and Hutchinson developed a real-time path planning framework for changing environments [6]. Their framework, which appears to be tailored to multi-link manipulator robots, relied on a *preprocessing phase* that generated a roadmap of the unobstructed configurations space (i.e., configuration space with no obstacles present). It then developed a mapping from nodes in the unobstructed configuration space to discrete cells in the workspace. When the online phase of the planner, referred to as the *query phase*, was initialized and obstacles were introduced, obstructed cells in the workspace could be mapped to corresponding nodes in the configuration space. These nodes were then removed from the roadmap and planning could occur on this augmented roadmap. This approach yielded impressive online planning times of less than one second.

While Leven and Hutchinson's framework is the most similar in form to that presented in our current work – consisting of a framework with offline and online phases to minimize real-time computations – there are several key differences. Foremost, Leven and Hutchinson's work centered on kinematically-constrained, but not differentially-constrained, robot manipulators. Furthermore they implement a "local planner" that consists of straight-line connections between sampled nodes in the configuration space, thus neglecting some of the kinematic constraints that are fundamental to the manipulators. For the straight-line connection assumption to be valid, they spend considerable time developing a *distance metric* that measures the "closeness" between two configurations. Leven and Hutchinson state that the ideal distance metric is swept volume, yet this is too expensive to calculate in real-time so a set of norms in the configuration and workspace are used instead [6]. In our work presented here, we seek to address differentially-constrained systems. To do so we must avoid straight-line approximations for state connections, instead relying on solving an optimal control problem between states (See 3.3 and 4.3). Therefore our distance metric indeed becomes optimal cost. As with Leven and Hutchinson's work, we face the problem that computing optimal cost may be too expensive to allow real-time computation. To this end we implement a machine-learning algorithm to approximate the optimal cost when real-time calculations are necessary (see 3.2). If our approach were applied to Leven and Hutchinson's work, they could directly estimate swept volume instead of relying on norm-based alternatives.

Singh et al. utilized contraction theory to extend the feedback motion planning and robust control work of Tedrake et al. to be implementable in an online fashion without prior knowledge of obstacle sets [7,8]. Given a nominal, feasible trajectory for a kinodynamic system navigating a cluttered environment, the work of Singh et al. produces a "tube" through the state space in which the system is guaranteed to remain even in the presence of bounded disturbances. While the work of Tedrake et al. and Singh et al. are important contributions to the robust control of kinodynamic systems in obstructed environments, they do not seek to answer the initial question of how to generate a nominal, feasible trajectory; instead considering this portion of the motion planning process as a "black box" [7]. In contrast, the work presented here explicates a framework for generating such a nominal trajectory in real-time. Thus this work can be thought of as the inner workings of such a "black box" referenced by Singh et al.

Since the publication of the authors' prior works [9–11], other groups have started to adopt similar approaches to solving the kinodynamic motion planning problem. Pendleton et al. develop

a "reachability-guided" sampling-based planning approach to differentially constrained problems that mirrors the authors' approach in several regards [12]. Similar to the approach presented in this paper and prior works, Pendleton et al. uses an offline, precomputation phase to generate a "reachable map" to be used in the online execution of the planner. The primary difference with the work of Pendleton et al. is that it employs control sampling and state propagation to approximate reachability sets, whereas our approach estimates reachability sets using machine learning algorithm (see 3.2).

In the theme of the second motivating question, the most relevant and progressive work in obstacle avoidance and control of quadrotors is, arguably, that of Richter et al. [13,14]. Relying on foundational work by Mellinger and Kumar [15], the work of Richter et al. demonstrated aggressive maneuvers for quadrotors flying in obstructed indoor environments. This was accomplished by generating a set of waypoints through the workspace and then developing a minimum-snap, polynomial trajectory connecting these waypoints. This minimum-snap trajectory produces a "graceful" flight pattern and guarantees dynamic feasibility [15]. Using the differentially flat dynamics of a quadrotor [15], the trajectory polynomials are used to generate analytical expressions for control inputs that are used in a feedforward fashion in the quadrotor flight controller [13]. While the work of Richter et al. represented an important step toward quadrotor planning and control, there remain several critical aspects yet to be achieved. Foremost, the planning algorithm used, RRT* [16], was not implemented in a real-time fashion. The planning phase was accomplished offline, with an a priori map of obstacles. This leaves the approach in Richter et al. unable to handle dynamic obstacles, which is illustrated in their demonstrations that only feature static obstacles. Furthermore, the RRT* algorithm used a simple straight-line metric for the initial planning phase to connect start and goal states; it did not account for the differential motion constraints of the quadrotor [13]. Therefore the initial planning phase produces waypoints that are minimum distance, not necessarily minimum time, to the goal. The snap-minimizing, polynomial trajectories – which guarantee dynamic feasibility – are only produced after the planning phase, implying that the generated trajectory might be significantly suboptimal. The work that is presented in this paper overcomes these shortfalls by employing a *kinodynamic* planner in a truly *real-time* fashion, with obstacle information only available during online execution.

Other works have made significant contributions to the theory of quadrotor control. Sreenath et al. developed a controller for a quadrotor carrying a cable-suspended load [17]. Hehn and D'Andrea demonstrated stabilization of an inverted pendulum balanced on a quadrotor [3]. Mellinger et al. devised a hybrid controller capable of perching a quadrotor on an over-vertical surface [18]. While important and impressive in their own right, these works are fundamentally controller designs that wholly neglect motion planning/obstacle avoidance. The work presented in this paper takes kinodynamic planning and flight control as subcomponents of a single problem and proposes a method for addressing both simultaneously.

Several papers have approached the topic of motion planning for quadrotors, even so far as real-time planning. Cowling et al. [19,20], and Bouktir et al. [21] both demonstrate a similar approach that combines trajectory optimization and trajectory control to accomplish high-speed collision avoidance of quadrotors. These papers, however, rely on a mathematically explicit representation of obstacles so that the flight controller can be customized to incorporate these specific obstacles. This limits the approach to a relatively limited number of obstacle configurations that are well defined ahead of time. The approach presented in our paper avoids the explicit mathematical representation of the obstacle space so as to be applicable to virtually any obstacle configuration and does not require obstacle information until online initiation.

Webb and van den Berg made a significant contribution to the field of kinodyanmic planning with their development of Kinodyanmic RRT* [22]. This work avoided the explicit obstacle representation found in Bouktir et al. [21] and Cowling et al. [19,20] and demonstrated kinodynamic planning for a simulated quadrotor system with linearized dynamics. The Kinodynamic RRT* algorithm is shown to execute in 10 s to 100 s of seconds; therefore failing to achieve real-time implementation.

An additional, important aspect in this field is validation on a physical system. The papers Frazzoli et al. [5], Leven and Hutchinson [6], Cowling et al. [19,20], Bouktir et al. [21], Webb and van den Berg [22] only provide simulation results, without a physical demonstration for validation. In contrast Landry produced physical demonstrations of planning and control of a quadrotor navigating a challenging, cluttered environment [23]. Landry's work, however, is not real-time, as it requires the entire problem to be solved ahead of time before online execution. As with the work of Richter et al., Landry's work is, therefore, limited to static obstacles. Grzonka et al. developed an autonomous quadrotor system capable of navigating highly obstructed indoor environments that executed a variant of the A* algorithm for real-time motion planning [24]. While this work demonstrated real-time planning, the quadrotor was flown at speeds low enough such that differential motion constraints of the quadrotor could be ignored. This implies that the motion planning algorithm demonstrated was in fact geometric and not kinodynamic, leaving it capable of only navigating static or very slow dynamic obstacles. In contrast, our work demonstrates a kinodynamic planner for quadrotor obstacle avoidance capable of navigating high-speed, even adversarial, dynamic obstacles.[4]

This paper is the culmination of the authors' prior works. In Allen et al. we introduce the concept of machine learning for rapid estimation of reachable sets for dynamical systems [9]. In our current work we extend this approach to the control-penalized double integrator (see Section 4.3) and show improved estimation accuracy. In Allen and Pavone [10] we first introduce the generalized framework for kinodynamic planning and show how online computation times can be reduced by several orders of magnitude for simulated systems. The subsequent paper applied the kinodynamic planning framework to a quadrotor robot and demonstrated real-time planning on this physical system in the presence of static obstacles [11]. In our current work we extend the real-time framework to dynamic obstacles by developing a "locally reactive" control layer and an event-based replanning scheme. Furthermore we provide extended simulation results to test the framework in a wider variety of obstacle sets than is possible in a laboratory environment, allowing statistical analysis of the framework performance.

[4] *A note on terminology*: throughout this work the terms "dynamic" and "adversarial" are used to describe the behavior of obstacles present in the environment which take the form of human actors interacting with the quadrotor. The term dynamic is meant to imply that the obstacles are moving through the environment, sometimes at high speed. The term adversarial is meant to imply that the obstacles may actively attempt to obstruct the path of the autonomous robot, as opposed to simply interfering by incident. It is important to understand the implications of these terms within this work so as not to confuse them with similar terms used in other literature. Firstly, planning literature sometimes refers to "dynamic obstacles" in order to imply that the motion planner attempts to create temporal models of obstacles in order to predict future states. This is not the case for this work; instead obstacles are modeled as static at every planning cycle and very rapid replanning is used to account for the fact that the obstacles are in fact dynamic. Secondly, machine learning literature may use the term "adversary" to imply that an autonomous agent may update its policy based on the observed behavior of an external agent. Within this work, no such adversary-modeling occurs. The human actor obstructing the motion plan is merely treated as a physical obstacle at each planning cycle.

Contribution

In the pursuit of addressing the two motivational questions, this work results in three key contributions. *Theoretical*: we show that machine learning of reachable sets for dynamical systems is an enabling concept for real-time motion planning. *Practical*: we present a newly developed reactive controller and event-based replanning structure which are synthesized with our prior work and existing theoretical results into a coherent, full-stack framework for kinodynamic planning. While the reactive controller and event-based replanner are pragmatic contributions – as opposed to more broadly applicable theoretical contributions – they are vital to the implementation of the planning framework on physical hardware. *Experimental*: arguably the first demonstration of truly real-time planning on a physical quadrotor capable of navigating dynamic, even adversarial, obstacles (see footnote 1).

Organization

The paper is structured as follows. Section 2 gives a formal definition of the kinodynamic planning problem we wish to solve. Section 3 seeks to address Motivating Question 1 by developing the real-time kinodynamic planning framework in a form that is generally applicable to dynamic systems. Section 3 then proceeds to detail two major components of the generalized framework: the sampling-based planning algorithm (Section 3.1) and machine learning for state neighborhood estimation (Section 3.2). Section 4 addresses Motivating Question 2 by detailing how the real-time kinodynamic planning framework and its subcomponents can be tailored to a quadrotor system in order to maximize performance and minimize computation time. Section 5 presents the experimental setup and results, validating the framework. Finally, in Section 6 we draw our conclusions and present directions for future research.

2. Problem statement

The optimal kinodynamic planning problem consists of the determination of a control function $\boldsymbol{u}(t) \in \mathbb{R}^m$, and corresponding state trajectory $\boldsymbol{x}(t) \in \mathbb{R}^n$, that minimize a cost function $\mathcal{J}(\cdot)$ while obeying control constraints, $\boldsymbol{u}(t) \in \mathcal{U}$, dynamical (differential) constraints, $\boldsymbol{f}[\dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{u}(t), t]$, and state (obstacle) constraints, i.e., $\boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}(t) \subseteq \mathcal{X}$ (where $\mathcal{X}$ denotes the state space and $\mathcal{X}_{\text{free}}(t)$ is the obstacle-free space which is a function of time to account for moving obstacles). The state at the final time must belong to a given goal region, i.e., $x(t_{\text{final}}) \in \mathcal{X}_{\text{goal}} \subseteq \mathcal{X}$. Formally, the problem can be posed as a continuous Bolza problem:

Optimal Kinodynamic Planning Problem:

$$\begin{aligned} &\text{Find: } \boldsymbol{u}(t) \\ &\text{that minimizes: } \mathcal{J}[\boldsymbol{x}(t), \boldsymbol{u}(t), t_{\text{final}}] \\ &\text{subject to: } \boldsymbol{u}(t) \in \mathcal{U} \\ &\qquad \forall t \in [t_{\text{init}}, t_{\text{final}}] \\ &\qquad \boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}(t) \\ &\qquad \forall t \in [t_{\text{init}}, t_{\text{final}}] \\ &\qquad \boldsymbol{f}_l \leq \boldsymbol{f}[\dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{u}(t), t] \leq \boldsymbol{f}_u \\ &\qquad \forall t \in [t_{\text{init}}, t_{\text{final}}] \\ &\qquad \boldsymbol{x}(t_{\text{final}}) \in \mathcal{X}_{\text{goal}} \end{aligned} \tag{1}$$

where $\boldsymbol{f}_l$ and $\boldsymbol{f}_u$ are the lower and upper bounds for the system dynamics described by a differential inclusion (note that, for generality, the dynamics are represented as a differential inclusion even though the quadrotor system discussed later is in fact just an ordinary differential system), t_{init} represents the fixed start time, and t_{final} represents the *free* final time.

Fig. 2. The real-time framework for kinodynamic planning and control is split into offline and online computation phases. Information that is generated in the offline phase serves two purposes: 1. develop a precomputed roadmap of the state space (however, this roadmap lacks information on obstacle placement which is not available until online initiation); and 2. train a machine learning algorithm to rapidly approximate state neighborhoods of newly sampled states that are not part of the precomputed roadmap (i.e. initial state and goal regions which are not known until online initiation). When obstacle information and terminal states are presented at online initiation, the online phase of the framework executes a sequence of computations that consist of 1. leveraging the machine learning algorithm to rapidly connect terminal states to the precomputed roadmap; 2. calculate the optimal trajectory between terminal states using an asymptotically optimal motion planning algorithm, such as *kino*-FMT*, which leverages the precomputed roadmap as a look-up table for dynamically feasible trajectories to connect states, performing collision checking in real-time; and 3. "post-process" the motion plan with a path smoother and translate the result to a low-level controller for execution on the robot.

Note that if $\mathcal{X}_{\text{free}}(t)$ can be explicitly represented, then the Optimal Kinodynamic Planning Problem in Eq. (1) may best be solved using existing optimal control methods, similar to what is presented in Schulman et al. [25]. However, we are concerned with cases where it is intractable to explicitly represent $\mathcal{X}_{\text{free}}(t)$ and we are only allowed the ability to perform query-based collision checks. The inability to explicitly represent $\mathcal{X}_{\text{free}}(t)$ is often the case for even modestly complex planning problems due to the immense challenge of representing obstacles in the *configuration space* [26].

Note that, for the quadrotor planning problem discussed in Section 4, we choose a minimum-time cost function, that is:

$$\mathcal{J}[\boldsymbol{x}(t), \boldsymbol{u}(t), t_{\text{final}}] = t_{\text{final}}. \tag{2}$$

3. Generalized real-time kinodynamic planning framework

In this section we seek to address Motivating Question 1 by developing a framework for solving the general form of the kinodynamic planning problem presented in Eq. (1). Note that this section expands on the authors' prior work by introducing, for the first time, an event-based replanning structure to account for dynamic obstacles [11].

In order to "construct" a real-time framework for solving the kinodynamic planning problem, we begin with the current state-of-the-art approach: sampling-based planning. Sampling-based

planning algorithms have become the accepted approach for planning in high-dimensional spaces. In a nutshell, the key idea behind sampling-based algorithms is to avoid the explicit construction of the configuration space (which can be prohibitive in complex planning problems) and instead conduct a search that either probabilistically or deterministically probes the configuration space with a sampling scheme. This probing is enabled by a collision detection module, which the motion planning algorithm considers as a "black box" [26]. In this way, a complex trajectory control problem is broken down into a series of many smaller, simpler optimal boundary value problems (OBVP)[5] that are subsequently evaluated *a posteriori* for obstacle constraint satisfaction and efficiently strung together into a graph (e.g., tree or roadmap). The primary hurdle for real-time implementability is that – without detailed information about a system's reachability set – a naive sampling-based planner may require the solution to $O(N_s^2)$ OBVPs during online execution, where N_s is the number of sampled states. It is prohibitively expensive to solve such a number of OBVPs in real-time even for the most modestly sized planning problems [30].

To address this we wrap a sampling-based planner in a real-time framework, given in Fig. 2, that minimizes the number of OBVPs that need to be solved online. The broad structure of our framework, featuring an offline–online computation paradigm, has similarities to that presented by Leven and Hutchinson [6]. However the details of framework's construction and subcomponents, which are considered a novel contribution of this work, differ significantly from that of Leven and Hutchinson.

The "philosophy" of our framework can be condensed to:

> *efficiency through machine learning, decision making through optimal control, precomputation when possible.*

To elaborate more, the framework (originally proposed in our earlier work [10] and further expanded in [11]) splits computation into offline (Algorithm 1) and online (Algorithm 2) phases. During the offline phase the subroutine `Sample` quasi-randomly draws N_s samples from the continuous state space, without any regard to obstacle locations, which are unknown until online initiation. `SampleData` randomly draws N_{pair} states – with replacement and such that $N_{\text{pair}} \leq N_s(N_s-1)$ – from the discrete set of sampled states V, and stores them in two sets A and B. The N_{pair} samples stored in A and B are then paired and OBVPs are solved for each pair; storing the solutions for use during the online phase in a look-up table titled `Cost`. The OBVP solution subroutine, `SolveOBVP`, which is often referred to as a "steering function" in the motion planning literature, is discussed in Section 3.3 for a general system and Section 4.3 for the quadrotor-specific implementation. The look-up table `Cost` can equivalently be thought of as a precomputed, unobstructed roadmap (i.e. it is wholly ignorant of obstacle information which is not available until online initiation) through the state space. During the offline phase, a support vector machine (SVM) classifier, referred to as `NearSVM`, is trained using the look-up table `Cost`. The SVM provides query-based estimates of cost-limited reachable sets (i.e., *neighborhoods*) and is discussed in further details in Section 3.2. The cost threshold of the reachable set, often referred to a "neighborhood radius" in the motion planning literature, is a user-defined value J_{th}.

[5] Note that not all sampling-based planners require the solution to optimal boundary value problems. State space exploration for the RRT algorithm is often achieved by employing a forward dynamic propagator based on randomized or deterministically chosen control inputs [27]. These techniques are prone to "wander" through the state space, lacking the optimality guarantees of algorithms such as RRT*, PRM*, and FMT* [16,28]. Li et al. developed the STABLE SPARSE RRT (SST) algorithm that achieves optimality guarantees without requiring OBVP solutions, only a forward dynamic propagator, but execution times for a quadrotor system are on the order of 100 s of seconds which is too slow for real-time implementation [29].

Algorithm 1 Offline Phase for the Kinodynamic Motion Planning Framework

```
V ← Sample(𝒳, N_s)
A ← SampleData(V, N_pair, replace)
B ← SampleData(V, N_pair, replace)
Cost ← SolveOBVP(A, B)
NearSVM ← TrainClassifier([A, B], Cost(A, B), J_th)
```

At the initiation of the online phase, obstacle data is presented along with the start state, x_{init}, and goal region, $\mathcal{X}_{\text{goal}}$.[6] A set of N_{goal} states are sampled from the goal region and stored in the discrete set X_{goal}. The SVM classifier is used to rapidly approximate the outgoing neighborhood of x_{init} and the incoming neighborhood of $\mathcal{X}_{\text{goal}}$ among the pre-sampled states; storing the sets in $N_{\text{init}}^{\text{out}}$ and $N_{\text{goal}}^{\text{in}}$, respectively (see Section 3.2 for discussion on outgoing and incoming neighborhoods). OBVPs are then solved from x_{init} and $\mathcal{X}_{\text{goal}}$ to their nearest neighbors and the solutions are stored in the look-up table. Note that this reduces the number of online OBVPs to be solved from $O\left(N_s^2\right)$ to $O(1)$. This reduction in online OBVP solutions is one of the defining characteristics of the planning framework that enables real-time execution[7]!

The sampling-based planner, *kino*-FMT*, is then called to return the optimal trajectory through the set of sampled states, V, using the look-up table, or "roadmap", `Cost`. Though many candidate sampling-based planners could be used to compute a trajectory across this roadmap, we rely on the asymptotically-optimal FMT* algorithm for its efficiency (see [28] for a detailed discussion of the advantages of FMT* over state-of-the-art counterparts; see [31] for its kinodynamic extension). The Kinodynamic Fast Marching Trees algorithm (*kino*-FMT*) (adapted from [31]) leverages the roadmap to efficiently determine the optimal sequence of sampled states to connect x_{init} and $\mathcal{X}_{\text{goal}}$, performing collision checking in real-time (see Section 3.1).

Algorithm 2 Online Phase for the Kinodynamic Motion Planning Framework

```
X_goal ← Sample(𝒳_goal, N_goal)
N_init^out ← NearSVM(x_init, V\{x_init}, J_th)
N_goal^in ← NearSVM(V\{X_goal}, X_goal, J_th)
for x ∈ V do
    if x ∈ N_init^out then
        Cost ← SolveOBVP(x_init, x)
    if x ∈ N_goal^in then
        Cost ← SolveOBVP(x, X_goal)
Path ← kino-FMT*(V, Cost, x_init, X_goal)
return SmoothPath(Path)
```

Finally the sequence of states generated by *kino*-FMT* can be used as a set of waypoints for a path smoothing algorithm. For the quadrotor system discussed in Section 4 the path smoothing

[6] If this information was available a priori, than all computations could be performed offline and the real-time implementation would become irrelevant.

[7] It is noted that most modern sampling-based motion planners attempt $O(N_s \log N_s)$ state connections, where N_s is the number of sampled states. This result, however, is based on the assumption that state connections are only attempted between states that are within a user-specified neighborhood of one another. As previously noted, this assumption breaks down for many kinodynamic systems where explicit neighborhood evaluation may require solving optimal boundary value problems, which is identical to evaluating a state connection. Under these circumstances a "bare" sampling-based planner (i.e. one that does not leverage other components of the framework presented in this paper) would indeed require $O\left(N_s^2\right)$ state connections; not the $O(N_s \log N_s)$ that is often quoted. Thus why we use $O\left(N_s^2\right)$ as the benchmark to show the improvement in computational performance offered by our framework.

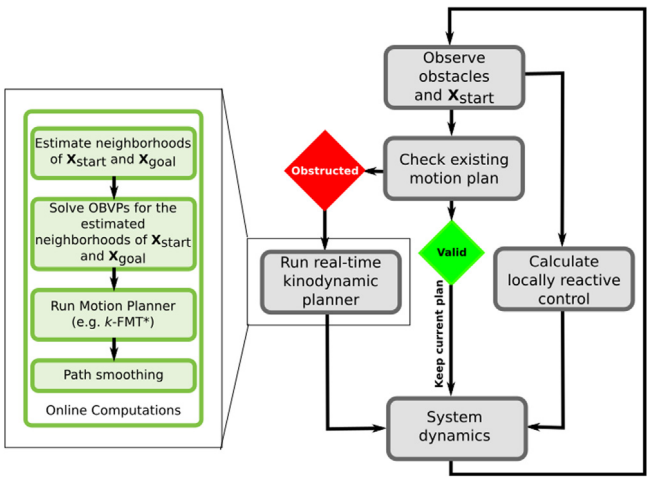

Fig. 3. The event-based replanning structure used to account for dynamic obstacles works by continuously checking the most recent motion plan and re-executing the online phase of the kinodynamic planning framework when necessary. This approach is termed "event-based" because the online portion of the planning framework is only executed when the robot becomes obstructed from the existing motion plan, thus indicating a dynamic obstacle has interfered and a new motion plan is required.

algorithm generates a minimum-snap, dynamically feasible trajectory for the (see Section 4.4). Mapping the differentially flat output variables from the smooth trajectory back to the full state and control space (Section 4.5), we can provide feedforward terms to the flight controller (Section 4.6).

To handle dynamic obstacles we must develop a replanning structure that recomputes the kinodynamic motion plan as the environment evolves. We choose to implement an event-based replanner where the existing solution trajectory is continuously checked for collisions with obstacles and replanning is only initiated once the existing plan becomes obstructed. This replanning structure is represented in Fig. 3. This event-based replanning is in contrast to a purely time-based, receding horizon replanner more typical for model predictive control. The event-based structure minimizes the number of replanning events which is desirable since even minor communication latency can cause overly aggressive maneuvers when transitioning from one solution trajectory to another; see Section 5.3 for more discussion.

To further reduce the number of replanning events and provide more "graceful" behavior in proximity to dynamic obstacles we also implement a locally reactive controller. This controller is inspired by the concept of potential fields where nearby obstacles impose a virtual, repelling force on the autonomous system [32]. This reactive controller is represented in Fig. 3 and is discussed further in Section 4.6. It is important to note this locally reactive controller is not necessary for the fundamental objective of real-time planning in dynamic environments which is achieved solely based on computation times of the real-time framework. In experimentation, however, it was shown to greatly improve performance, therefore it is discussed in this paper.

We now present the mathematical details for each of the framework components (to make the paper self-contained, we also state a number of results already available in the literature).

3.1. Sampling-based planner

The sampling-based motion planner at the core of our real-time framework is a kinodynamic variant of the Fast Marching Tree (FMT*) algorithm [28], and is presented in pseudo-code in Algorithm 3. The algorithm works by expanding a tree, stored in a set of edge connections E, along the minimum cost-to-come front through the pre-sampled set of states V. The frontier of the tree is stored in set H and unconnected samples are stored in set W.

For each iteration of the algorithm, the minimum cost-to-come sample z is used as a pivot for exploration. The forward-reachable set of z among the sampled states V is stored in the discrete set N_z^{out}. The intersection of N_z^{out} and set W is determined and the result is stored in set X_{near}. Each sample, $x \in X_{\text{near}}$, represents a candidate for expansion of the tree. For each candidate x the backward reachable set among sampled states is determined and saved as set N_x^{in}. The set Y_{near} is determined as the intersection of H and the backward reachable set of x, N_x^{in}. The sample $y_{\min} \in Y_{\text{near}}$ represents the optimal connection point (assuming no obstacles) between x and the existing tree. If the connection from $y_{\min}$ to x is free of collisions with obstacles, as determined by function `CollisionFree`,[8] then the $(y_{\min}, x)$ edge is added to the tree, x is added to the frontier set H and removed from W. Once all nodes in X_{near} are analyzed, the pivot node z is removed from the frontier set and the process is repeated. The algorithm succeeds in finding a path from x_{init} to $\mathcal{X}_{\text{goal}}$ as soon as the current pivot, z, is an element of $\mathcal{X}_{\text{goal}}$. If the frontier set H ever becomes empty, then *kino*-FMT* reports failure. The (asymptotic) optimality properties of FMT* (and its kinodynamic variants) are discussed in [28,31,33].

Algorithm 3 Kinodynamic Fast Marching Tree Algorithm (*kino*-FMT*)

$V \leftarrow V \cup \{x_{\text{init}}\} \cup \{X_{\text{goal}}\}$
$E \leftarrow \emptyset$
$W \leftarrow V \backslash \{x_{\text{init}}\}$; $H \leftarrow \{x_{\text{init}}\}$
$z \leftarrow x_{\text{init}}$
while $z \notin \mathcal{X}_{\text{goal}}$ **do**
  $N_z^{out} \leftarrow \texttt{Near}(z, V \backslash \{z\}, J_{\text{th}})$
  $X_{\text{near}} = \texttt{Intersect}(N_z^{\text{out}}, W)$
  for $x \in X_{\text{near}}$ **do**
    $N_x^{\text{in}} \leftarrow \texttt{Near}(V \backslash \{x\}, x, J_{\text{th}})$
    $Y_{\text{near}} \leftarrow \texttt{Intersect}(N_x^{\text{in}}, H)$
    $y_{\min} \leftarrow \arg\min_{y \in Y_{\text{near}}} \{\texttt{Cost}(y, T = (V, E)) + \texttt{Cost}(\overline{yx})\}$
    if $\texttt{CollisionFree}(y_{\min}, x)$ **then**
      $E \leftarrow E \cup \{(y_{\min}, x)\}$
      $H \leftarrow H \cup \{x\}$
      $W \leftarrow W \backslash \{x\}$
  $H \leftarrow H \backslash \{z\}$
  if $H = \emptyset$ **then**
    return Failure
  $z \leftarrow \arg\min_{y \in H} \{\texttt{Cost}(y, T = (V, E))\}$
return $\texttt{Path}(z, T = (V, E))$

3.2. Machine learning of neighborhoods

This section details the use of machine learning algorithms for rapid approximation of reachable sets of dynamical systems; i.e. "state neighborhoods". We argue that this concept – which was introduced in the authors' prior work [9] – stands an important contribution of this body of work.

When the terminal states, x_{init} and $\mathcal{X}_{\text{goal}}$, are introduced at online initiation they must be connected to the pre-sampled states before the motion planner can execute. Naively connecting the terminal states to all pre-sampled states would require $O(N_s)$ calls to `SolveOBVP`, which is prohibitively many to execute in real-time. Instead we seek to only connect the terminal states with

[8] Note that the details of the collision checking function `CollisionFree` are outside the scope of this work, therefore not discussed. For our purposes it is considered a "black box" function that reports whether a given path or trajectory intersects an obstacle.

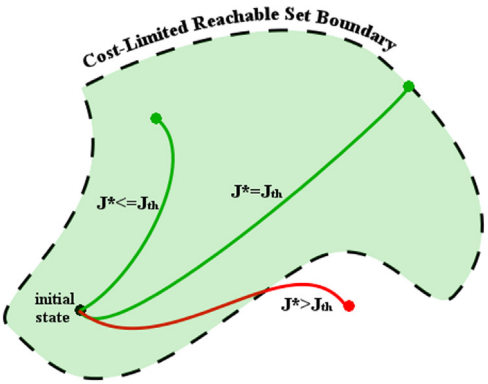

Fig. 4. Conceptual representation of a cost-limited reachable set for a notional 2D dynamical system. Formally, a (forward) cost-limited reachable set is the set of states that can be reached from a given state with a cost bounded above by a given threshold (denoted as J_{th}).

their nearest neighbors, as defined by the cost-limited reachable set (see Fig. 4). By limiting edge connections from the terminal states to a fixed number of states in their respective neighborhoods we have effectively reduced the number of online OBVPs to $O(1)$. *This reduction in online OBVPs lies at the core of achieving real-time execution of a kinodynamic planner.*[9]

A conceptual diagram of a cost-limited reachable set, i.e. *neighborhood*, of a given state is represented in Fig. 4. The mathematical definition of the "outgoing neighborhood" or *forward* cost-limited reachable set of a state $\boldsymbol{x}_a$ is:

$$R^{\text{out}}(\boldsymbol{x}_a, \mathcal{U}, J_{\text{th}}) := \{\boldsymbol{x}_b \in \mathcal{X} \mid \exists \boldsymbol{u} \in \mathcal{U} \text{ and } \exists t' \in [t_0, t_f] \text{ s.t. } \boldsymbol{x}(t') = \boldsymbol{x}_b \text{ and } \mathcal{J}^* \leq J_{\text{th}}\}, \tag{3}$$

where J_{th} is a user-defined cost threshold. In plain English, the forward reachable set is the union of all states $\boldsymbol{x}_b \in \mathcal{X}$ such that the optimal cost, $\mathcal{J}^*$, to steer the system from $\boldsymbol{x}_a$ to $\boldsymbol{x}_b$ is less than the cost threshold J_{th}. Also of importance is the concept of an "incoming neighborhood" or backward reachable set. The backward reachable set of state $\boldsymbol{x}_b$ is the union of all states, $\boldsymbol{x}_a$, such that $\boldsymbol{x}_b$ is in the forward reachable set of $\boldsymbol{x}_a$.

In general the determination of reachability sets is a computationally-expensive problem [34], therefore the real-time planning framework applies an approximation to the reachable sets based on machine learning. During the offline phase a support vector machine (SVM) is trained with data stored in `Cost` and provides a query-based classification of nearest neighbors. This approach leverage the authors' prior work [9], which demonstrated the accuracy and efficiency of this procedure for a number of nonlinear dynamical systems. To elaborate, we seek a function that makes a simple, binary discrimination:

> *is the optimal cost to traverse from an arbitrary state $\boldsymbol{x}_a$ to an arbitrary state $\boldsymbol{x}_b$ less than a given threshold J_{th}, or not?*

To develop such a function, we leverage the data in `Cost` to provide *training examples*. A training example consists of a initial state $\boldsymbol{x}_a$, final state $\boldsymbol{x}_b$, and optimal cost of traversal between the two. For each training example $i = 1, \ldots, N_{\text{train}}$ where $N_{\text{train}} \leq N_{\text{pair}}$, the initial and final states are concatenated into an attribute vector $\mathbf{p}^{(i)}$. If the optimal cost of the training example is less than the user-defined threshold, J_{th}, then it is given a label $y^{(i)} = +1$; otherwise it is given label $y^{(i)} = -1$. The training of the SVM is accomplished with the optimization given in Eq. (4) [35]:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{N_{\text{train}}} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N_{\text{train}}} y^{(i)} y^{(j)} \alpha_i \alpha_j K\left(\mathbf{p}^{(i)}, \mathbf{p}^{(j)}\right) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \ \ i = 1, \ldots, N_{\text{train}} \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{4}$$

where the α_i's are Lagrange multipliers, C is a user-defined parameter that relaxes the requirement that the training examples be completely separable, and $K(\cdot)$ is the kernel function. The vectors corresponding to non-zero Lagrange multipliers α_i's are the *support vectors*. For this work the *kernel function*, K, has the form

$$K(\mathbf{p}^{(1)}, \mathbf{p}^{(2)}) = \left(\phi\left(\mathbf{p}^{(1)}\right)^{\mathrm{T}} \phi\left(\mathbf{p}^{(2)}\right) + c\right)^p,$$

where ϕ is a nonlinear mapping of the attribute vector to a *feature vector* (see Table 4 for an example of feature vector used in this work), c is a weighting parameter between first and second order terms, and p is the kernel order chosen by the user. Once the support vectors are obtained, predictions on reachability for a new OBVP, paramaterized by $\tilde{\mathbf{p}}$, can be made with the predictor

$$\text{sgn}\left(\sum_{i=1}^{N_{\text{train}}} \alpha_i y^{(i)} K\left(\mathbf{p}^{(i)}, \tilde{\mathbf{p}}\right) + b\right), \tag{5}$$

where b is a bias term that is determined as a function of the Lagrange multipliers [35].

To contrast this approach with that of prior literature, Leven and Hutchinson used a set of norms in the workspace and configuration space as a rough surrogate for their desired distance metric of swept volume [6]. If our machine learning approach were applied to the work in [6], swept volume reachable sets could be directly approximated instead of having to devise a surrogate function. This allows the flexibility in our framework to be applied to a more general set of planning problems.

It is important to note that `NearSVM` is trained on data in `Cost` which is generated with no knowledge of obstacle placement. Therefore, `NearSVM` has no function in predicting obstacle collisions. Collision checking is solely within the realm of the sampling-based planner discussed in Section 3.1. Results on training and testing of the SVM classifier for a quadrotor system are presented in Section 5.5.

3.3. Solving optimal boundary value problems

In order to train the `NearSVM` algorithm for state neighborhood estimation, it is necessary to solve a large number of optimal boundary value problems for the system of interest in order to produce "true" examples of steering problems with known optimal cost. As previously noted, solving the large number of OBVPs is accomplished during the offline-phase of the planning framework, the results of which are stored in the `Cost` data structure. Until this point, though, we have not discussed how such OBVPs may be solved. Sections 4.2 and 4.3 detail the solution to OBVPs for a quadrotor system; however this is not generally applicable to all dynamic systems. In order to address Motivating Question 1 in Section 1, here we dedicate discussion to the solution of OBVPs for a general system with dynamics described by Eq. (1) (we again emphasize that this section does not consider the obstacles

[9] It is worth noting here that, while classification of reachable sets can be leveraged to guarantee execution of the real-time planning framework in $O(1)$ time, it does not guarantee that a solution will be found in $O(1)$ time. Due to the fact that the machine learning algorithms are not infallible and misclassifications occur, it is possible that no feasible planning solution will be found even if one exists. Under these circumstances, however, failure of the planning framework will still be reported with $O(1)$ OBVP evaluations. We spend much of Section 5.5 discussing the rate of such machine learning failures and their effect on implementation of the planning framework on physical robots.

presented in Eq. (1), just the differential constraints). Note that this section is a restatement of material described in the authors' prior work [9], but it is included here for completeness.

The solution to an optimal boundary value problem with differential constraints described in Eq. (1) is obtained in a two-step fashion. First the continuous-time problem is *time-discretized* and transformed into a nonlinear programming problem (NLP). Subsequently an NLP solution technique, such as sequential quadratic programming (SQP) [36, Ch. 18], is employed to solve the NLP and, therefore, approximate the solution to the original optimal control problem. The time-discretization method chosen for our work is the Chebyshev Pseudospectral Method [37] because of its accuracy and ability to extend to more general dynamic constraints (e.g. differential–algebraic equations or differential inclusions). This method works by approximating the state, $\boldsymbol{x}(t)$, and control, $\boldsymbol{u}(t)$, trajectories with nth degree Lagrange polynomials and then only enforcing the dynamic constraints, $\boldsymbol{f}_l \leq \boldsymbol{f}[\dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{u}(t), t] \leq \boldsymbol{f}_u$, at the Chebyshev–Gauss–Lobatto (CGL) points. This creates a NLP problem where the solution vector contains the values of the state and control variables at these CGL points. As shown in [37], the NLP transformation of an optimal boundary value problem is given as:

$$\begin{aligned} \underset{\boldsymbol{X},\boldsymbol{U}}{\text{minimize}} \quad & J^n\left[\boldsymbol{X}, \boldsymbol{U}, \tau_f\right] \\ \text{subject to} \quad & \boldsymbol{g}_l \leq \boldsymbol{g}\left[\mathbf{x}_k, \mathbf{u}_k, \tau_k\right] \leq \boldsymbol{g}_u \\ & \boldsymbol{f}\left[\frac{2}{\tau_b - \tau_a}\boldsymbol{d}_k, \mathbf{x}_k, \mathbf{u}_k, \tau_k\right] = 0 \\ & \mathbf{x}\left(\tau_0\right) = \mathbf{x}_a, \mathbf{x}\left(\tau_f\right) = \mathbf{x}_b \\ & \text{for } k = 0, \ldots, n, \end{aligned} \tag{6}$$

where J^n is the nth order approximation of the cost function, $\boldsymbol{x}_k \in \mathcal{X}$ and $\boldsymbol{u}_k \in \mathcal{U}$ are the state and control values at the CGL points, $\boldsymbol{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_n]$ and $\boldsymbol{U} = [\mathbf{u}_0, \ldots, \mathbf{u}_n]$, τ is a transformed time variable, $\boldsymbol{g}$ is a condensed representation of the state and control constraints, and $\boldsymbol{d}_k$ is a product of a differentiation operation. For brevity we omit some of the details of Eq. (6); however these details are well discussed in Fahroo and Ross [37]. Even though we have successfully discretized the OBVP, there is no known analytical solution to the resulting nonlinear programming problem. As previously mentioned, sequential quadratic programming (SQP) is a well-established technique for such problems, and although it offers no guarantees of a solution, the technique is commonly used [36, Ch. 18]. Attempting to solve the NLP presented in Eq. (6) with use of a SQP technique requires an initial guess. For the cases studied in the authors' prior work [9], a linear interpolation between boundary values proved sufficient for such a guess.

For the training of the machine learning algorithm in Section 3.2, any OBVPs with infinite cost – i.e. those with no feasible solution to the NLP given in Eq. (6) – should be neglected from the training set. To ensure a well-distributed sampling of the training queries, training data can be generated using the Halton sequence, assuming a hypercubical state space. It is worth again noting that the approach to solving OBVPs found here in Section 3.3 is later superseded by the quadrotor-specific approach found in Section 4; however Section 3.3 is included for completeness in addressing Motivating Question 1.

4. Real-time quadrotor planning framework

In this section we seek to address Motivating Question 2 by detailing the steps necessary to apply the generalized framework developed in Section 3 to a quadrotor system. Sections 4.1 and 4.2 present a nonlinear and linearized dynamics model of the quadrotor system, respectively. Section 4.3 then provides an analytical solution to boundary value problems described by the linearized quadrotor model in Section 4.2. Sections 4.4 and 4.5 detail the differentially flat properties of the quadrotor system and how this can be utilized for trajectory smoothing and control. Section 4.6 describes how the motion plan is communicated to and executed on the low level flight controller.

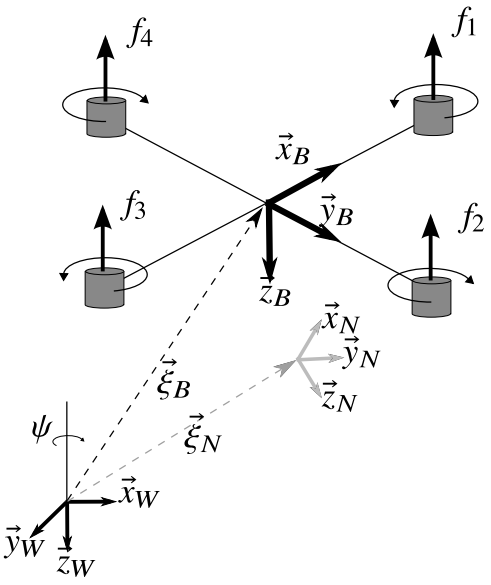

Fig. 5. Diagram of quadrotor dynamics with world (inertial), body, and nominal reference frames.

Note that Sections 4.1, 4.2, 4.3, 4.4, and 4.5 are included here for completeness but do not represent new contributions on our part. For the most part these sections constitute restatements of existing literature with slight modifications on notation to suit our purposes.

In contrast, Section 4.6 details a newly devised quadrotor flight controller architecture that blends control inputs from polynomial spline trajectories with a "locally reactive" forcing function to produce smooth, reliable path following behavior in the presence of dynamic obstacles. Section 4.6 constitutes an important contribution of this paper in terms of achieving physical demonstrations of kinodynamic planning and obstacle avoidance.

4.1. Nonlinear quadrotor dynamics

A quadrotor is modeled as an underactuated rigid body where net thrust is constrained along the $-\vec{z}_B$ axis (see Fig. 5). The diagram given in Fig. 5 represents the relevant coordinate frames and variables for the quadrotor planning and control problem. The world frame, W, is an inertial frame, which is implemented in our case with a North-East-Down (NED) orientation. The body-fixed frame, B, translates and rotates with the quadrotor. The nominal frame, N, is a target frame for trajectory tracking; therefore in perfect trajectory tracking $B = N$. The quadrotor dynamics are given in Eq. (7) [38]:

$$\begin{aligned} \dot{\vec{\xi}}_B &= \frac{d^W \vec{\xi}_B}{dt}, \\ \ddot{\vec{\xi}}_B &= mg\vec{z}_W - u_1\vec{z}_B, \\ \dot{R}_{BW} &= R_{BW}\widehat{\Omega}_{BW}, \\ J_B\dot{\vec{\Omega}}_{BW} &= \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \vec{\Omega}_{BW} \times J_B\vec{\Omega}_{BW}. \end{aligned} \tag{7}$$

The state vector is given by $\mathbf{x} = \left[\vec{\xi}_B, \dot{\vec{\xi}}_B, R_{BW}, \vec{\Omega}_{BW}\right]^{\mathrm{T}} \in \mathbb{R}^9 \times \mathrm{SO}(3)$ where $\vec{\xi}_B$ is the position of the body frame, $\dot{\vec{\xi}}_B$ is the velocity

of the body frame, R_{BW} is the rotation matrix from the body frame to the world frame, Ω_{BW} is the angular velocity of the body frame with respect to the world frame, and g is the gravity acceleration. The quadrotor mass is given by m. The control vector is given by $\mathbf{u} = \left[F_{z_B}, M_{x_B}, M_{y_B}, M_{z_B}\right] \in \mathbb{R}^4$ where F_{z_B} is the force applied along the body z-axis due to net thrust; and M_{x_B}, M_{y_B}, and M_{z_B} are the moments about the body x, y, and z axes, respectively, due to individual rotor thrusts or torque. Note that $\widehat{\cdot}$ denotes the *hat-map* (i.e., an isomorphism between 3×3 skew-symmetric matrices and vectors in $\mathbb{R}^3$) [38].

4.2. Approximate quadrotor dynamics

There are no known analytical solutions to the minimum-time optimal control problem under the quadrotor's nonlinear dynamics represented in Eq. (7). While numerical solutions are possible using the techniques to solve general OBVPs described in Section 3.3, we can reduce computation time by customizing our approach to the quadrotor system. To minimize online computation times we apply an *approximator-corrector* structure to our framework. The quadrotor is first approximated as a double integrator system, which allows analytical treatment for the *unobstructed* minimal-time control problem. These minimal-time control problems, which are subproblems to the overall planning problem, serve to connect edges in the sampling based planner; see Sections 3.1 and 4.3 for more details. At the end of the planning process, the solution trajectory is mapped, or "corrected", back into the fully nonlinear dynamics by leveraging the property of differential flatness; see Section 4.5 and Mellinger and Kumar [15]. The double integrator dynamics are given as

$$\begin{aligned} &\dot{\mathbf{x}}(t) = A\mathbf{x} + B\mathbf{u} + \mathbf{c} \\ &\text{where: } A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ I \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 \\ g \end{bmatrix}, \\ &\mathbf{x} = \begin{bmatrix} \vec{\xi}_B \\ \dot{\vec{\xi}}_B \end{bmatrix} \in \mathbb{R}^6, \quad \mathbf{u} = \ddot{\vec{\xi}}_B \in \mathbb{R}^3. \end{aligned} \tag{8}$$

Note that this approximator-corrector approach for the quadrotor dynamics is one of several trade-offs that arise from trying to address, simultaneously, both motivational questions stated in Section 1. The framework presented is indeed general enough to accommodate the fully nonlinear dynamics of the quadrotor, however, it is desirable to apply an approximation (along with the later correction) to improve online performance during physical demonstrations.

4.3. Solving the minimum-time double integrator OBVP

In contrast to other works such as Leven and Hutchinson [6] and Richter et al. [13] that use straight line connections between states, we require the solution to an optimal boundary value problem to connect sampled states. As explained in Section 4.2, we minimize computations by approximating our system as the double integrator given in Eq. (8). This approximation enables partial–analytical solutions to the optimal boundary value problem between two sampled states, which is executed in the `SolveOBVP` algorithm. The approximation is corrected for in Section 4.5. The results in this section come from the works [22,39].

To address control constraints on thrust, a control penalty term is added to the minimum-time cost function, that is:

$$\mathcal{J}[\mathbf{u}, \tau] = \int_0^\tau 1 + \mathbf{u}[t]^{\mathrm{T}} R_u \mathbf{u}[t]\, \mathrm{d}t, \tag{9}$$

where $R_u \in \mathbb{R}^{m \times m}$ is symmetric positive definite. For a fixed final time, τ, the optimal cost $\mathcal{J}^*$ for the control-penalized double integrator model is given in closed form by Eq. (10) where $R_u = w_R I$ and w_R is the control penalty weight [22,39]:

$$\mathcal{J}^*[\tau] = \tau + (\mathbf{x} - \bar{\mathbf{x}}[\tau])^{\mathrm{T}}\, \mathbf{d}[\tau]. \tag{10}$$

The corresponding control and state trajectories as functions of time t, for a fixed final time τ, are given in Eq. (11), respectively [22, 39]:

$$\begin{aligned} \mathbf{u}[t] &= R_u^{-1} B^{\mathrm{T}} \exp\left[A^{\mathrm{T}}(\tau - t)\right] \mathbf{d}[\tau], \\ \mathbf{x}[t] &= \bar{\mathbf{x}}[t] + G[t] \exp\left[A^{\mathrm{T}}(\tau - t)\right] \mathbf{d}[\tau], \end{aligned} \tag{11}$$

where

$$\begin{aligned} &\mathbf{d}[\tau] = G[\tau]^{-1} \left(\mathbf{x} - \bar{\mathbf{x}}[\tau]\right), \\ &G[t] = \frac{1}{w_R} \begin{bmatrix} t^3/3 & 0 & 0 & t^2/2 & 0 & 0 \\ 0 & t^3/3 & 0 & 0 & t^2/2 & 0 \\ 0 & 0 & t^3/3 & 0 & 0 & t^2/2 \\ t^2/2 & 0 & 0 & t & 0 & 0 \\ 0 & t^2/2 & 0 & 0 & t & 0 \\ 0 & 0 & t^2/2 & 0 & 0 & t \end{bmatrix}, \\ &\bar{\mathbf{x}}[t] = \exp\left[At\right] \mathbf{x}_0 + \left[0, 0, gt^2/2, 0, 0, gt\right]^{\mathrm{T}}. \end{aligned} \tag{12}$$

Note that Eqs. (10) and (11) require a *fixed* final time τ. The optimal final time, $\tau^* = \operatorname{argmin} \mathcal{J}[\tau]$, can be solved for via a bisection search of Eq. (10).

At this point the reader may question the need for the machine learning techniques discussed in Section 3.2 if state connections for quadrotor planning are to be resolved with a minimum-time double integrator boundary value problem for which the solution has just been given. In response to this potential question, two notes are to be made. First, the solution to the minimum-time OBVP is not entirely analytical, requiring a bisection search to optimize for time. While the bisection search is relatively fast compared to attempting the solution of a nonlinear OBVP, it is still slower than the machine learning approach; see Section 5.5 for numerical results. As a second note, this again highlights the balancing act attempted between Motivating Questions 1 and 2, whereby the general framework developed in Section 3 – that relies heavily on the use of machine learning to estimate state neighborhoods for systems that lack analytical solutions for OBVPs – is customized to a specific system that admits analytical treatment (i.e. the quadrotor) in order to maximize performance.

4.4. Minimum-Snap trajectory smoother

Trajectory smoothing is commonly implemented in motion planning to improve the quality of the trajectory returned by the planner. Furthermore, in our case, we need to correct for the double integrator approximation previously made. To this end we improve the sampling-based planner's solution computed via *kino*-FMT* by connecting the solution samples with a high-order polynomial spline. Building on Mellinger and Kumar's work [15], Richter et al. [13] formulate the spline determination as an unconstrained quadratic programming problem that minimizes the integral of the square of the *snap* (i.e. the 4th derivative of position); see Eq. (13). In the unconstrained formulation, derivatives at samples of the motion plan, i.e. waypoints, are left as free parameters for optimization. For completeness we present the essential results of Richter et al. as they are used in our current approach [13,14].

Our goal in this section is to determine the coefficients of M polynomials of order N. These polynomials form a spline that is continuous up to the 4th derivative and passes through the sampled states, or "nodes", of the solution trajectory determined in Section 3.1. While an infinite number of splines may exist that satisfy these conditions, we seek the spline that minimizes the integral of the square of the snap. Let us begin by considering

a single polynomial $P(t) = \sum_{n=0}^{N} p_n t^n$. The minimum-snap cost function for a single polynomial is defined as

$$J_{\text{snap}} = \int_0^T P^{(4)}(t)^2\, dt = \boldsymbol{p}^{\mathrm{T}} Q(T) \boldsymbol{p}, \tag{13}$$

where $Q(T)$ is the Hessian matrix of J_{snap} with respect to the polynomial coefficients, $\boldsymbol{p}$ is a vector of the $N+1$ polynomial coefficients, and T is the polynomial segment time which is determined by the kinodynamic planner. The superscript "(4)" implies the 4th derivative of the polynomial. Without derivation, the Hessian matrix is given as[10]

$$\begin{aligned} Q_{i,j}(T) &= 2\left(\prod_{k=0}^{3}(i-k)(j-k)\right)\frac{T^{i+j-7}}{i+j-7} \quad \text{for: } i \geq 4 \wedge j \geq 4, \\ Q_{i,j}(T) &= 0 \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{otherwise.} \end{aligned} \tag{14}$$

As previously mentioned, the polynomial is constrained at its terminal points, $t = 0$ and $t = T$, to the waypoints of the motion plan determined in Section 3.1. The derivatives of the polynomial at its terminal points can be fixed or left as free parameters for optimization. Even as free parameters, however, the derivatives must satisfy continuity between polynomials in the spline. These constraints can be encoded as the linear function

$$A\boldsymbol{p} = \boldsymbol{d} \tag{15}$$

$$A = \begin{bmatrix} A_0 \\ A_T \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_T \end{bmatrix}, \tag{16}$$

where the terms are given as

$$A_{0_{i,j}} = \begin{cases} \prod_{k=0}^{i-1}(i-k) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{17}$$

$$\boldsymbol{d}_{0_i} = P^{(i)}(0) \tag{18}$$

$$A_{T_{i,j}} = \begin{cases} \left(\prod_{k=0}^{i-1}(i-k)\right) T^{i-j} & \text{if } i \geq j \\ 0 & \text{if } i < j \end{cases} \tag{19}$$

$$\boldsymbol{d}_{T_i} = P^{(i)}(T) \tag{20}$$

Numerical stability can be achieved by reformulating the constrained problem represented in Eqs. (13) and (16) as an unconstrained optimization [13,14]. This is achieved by optimizing over the polynomial derivatives at the terminal points instead of the polynomial coefficients. Under this reformulation, Eqs. (13) and (16) become

$$J_{\text{snap}} = \boldsymbol{d}^{\mathrm{T}} A^{-\mathrm{T}} Q(T) A^{-1} \boldsymbol{d}, \tag{21}$$

and the polynomial coefficients are determined, a posteriori, via inversion of Eq. (15).

Now that we have formulated the optimization problem for a single polynomial, we must consider the optimization over the spline of M polynomials. To this end we form $A_{1\ldots M}$ and $Q_{1\ldots M}$ which are block diagonal matrices composed of the A and Q matrices for each segment. We could also simply concatenate the derivative vectors into a vector $\boldsymbol{d}_{1\ldots M}$, however it is desirable to separate this vector into components that are fixed and those that are free parameters of optimization. Therefore the derivative vector for the spline optimization is formed as

$$\boldsymbol{d}_{\text{total}} = \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}. \tag{22}$$

With this reordering of the derivative vector in Eq. (22), an ordering matrix C is required that preserves the proper relationships with the block matrices $A_{1\ldots M}$ and $Q_{1\ldots M}$. Furthermore, the ordering matrix C also encodes the enforcement of continuity of derivatives at intermediate waypoints. Now the minimum-snap cost function for the entire spline is given as

$$J_{\text{snap}} = \boldsymbol{d}_{\text{total}}^{\mathrm{T}} C A_{1\ldots M}^{-\mathrm{T}} Q_{1\ldots M} A_{1\ldots M} C^{\mathrm{T}} \boldsymbol{d}_{\text{total}}. \tag{23}$$

For simplicity, define the matrix $H = CA_{1\ldots M}^{-\mathrm{T}} Q_{1\ldots M} A_{1\ldots M} C^{\mathrm{T}}$ and partition it such that Eq. (23) can be written

$$J_{\text{snap}} = \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}. \tag{24}$$

Differentiating and setting to zero solves for the free derivatives at the waypoints

$$\boldsymbol{d}_{\text{free}}^{*} = -H_{22}^{-1} H_{12}^{\mathrm{T}} \boldsymbol{d}_{\text{fix}}. \tag{25}$$

Now that the derivatives at each waypoint are determined, the polynomial coefficients can be determined by inverting Eq. (15). This process is applied for the determination of four splines: x, y, and z positions and yaw. These splines correspond to the differential flat output variables discussed in Section 4.5.

It is important to note here that once smoothing is applied, the trajectory is no longer guaranteed to be collision free. Therefore it is necessary to perform an additional collision checking phase during the trajectory smoothing phase. If one of the polynomials in the spline is found to collide with an obstacle, then a new smoothed trajectory must be determined. This is accomplished by sampling the midpoint of the underlying motion plan solution which is guaranteed to be collision free (else it would have not been selected as a valid motion plan). The trajectory smoother than solves the minimum-snap optimization problem for $M + 1$ trajectory segments. This is repeated until the smoothed trajectory is collision free. See Richter et al. for more details [13,14].

4.5. Differentially flat mapping

The trajectory smoother from Section 4.4 produces polynomial splines for position and yaw that are continuous up to their fourth derivative. Mellinger and Kumar showed that the state and control variables for the nonlinear quadrotor dynamics can be expressed in terms of $\vec{\xi}_N$ and ψ_N and their derivatives up to fourth order; thus proving Eq. (7) represents a differentially flat system with flat output variables $\vec{\xi}_N$ and ψ_N [15]. This mathematical property proves that the smoothed trajectory from Section 4.4 is guaranteed to be dynamically feasible for the quadrotor; therefore correcting the double-integrator approximation made to solve the planning problem. For completeness we state the results of Mellinger and Kumar for the mapping from the flat outputs to the nominal state and control variables. Note that, while the following equations are taken almost directly from [15], there are some subtle coordinate frame changes.

The nominal position and velocity state variables are identically $\vec{\xi}_N$ and $\dot{\vec{\xi}}_N$, respectively. The thrust control variable is given as

$$u_{1_{\text{ff}}} = -\vec{z}_B \cdot \vec{F}_N, \quad \text{where: } \vec{F}_N = m\ddot{\vec{\xi}}_N - mg\vec{z}_W. \tag{26}$$

[10] Note that we diverge from Richter et al. by only considering the minimization on the 4th derivative, where Richter et al. leaves the formulation more general as a weighted sum of squares of derivatives. Furthermore, due to the fact that Richter et al. uses a geometric planner to determine waypoints, his approach requires a time allocation optimization to determine polynomial segment times, T [13,14]. In contrast, our work determines the polynomial segment times during the time-minimizing kinodynamic planning; see Section 3.1.

The subscript ff indicate that this thrust value appears as a feedforward term in the flight controller (Section 4.6). The nominal orientation matrix is given by the nominal frame axes represented in world coordinates:

$$\vec{R}_N = \left[{}^W\vec{x}_N,\ {}^W\vec{y}_N,\ {}^W\vec{z}_N\right], \tag{27}$$

where

$$\begin{aligned}\vec{z}_N &= -\frac{\vec{F}_N}{\|\vec{F}_N\|}\\ \vec{y}_S &= [-\sin\psi_N, \cos\psi_N, 0]^{\mathrm{T}}\\ \vec{x}_N &= \frac{\vec{y}_S \times \vec{z}_N}{\|\vec{y}_S \times \vec{z}_N\|}\\ \vec{y}_N &= \vec{z}_N \times \vec{x}_N.\end{aligned} \tag{28}$$

The nominal angular velocity vector is given by

$$\vec{\Omega}_{NW} = p_N\vec{x}_N + q_N\vec{y}_N + r_N\vec{z}_N, \tag{29}$$

where the individual components of the nominal angular velocity are

$$\begin{aligned}p_N &= -\vec{h}_\Omega \cdot \vec{y}_N\\ q_N &= \vec{h}_\Omega \cdot \vec{x}_N\\ r_N &= \dot{\psi}_N\vec{z}_W \cdot \vec{z}_N\end{aligned} \tag{30}$$

For compactness we have defined

$$\vec{h}_\Omega = \frac{m}{u_{1_{\mathrm{ff}}}}\left(\left(\vec{\xi}_N^{(3)} \cdot \vec{z}_N\right)\vec{z}_N - \vec{\xi}_N^{(3)}\right). \tag{31}$$

The nominal angular acceleration, used in the calculation of the feedforward moment terms, is derived to be

$$\dot{\vec{\Omega}}_{NW} = \alpha_{1_N}\vec{x}_N + \alpha_{2_N}\vec{y}_N + \alpha_{3_N}\vec{z}_N, \tag{32}$$

where the individual components of the nominal angular acceleration are

$$\begin{aligned}\alpha_{1_N} &= -\vec{h}_\alpha \cdot \vec{y}_N\\ \alpha_{2_N} &= \vec{h}_\alpha \cdot \vec{x}_N\\ \alpha_{3_N} &= \left(\ddot{\psi}_N\vec{z}_N - \dot{\psi}_N\vec{h}_\Omega\right) \cdot \vec{z}_W\end{aligned} \tag{33}$$

For compactness we again define a new variable

$$\begin{aligned}\vec{h}_\alpha = -\frac{1}{u_{1_{\mathrm{ff}}}}\Big(&m\vec{\xi}_N^{(4)} + \ddot{u}_{1_{\mathrm{ff}}}\vec{z}_N + 2\dot{u}_{1_{\mathrm{ff}}}\vec{\Omega}_{NW}\\ &\times\vec{z}_N + \vec{\Omega}_{NW} \times \vec{\Omega}_{NW} \times \vec{z}_N\Big).\end{aligned} \tag{34}$$

The derivatives of the net thrust, which appear in Eq. (34), are derived to be

$$\begin{aligned}\dot{u}_{1_{\mathrm{ff}}} &= -m\vec{\xi}_N^{(3)} \cdot \vec{z}_N\\ \ddot{u}_{1_{\mathrm{ff}}} &= -\left(m\vec{\xi}_N^{(4)} + \vec{\Omega}_{NW} \times \vec{\Omega}_{NW} \times \vec{z}_N\right) \cdot \vec{z}_N\end{aligned} \tag{35}$$

Note that the equations presented in this section are taken almost directly from Mellinger and Kumar [15] but are stated here for completeness of our approach.

4.6. Flight controller

In this section we describe how the results of the real-time kinodynamic planner are interpreted by the quadrotor's flight controller. Additionally we present new, previously unpublished work on a "locally reactive" control function that enables more reliable, graceful flight maneuvers in close proximity to obstacles.

The flight controller synthesizes work by Lee et al. [38] with Ge and Cui [32] and is composed of three parts: a feedforward component, a feedback component, and a "locally reactive" component. Feedforward inputs, denoted with subscript "ff", are generated from the differentially flat mapping in Section 4.5 and feedback terms, denoted with subscript "fb", are generated via proportional-derivative (PD) tracking of position, velocity, orientation and angular velocity. The locally reactive terms, denoted with subscript "lr", are loosely based on the concept of potential fields where proximity to obstacles create a virtual force to "push" the quadrotor away (i.e. the quadrotor reacts to obstacles). The reactive terms were originated from Ge and Cui's work, but were modified during experimentation until a desirable behavior was observed. Since these terms were empirically derived, they no longer represent a gradient of a potential field. As previously noted, the locally reactive terms of the controller are not necessary to achieve real-time obstacle avoidance as the planning framework is fast enough to account for dynamic obstacles on its own. During flight tests, however, the locally reactive controller terms significantly improved the performance of the quadrotor by generating more predictable, graceful manuevers. Eq. (36) gives the net thrust control input due to the feedforward, feedback, and locally reactive components.

$$\begin{aligned}u_1 &= u_{1_{\mathrm{ff}}} + u_{1_{\mathrm{fb}}} + u_{1_{\mathrm{lr}}}\\ &= -\vec{z}_B \cdot \left(\vec{F}_{\mathrm{ff}} + \vec{F}_{\mathrm{fb}} + \vec{F}_{\mathrm{lr}}\right)\\ &= -\vec{z}_B \cdot \left(m\ddot{\vec{\xi}}_N - mg\vec{z}_W + K_\xi\vec{e}_\xi + K_v\vec{e}_v + \vec{F}_{\mathrm{lr}}\right)\end{aligned} \tag{36}$$

Eq. (37) presents the control inputs for the moments about the body axes.

$$\begin{aligned}[u_2, u_3, u_4]^T &= [u_2, u_3, u_4]^T_{\mathrm{ff}} + [u_2, u_3, u_4]^T_{\mathrm{fb}}\\ &= J_B\left(R_B^TR_N\dot{\vec{\Omega}}_{BW} - \vec{\Omega}_{BW} \times \left(R_B^TR_N\vec{\Omega}_{BW}\right)\right)\\ &\quad + \vec{\Omega}_{BW} \times J_B\vec{\Omega}_{BW} + K_R\vec{e}_R + K_\Omega\vec{e}_\Omega.\end{aligned} \tag{37}$$

The error terms for feedback control are given by Eq. (38) [38]

$$\begin{aligned}\vec{e}_\xi &= \vec{\xi}_N - \vec{\xi}\\ \vec{e}_v &= \dot{\vec{\xi}}_N - \dot{\vec{\xi}}\\ \vec{e}_R &= \frac{1}{2}\left(R_B^{\mathrm{T}}R_D - R_D^{\mathrm{T}}R_B\right)^\vee\\ \vec{e}_\Omega &= R_B^{\mathrm{T}}R_D\vec{\Omega}_D - \vec{\Omega}_B\end{aligned} \tag{38}$$

where $^\vee$ represents the *vee-map*; the inverse of the *hat-map*. The matrices $K_\xi, K_v, K_R, K_\Omega \in \mathbb{R}^{3\times3}$ are user-defined gain matrices for PD trajectory tracking.

The rotation matrix R_D represents the *desired* orientation to account for feedback and locally reactive terms. This is distinct from the nominal orientation R_N that is independent of feedback and obstacle influence. During perfect trajectory tracking with no nearby obstacles one has $R_N = R_D = R_B$. The rotation matrix R_D is calculated by substituting $\vec{F}_{\mathrm{ff}} + \vec{F}_{\mathrm{fb}} + \vec{F}_{\mathrm{lr}}$ into Eq. (26) and proceeding with Eqs. (27) and (28).

The locally reactive force term, $\vec{F}_{\mathrm{lr}}$, in Eq. (36) is calculated based on obstacle proximity and velocity via

$$\vec{F}_{\mathrm{lr}} = \sum_{i=1}^{n_{\mathrm{obs}}} \frac{1}{\left|\left|\vec{r}_i\right|\right|^2}\left(-\eta_r\hat{n}_i + \eta_{va}v_{n_i}\hat{n}_i - \eta_{vp}v_{n_i}\left(v_{n_i}\hat{n}_i - \vec{v}_i\right)\right), \tag{39}$$

where n_{obs} is the number of obstacles, $\vec{r}_i$ is the position of the closest point on the ith obstacle with respect to the quadrotor body frame, $\hat{n}_i$ is the unit vector in the $\vec{r}_i$ direction, $\vec{v}_i$ is the relative velocity of the ith obstacle with respect to the quadrotor, and $v_{n_i} = \vec{v}_i \cdot \hat{n}_i$. The first two terms in Eq. (39) represents a repulsive force due to obstacle relative position and velocity, respectively. The third term is a steering term due to obstacle relative velocity. The variables η_r, η_{va}, and η_{vp} are weighting factors for position, aligned velocity, and perpendicular velocity, respectively. For obstacles outside of a user-defined influence region, the locally reactive force

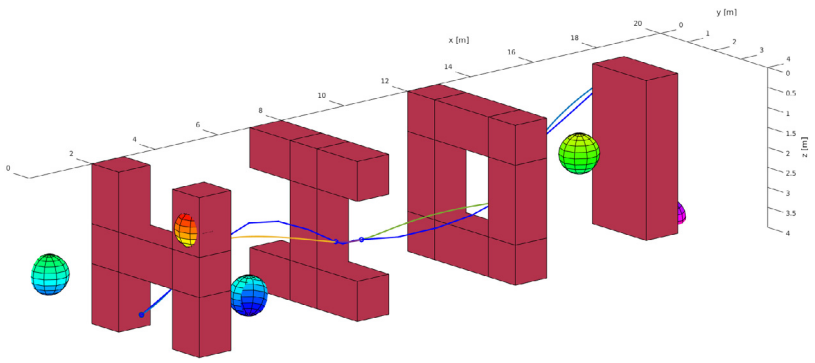

Fig. 6. An instance of one of the simulated flight tests with a 3D maze of wall structures and randomly placed spherical obstacles.

in Eq. (39) is set to zero. Furthermore, if $v_{n_i} < 0$ then the velocity terms of Eq. (39) are set to zero.

It should be again noted that this locally reactive control is non-essential for addressing the motivating questions in Section 1; however it improves performance during physical demonstrations by minimizing the number of replanning events necessary by avoiding occlusion of the existing motion plan. Distinguishing the locally reactive control as non-essential is important because it does require additional obstacle information that is not required by the rest of the framework; that is position and velocity data for each obstacle as opposed to just collision detection. Fortunately, the formulation of the locally reactive control layer only utilizes obstacle information in the workspace, not in the configuration space, therefore requiring very little computational overhead. However, if we want to be more strict with our assumptions of obstacle data, we could eliminate the locally reactive control without sacrificing the real-time planner as a whole.

5. Experimental validation

We validate the proposed real-time framework in both simulation and physical experiments. In this section we describe the results of these test campaigns.

5.1. Simulated results

While physical demonstrations are the ultimate test of the framework's effectiveness, limited laboratory space constrains the number and complexity of obstacles sets that can be tested. Therefore a simulation with a maze of obstacles is devised to validate our approach in more complex environments. The simulated environment consists of a corridor with dimensions 20 m × 4 m × 4 m with the start and goal states randomly generated from opposing ends of the corridor. Cuboid obstacles are arranged to create a 3-dimensional "maze". A fixed number of spherical obstacles with radii of one meter are placed at random to ensure that we have not inadvertently tailored our algorithm to this specific cuboid-maze obstacle set.[11] Fig. 6 gives an instance of this obstacle configuration and associated solution. The start state is shown on the left side of the image and the goal state is obscured by the final obstacle on the right. The initial motion plan, as returned by *kino*-FMT* (see Section 3.1), is indicated in blue and the smoothed, dynamically feasible trajectory (see Section 4.4) is indicated in multicolor.

To assess the performance of the real-time motion planner, we consider the following metrics: (1) online computation time, (2) solution cost, (3) failure rate, and (4) classification accuracy of reachable sets by the machine learning algorithm. In Section 5.1–5.4 we explain the experimental setup and discuss the performance in terms of computation time, solution cost, and failure rate; saving discussion of the machine learning performance for Section 5.5.

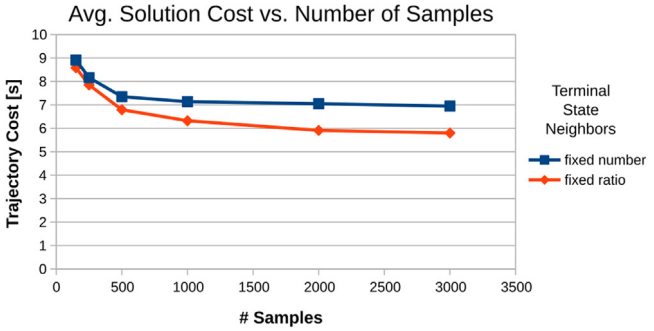

Fig. 7. Average trajectory cost as a function of number of state samples and number of terminal state neighbors for a fixed obstacle coverage.

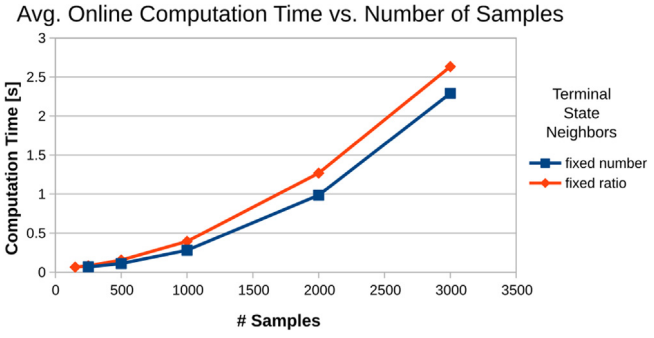

Fig. 8. Average online computation time as a function of number of state samples and number of terminal state neighbors for a fixed obstacle coverage.

Through the simulated test campaign it was determined that the performance metrics were most dependent upon two *design variables*: number of sampled states and number of terminal state neighbor connections; and a third, situation-dependent variable: obstacle coverage. The simulated test campaign involved selecting values for the design variables and executing 100 trials for each combination. For each trial the start state, goal state, and spherical obstacle placement were randomized. We summarize the trade-offs between design variables and performance metrics in Table 1.

Figs. 7 and 8 illustrate the trends of performance metrics as functions of the design variables. As expected, with increasing

[11] Note that the framework developed in Sections 3 and 4 is in no way restricted to cuboid and spherical obstacles. For implementation, however, we choose relatively simplistic obstacles because the development of sophisticated collision checking routines is outside the scope of this paper.

Table 1
Trajectory cost and computation time breakdown for the Real-Time Kinodynamic Framework for a range of design variables.

	Design variables		Performance metrics	
	# samples	# terminal state neighbors	Avg. trajectory cost [s]	Avg. computation time [s]
Fixed numbers	150	10	8.91	0.060
	250	10	8.16	0.067
	500	10	7.35	0.110
	1000	10	7.14	0.280
	2000	10	7.05	0.985
	3000	10	6.95	2.289
Fixed Ratios	150	15	8.58	0.065
	250	25	7.85	0.081
	500	50	6.79	0.154
	1000	100	6.33	0.394
	2000	200	5.91	1.268
	3000	300	5.80	2.632

number of samples N_s, the solution cost, $\mathcal{J}$, decreases while computation time increases. Based on Fig. 7, we can see that $N_s = 500$ is an acceptable sample density for this obstacle configuration as there is marginal decrease in solution cost for higher sample numbers. As shown in Table 1, 500 samples corresponds to an average solution time of 0.110 s for a fixed number of terminal state neighbors and 0.154 s for a fixed ratio of terminal state neighbors. It is argued that these computation times represent "real-time" planning; we verify this claim with physical demonstrations in Section 5.3 and compare to computation times in the existing literature in Section 5.4.

The effect of neighborhood sizes for the terminal states (i.e., the number of states in the pre-sampled set V for which connections are made to the start and goal states) is also presented in Figs. 7 and 8. For the fixed number of terminal state neighbors, the start and goal states are connected to the precomputed roadmap at the 10 closest states in the set of pre-sampled states, V; where closeness is approximated by the machine learning algorithm. For the fixed ratio of terminal state neighbors, the start and goal states are connected to the closest 10 *percent* of the set V. Each connection between the terminal states and the precomputed roadmap constitutes an online OBVP solution; therefore the fixed number corresponds to $\mathcal{O}(1)$ online OBVPs, where the fixed ratio corresponds to $\mathcal{O}(N_s)$ online OBVPs.

This comparison of number of terminal state neighbors is made to determine the effect of restricting the online OBVP solutions to constant order, which is argued to be an enabling technique for real-time kinodynamic planning. From Fig. 8 we see that, indeed, restriction of terminal state neighbors leads to a reduction in online computation time of up to 15%. While 15% is not a staggering difference in computation time, it is important to note that this is only representative of the quadrotor system where much work has been done to minimize the computation time for OBVPs (see Section 4.3). For more general systems where OBVPs may be very computationally expensive, this restriction to $\mathcal{O}(1)$ online OBVPs may reduce online planning times by several orders of magnitude [9]. The decrease in computation time, however, comes at the expense of increased solution cost as indicated by Fig. 7.

Another insightful question is, *for a given obstacle coverage, what is the appropriate number of samples?*. As indicated by Fig. 8, it is desirable to use the minimum number of samples, N_s, while still achieving acceptable solution cost, as this requires the minimum computation. Since the data given in Table 1 only represents a single obstacle coverage, a second test campaign was run to determine the necessary sample count as a function of obstacle coverage. Figs. 9, 10, and 11 summarize the data from this second test campaign. Note that we referred to *approximate* obstacle coverage which is measured as the ratio of obstacle volume to unobstructed workspace volume. This value may be greater than one because

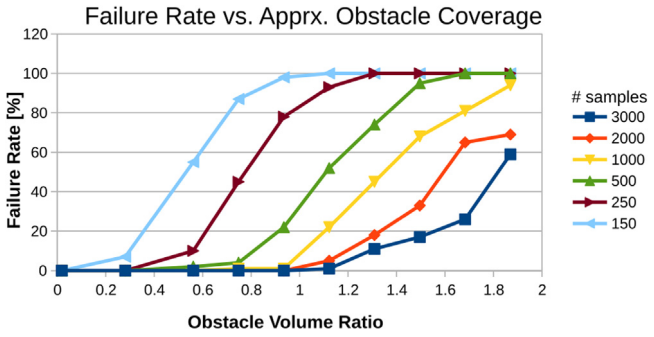

Fig. 9. Rate of failure to find solution as a function of approximate obstacle coverage for a range of sample sizes.

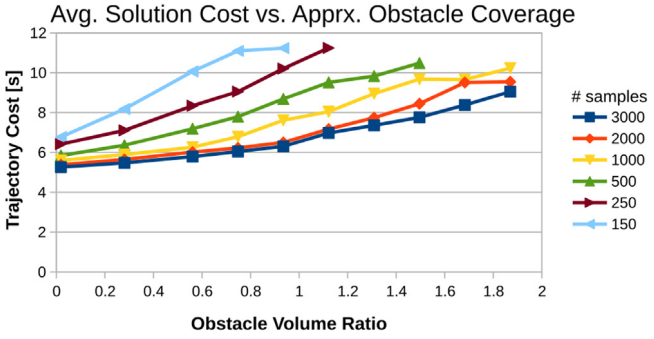

Fig. 10. Average solution trajectory cost as a function of approximate obstacle coverage for a range of sample sizes.

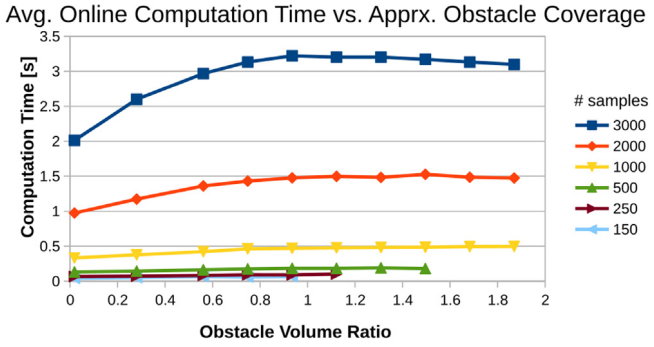

Fig. 11. Average computation time as a function of approximate obstacle coverage for a range of sample sizes.

obstacles were placed at random and overlapping volumes were double counted.

We immediately see several trends in the data that we expect. First, Fig. 9 shows that as the obstacle coverage increases, we must use larger sample numbers to prevent planner failure. Lower sample numbers, $N_s < 500$, quickly rise to 100% failure with increasing obstacle coverage. Note that for sample counts of $N_s = \{1000, 2000, 3000\}$, the curves diverge from 0% failure at roughly the same obstacle volume ratio. Therefore, $N_s = 1000$ is a favorable sample count because higher sample counts give no better guarantees for 0% failure at higher obstacle coverage.

Fig. 10 gives average cost of a solution trajectory as a function of obstacle coverage and sample count. We see two expected trends: solution cost increases with increasing obstacle coverage and decreases with increasing samples count (as was indicated in Fig. 7). We also see that there is marginal improvement in solution cost beyond a sample count of 2000.

Fig. 11 gives the online computation time as a function of sample count and obstacle coverage. Again we see the expected trends that computation time increases with increasing obstacle count and with increasing sample count. In more detail, we see that computation time roughly doubles for each tier of sample counts. This leads to large increases in computation time for sample counts greater than 1000. Based on this observation, plus the observation that there is marginal improvement in solution cost beyond $N_s = 2000$, plus the observation that $N_s = \{1000, 2000, 3000\}$ all diverge from 0% failure at the same obstacle coverage, we assert that $N_s = 1000$ is the best suited sample count for our physical experiments. Fig. 11 shows that computation times for $N_s = 1000$ are less than 0.5 s, even in the worst case. One unexpected feature of Fig. 11 that should be explained is the slight decrease in computation time with increase in obstacle volume ratio beyond 1.0 for the 3000 sample set of trials. This effect is due to cases where the planner fails to find a solution (see Fig. 9). After we reach an obstacle coverage threshold where the planner starts to fail to find a solution, adding more obstacle coverage can cause the planner to fail even earlier in the planning process, thus slightly reducing the average computation time.

The simulation test campaign verifies that real-time planning framework achieves computation times of well below 1 s – typically on the order of 0.1 s with 0.5 s as a worst case – for a wide range of obstacle coverage. The test campaign also gives us the empirically derived value of $N_s = 1000$ as an acceptable sample count for the indoor environments and obstacle configurations considered in this paper. Now we test the effectiveness of the framework on a physical system navigating dynamic obstacles.

5.2. Experimental flight setup

The real-time framework is demonstrated on a Pixhawk autopilot flown on a DJI F-450 and F-330 frame. Positioning information is provided by a Vicon motion tracker with data streamed to the quadrotor via a Wifly RN-XV module. Currently the motion planning and path smoothing computations are run in MATLAB/C++ on a single-threaded Intel Core i7-4790K CPU. The final trajectory is transmitted to the Pixhawk for low-level flight control. This communication structure is represented in Fig. 12. Table 2 gives detailed information on the computational platform and programming language for each of the major components of the framework discussed in Sections 3 and 4. Future work will convert all portions of the online phase (see Algorithm 2) to C++ to be run on an embedded processor on the quadrotor.

The quadrotor is navigating an indoor environment with dimensions of approximately 3 m × 4 m × 3 m. The framework was tested on a range of obstacle sets, two of which are discussed in detail in Section 5.3.

Table 2
Computational platform and programming language for the major components of the real-time framework.

Process	Reference	Processor	Language
localization	NA	workstation	C++
precomputations	3 & 4	Workstation	MATLAB
neighborhood estimation	3.2	Workstation	MATLAB
OBVP solutions	4.3	Workstation	MATLAB
sampling-based planning	3.1	Workstation	C++
min-snap smoothing	4.4	Workstation	MATLAB
flat-to-nonlinear mapping	4.5	Pixhawk	C/C++
flight control	4.6	Pixhawk	C/C++

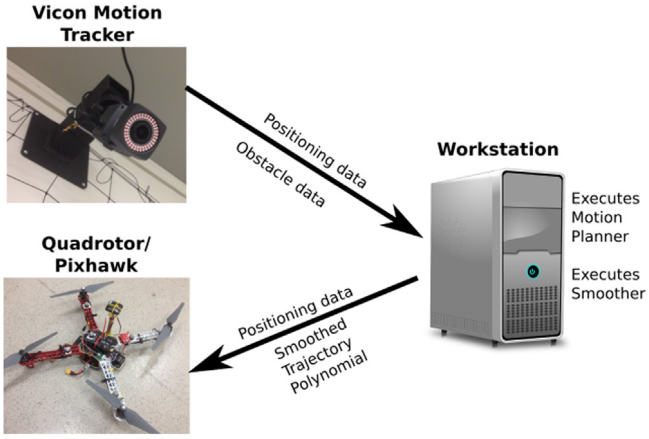

Fig. 12. Communication/computation structure for flight tests.

5.3. Experimental flight results

The real-time kinodynamic planner was successfully demonstrated in a campaign of flight tests. The test campaign was executed in the indoor environment of the Autonomous Systems Laboratory at Stanford University. During the campaign the quadrotor utilized the real-time planner to navigate a set of static and dynamic obstacles. Netting was hung from the ceiling to create maze-like environments while a human-subject would swing objects to create dynamic obstacles.

Fig. 13 gives a timelapse of the most basic flight test performed: The quadrotor navigating a set of parallel walls with no dynamic obstacles present. The walls are arranged to create a z-shaped corridor 1.5 m in width. This test acted as the first validating experiment for the real-time kinodynamic planner. The solution used 500 sampled states and required 0.313 s of online computation time. This test also demonstrated the agility of the quadrotor platform as indicated by the banked turns as it rounded the corners. Building from this initial experiment, originally presented in Allen and Pavone [10], the planning code was optimized to further reduce computation times and dynamic obstacles were introduced.

In the first physical experiments with dynamic obstacles, shown in a sequence of images in Fig. 14, the autonomous quadrotor is presented with a single obstacle in between its start state and its goal region. This creates two doors, each roughly one meter in width, from which the quadrotor can "choose" to navigate. When the real-time planner completes, the quadrotor begins executing the trajectory through the nearest of the two doors. Upon nearing the door, a human subject enters and presents a dynamic obstacle; in this demonstration the obstacle is the point of a fencing blade which is numerically expanded to a sphere with radius equal to the arm length of the quadrotor. The human subject continues to approach the quadrotor, causing it to be "pushed back" due to the reactive controller (see Section 4.6), until the existing trajectory is completely obstructed and replanning is initiated. Fig. 14 shows

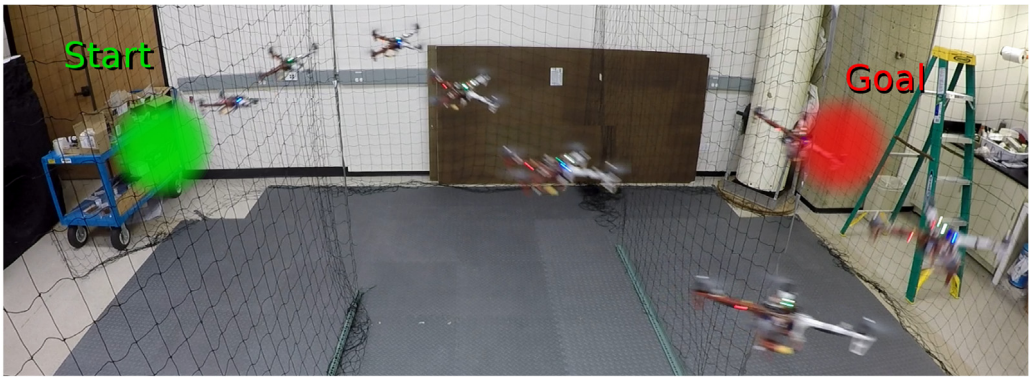

Fig. 13. Timelapse of quadrotor navigating static obstacles.

the moment of replanning along with the solution provided by the real-time planning framework. Due to the proximity to the initially chosen door, the quadrotor executes three planning cycles that attempt to navigate the initial door and the dynamic obstacle. Since the dynamic obstacle is acting adversarially, always obstructing the chosen trajectory, this continues until the quadrotor is forced to a point where the second door becomes the optimal solution and the quadrotor navigates to the goal region without further obstruction from the human subject.

Fig. 15 gives a second scenario where the autonomous quadrotor is tasked with navigating a parallel-walled maze with a corridor of roughly 1.5 m in width. Again, a human subject introduces a set of dynamic obstacles to force online recomputation of the motion plan. During this demonstration the dynamic obstacles are presented immediately before the quadrotor rounds a corner, causing an abrupt replanning cycle that navigates over and between the dynamic obstacles and the wall. Because the human subject does not continue to act adversarially in this case, only a single recomputation of the motion plan is necessary. Figs. 14 and 15 show that the online computation times for planning are on the order of 1/4 of a second.

Video of some of the flight experiments can be found here: https://www.youtube.com/watch?v=fv_6KB2eEFc

5.4. Discussion

A primary goal of this work – the goal that is implied by Motivating Question 2 in Section 1 – is to prove that the entire planning framework can be executed in real-time on a physical robot. As shown in Figs. 14 and 15, we achieve online computation times between 0.20 s and 0.33 s for 1000 sampled nodes. These computation times are in good agreement with those presented in Section 5.1 which ranged from 0.11 s to 0.5 s for comparable sample counts. The difference in computation times between simulated and physical experiments are due to differing obstacle sets (simulated tests being more densely obstructed) and additional computational tasks for physical experiments (e.g. communication of solution trajectories). To better understand the breakdown of computation time during physical experiments, Table 3 gives the percentage of computation time for separate tasks in the planner for a range of sample counts.

Referring to Table 3, the computation time is broken down into percentages for the major components of the framework: neighborhood classification for the terminal states (see Section 3.2); neighborhood OBVP solutions for the terminal states (see Section 4.3); sampling-based motion planning (see Section 3.1); path smoothing to generate a minimum-snap, dynamically feasible trajectory (see Section 4.4); and communication (see Section 5.2). We see that the majority of the computation time is consumed by the solution of optimal boundary value problems between the terminal states, x_{init} and the samples in $\mathcal{X}_{\text{goal}}$, and their estimated neighborhoods. This result exemplifies the motivation to minimize the number of online OBVPs to be solved. For the double integrator model of the quadrotor, the average OBVP solution time is 0.0235 s per OBVP solution. In comparison, the average `NearSVM` classification time is 1.95×10^{-5} s per classification; roughly 1200 times, or three orders of magnitude, faster than OBVP solution. This rapid approximation of neighborhood sets – in contrast to the explicit neighborhood determination via OBVP solutions – is the critically enabling component for real-time implementation.

To compare the computation times we achieved to those presented in the existing literature, Webb and van den Berg simulate an almost identical problem; however they do not perform any path smoothing or communication to a physical quadrotor [22]. With 1000 sampled states Webb and van den Berg's solution takes 51.603 s; *i.e.* ∼150x, or 2 orders of magnitude, slower than the technique presented here. Richter et al. do not state the computation time for motion planning demonstrated in their work [13]. They do, however, give the computation time for a simplified, 2-dimensional problem that incorporates geometric path planning and minimum-snap path smoothing. The simplified, 2D planning problem in Richter et al. takes 3 s of computation time; *i.e.* ∼9.1x, or 1 order of magnitude, slower than the slowest physical experiment computation time presented here. Therefore the real-time kinodynamic framework demonstrates a significant reduction in computation time when compared to existing techniques.

Frazzoli et al. boasts the most impressive computation times with sub-second execution for the similar, but not identical, helicopter system navigating static spherical objects [5]. Computation times for dynamic obstacles rise to 10 s of seconds for a parallel wall obstacle set. Therefore we again see our method produce roughly 2 orders of magnitude improvement in computation time when compared to existing techniques for dynamic obstacles. There are important caveats to mention when trying to make such a direct comparison with the results of Frazzoli et al. First of all Frazzoli et al. was written in 2002 and therefore implemented on less advanced computational platforms. Secondly, Frazzoli et al. only tested simulated systems and did not have to account for the additional computational burdens of implementation on a physical system; i.e. path smoothing and communication. Furthermore the approach in Frazzoli et al. only sought feasible trajectories, not necessarily optimal ones. The work employs only a small set of motion primitives – avoiding the solution to online OBVPs all together – to achieve path planning. Restricting trajectories to a small set of predefined maneuvers limits the technique's ability to handle novel, complex, or even pathological obstacle environments.

We note here that our physical demonstrations were not infallible; roughly 50% of experiments ended in some form of a crash. These failures were found to be due to two factors: poor

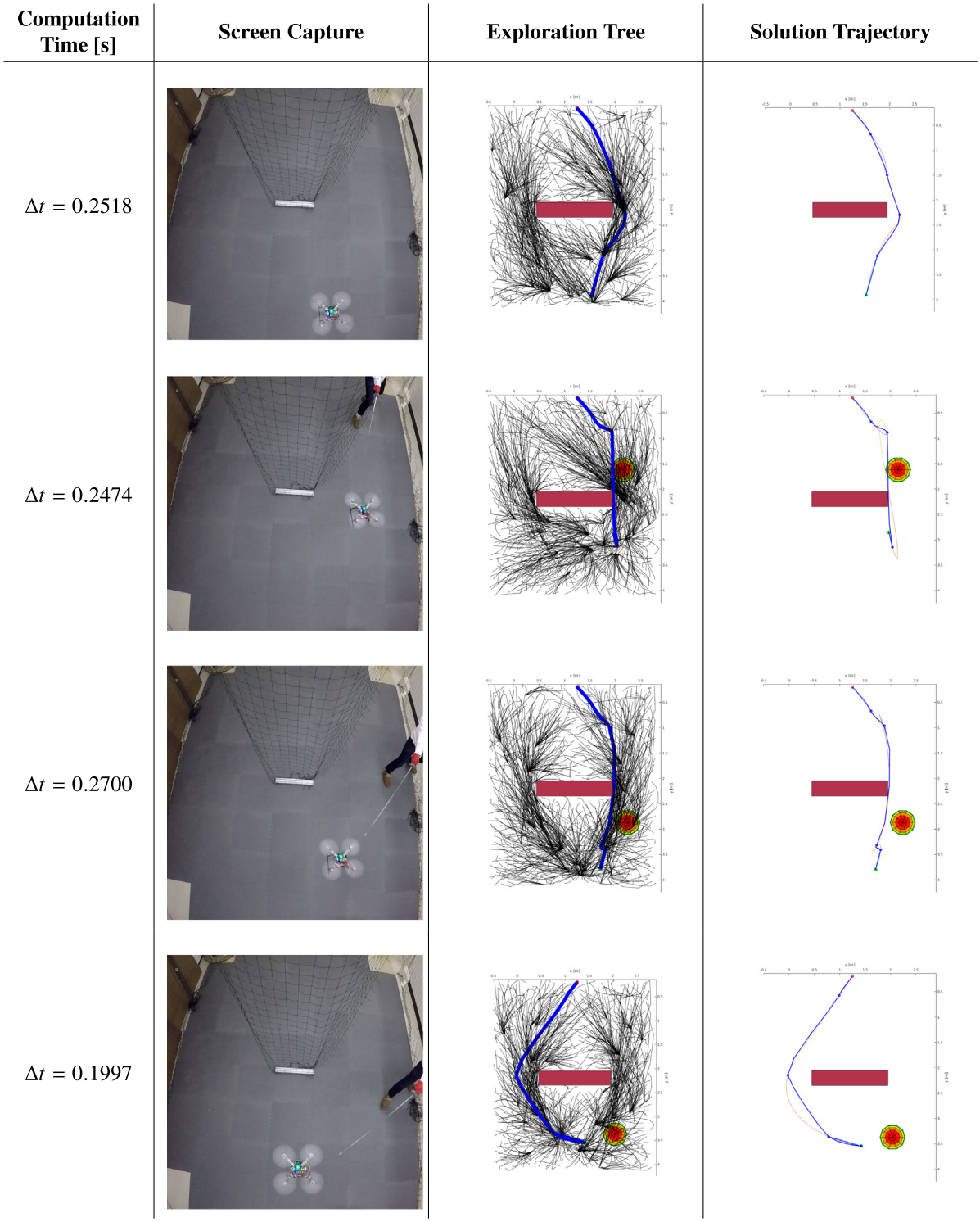

Fig. 14. Time sequence of real-time planning in "two door" environment with static and dynamic obstacles. Column 1 gives the online computation time for the planning event. Column 2 gives screen capture of the moment of replanning. Column 3 gives the tree explored during replanning with the preliminary solution in blue. Column 4 gives the preliminary planning solution and the smoothed trajectory. Obstacles are represented by red rectangles or spheres.

Table 3
Computation time breakdown for the Real-Time Kinodynamic Framework for differing numbers of sampled states.

# of Samples	Neighbor Classifier [%]	Neighbor OBVPs [%]	Kino-FMT* [%]	Smoothing [%]	Comms [%]
500	6.42	37.73	9.43	30.43	25.29
1000	5.31	41.01	13.60	32.70	4.24
2000	4.56	41.65	19.81	26.40	3.38
3000	3.08	53.69	29.74	9.65	1.63

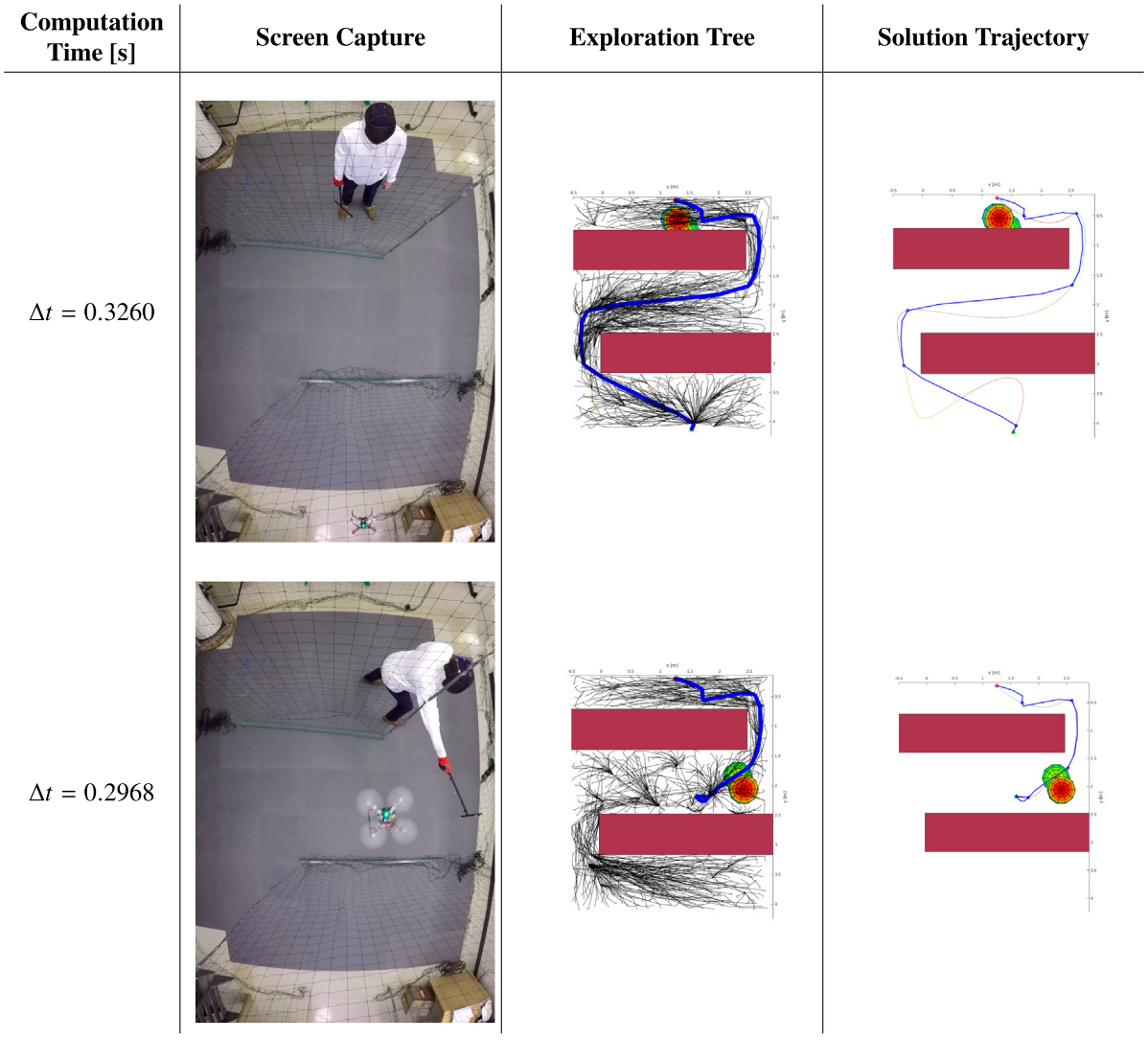

Fig. 15. Time sequence of real-time planning in "maze" environment with static and dynamic obstacles. Column 1 gives the online computation time for the planning event. Column 2 gives screen capture of the moment of replanning. Column 3 gives the tree explored during replanning with the preliminary solution in blue. Column 4 gives the preliminary planning solution and the smoothed trajectory. Obstacles are represented by red rectangles or spheres.

system identification of the quadrotor leading to inaccurate dynamic parameters in the flight controller (see Section 4.6), and loss of positioning data due to exiting Vicon coverage. Both of these failure modes were outside the scope of this work which was solely focused on developing the real-time planning framework. These failure modes, do however, motivate future work: advanced system identification for improved controller performance and robust, onboard estimation/localization to eliminate reliance on a motion capture system.

5.5. Results for machine learning of reachable sets

Due to the reliance on machine-learning of neighbor sets, it is important to determine the classification accuracy of the `NearSVM` algorithm. In this work we apply the `NearSVM` algorithm (see Section 3.2) to the control-penalized double integrator system presented in Section 4.3. The state space of the double integrator system in Eq. (8) is 6-dimensional. The two boundary values for an OBVP are concatenated into a 12-dimensional attribute vector, given as $\mathbf{p}$ in Eq. (4). The feature vector is a 33-element vector, given in Table 4, composed of nonlinear mappings of elements from the attribute vector. A third order kernel function is chosen; therefore $p = 3$ in Eq. (5). For training and testing of the SVM classifier, 50000 OBVPs are solved from randomly selected pairs of sampled states during the offline computation phase. A neighbor radius, or cost threshold, is chosen as the 10th quantile of all OBVP costs; which for this test campaign evaluated to neighbor cost threshold of roughly 0.69 s. In other words, for a given state, roughly 10% of all other states are within 0.69 s as measured by a minimum-time optimal control problem. To train the SVM classifier, $N_{\text{train}} = 20000$ of the 50000 OBVP solutions were used with the 0.69 s cost threshold. On average less than one training error occurred per the 20000 training examples.

Table 4
Feature vector for neighbor determination of the double integrator quadrotor model.

x_1	x_2	$\lvert\Delta x\rvert$	$(\Delta x)^2$	$(\Delta x)^3$	$\sqrt{(\Delta x)^2+(\Delta y)^2+(\Delta z)^2}$
y_1	y_2	$\lvert\Delta y\rvert$	$(\Delta y)^2$	$(\Delta y)^3$	$\sqrt{(\Delta \dot{x})^2+(\Delta \dot{y})^2+(\Delta \dot{z})^2}$
z_1	z_2	$\lvert\Delta z\rvert$	$(\Delta z)^2$	$(\Delta z)^3$	$\sqrt{(\Delta x)^2+(\Delta y)^2+(\Delta z)^2+(\Delta \dot{x})^2+(\Delta \dot{y})^2+(\Delta \dot{z})^2}$
$\dot{x}_1$	$\dot{x}_2$	$\lvert\Delta \dot{x}\rvert$	$(\Delta \dot{x})^2$	$(\Delta \dot{x})^3$	
$\dot{y}_1$	$\dot{y}_2$	$\lvert\Delta \dot{y}\rvert$	$(\Delta \dot{y})^2$	$(\Delta \dot{x})^3$	
$\dot{z}_1$	$\dot{z}_2$	$\lvert\Delta \dot{z}\rvert$	$(\Delta \dot{z})^2$	$(\Delta \dot{x})^3$	

Table 5
Training and testing accuracy of machine-learning-based neighborhood classification algorithm.

# training examples	Avg. # training errors	# testing examples	Avg. # true positives	Avg. # true negatives	Avg. # false positives	Avg. # false negatives	Avg. testing error [%]
20 000	0.6	30 000	2693	26 600.6	371.8	334.6	2.35

The algorithm was tested against 30000 additional OBVP examples to ensure that the SVM was not *over-trained* to the training set.[12] The average testing error was under 3%, well within the acceptable tolerance for the purpose of this work and a marked improvement over the author's prior work on machine learning of cost-limited reachable sets [9]. Table 5 gives the training and testing results. A 'positive' indicates that `NearSVM` classified the OBVP example as within the cost threshold, and a 'negative' indicates a classification of the OBVP outside of the cost threshold. The number of true positives is roughly 10% of the number of true negatives; as expected with the 10th quantile cost threshold. The average number of false positives and false negatives are approximately equal indicating that the classifier is not biased toward one classification.[13]

The information given in Table 5 only tells us the rate of neighborhood classification error, it does not tell us where in the state space these misclassifications occur. To form a deeper understanding of where/why misclassification of neighbors occur, we illustrate the `NearSVM` results with a simplified case. Fig. 16 presents a set of trials for the neighborhood classifier when compressed to two spatial dimensions and one velocity dimension. Each trial, represented by its own image in Fig. 16, attempts to classify the reachable neighborhood of an initial state. For each trial, the initial state is the origin with a y-velocity ranging from 0 m/s to 14 m/s. The final states for each classification are spread across the xy-axes and all have a final velocity of zero. The neighborhood cost threshold is 2.178 s which corresponds to the 10th quantile of costs for this set of training data.

The reachability of each final state is assessed by solving an OBVP, as discussed in Section 4.3, between the initial state and each final state. Reachable states are indicated in blue circles and non-reachable are indicated in black triangles. The machine learning algorithm, `NearSVM`, is then applied to estimate reachability. Misclassification of reachability, whether it be a false-positive or false-negative, is indicated by a red star.

As expected, Fig. 16 shows that the true reachable set shifts further along the y-axis as the initial velocity in the y-direction increases. We also see that misclassification of reachability always occurs on the boundary between the reachable and non-reachable sets. This is a desirable result for a well-trained algorithm. A poorly-trained algorithm would make classification errors well within the reachable or non-reachable sets. If a poorly trained neighborhood classifier were used in the real-time planning framework, we would end up solving OBVPs for states that our well outside of neighborhood – likely leading to collisions with obstacles, increasing computation time – or fail to recognize nearby states, thus increasing solution cost of our trajectory.

It is important to note that an analytical solution exists for the reachable set of the control penalized double integrator [22]. The question is then, why would you use the machine learning approximation for reachable sets when an analytical solution exists? This question lies at the balance point between the two motivating questions presented in Section 1. While we want to demonstrate real-time planning for a quadrotor, we also want to validate a planning framework that is applicable to a more general set of dynamical systems. Since an arbitrary dynamical system cannot be expected to have an analytical solution for reachable sets, we maintain the use of the machine learning approach in effort to validate it in physical experiments.

6. Conclusions

This work was motivated by two related, yet distinct, questions: 1. can we produce an algorithmic framework for solving a general kinodynamic planning problem in real-time; and 2. can such a framework be implemented on a real-world system – such as a quadrotor – and proven to be effective at navigating an obstructed environment with dynamic obstacles. These questions were addressed by the development of a full-stack planning architecture that is broad enough to encompasses a wide range of kinodynamic systems yet flexible enough to be tailored to a specific system, i.e. the quadrotor, in order to maximize performance. This work extends the authors' prior work by introducing an event-based replanning structure and a locally-reactive control layer in order to navigate dynamic obstacles including a human adversary who actively attempts to obstruct the path of the UAV. For the relevant, real-world problem of quadrotor obstacle avoidance, this framework is shown to reduce online computation times to below one second; several orders of magnitude faster than techniques presented in existing literature. This is arguably one of the first – if not the first – demonstration of truly real-time kinodynamic planning on a quadrotor system navigating an obstructed, dynamic environment. The drastic improvement in online computation time is achieved by reducing the number of online optimal boundary value problems to be solved to constant order. The reduction to constant order OBVP solutions is enabled by machine learning estimates of reachability sets for a dynamical system.

Further work will validate and extend these results. All components of the planning framework will be translated to C/C++ and run on an embedded processor flown on the quadrotor. While the processing power on an embedded system will be diminished when compared to the workstation used in this paper, the translation from MATLAB to C/C++ is expected to roughly balance the effect; therefore computation times are not expected to change significantly. The localization and mapping that is currently achieved with the Vicon motion capture system will be integrated into the quadrotor system using a range of visual, laser, and ultrasonic sensors. In this way, real-time localization and planning will be achieved on a fully self-contained platform. Enhanced system identification will be performed on the quadrotor system, improving flight control and allowing for even more agile interactions with obstacles.

In the work presented here, we have assumed perfect knowledge of the quadrotor's and obstacles' states and dynamics. This simplifying assumption will be removed in future work that will involve planning under uncertainty. To handle the additional computational complexity of planning under uncertainty, a parallelized planning algorithm will be executed on an onboard GPU.

Beyond quadrotor application, we hope to demonstrate the real-time kinodynamic planning framework on a wide range of dynamical systems such as spacecraft, cars, marine and submarine vehicles, cranes, etc. Furthermore, this work can be extended from

[12] Typically the training set would be much larger than the testing set, but due to convergence issues while training, the training set was reduced and the remainder of OBVP examples was dedicated to the testing set.

[13] For example, we could use a trivial classifier that only predicted negatives and it would return a testing error of 10% because only 10% of cases are positive. This would actually constitute an acceptable rate of classification error if it were not for the fact that all errors would be false negatives as the classifier is trivial. Therefore a well trained classifier should not be biased toward one type of error.

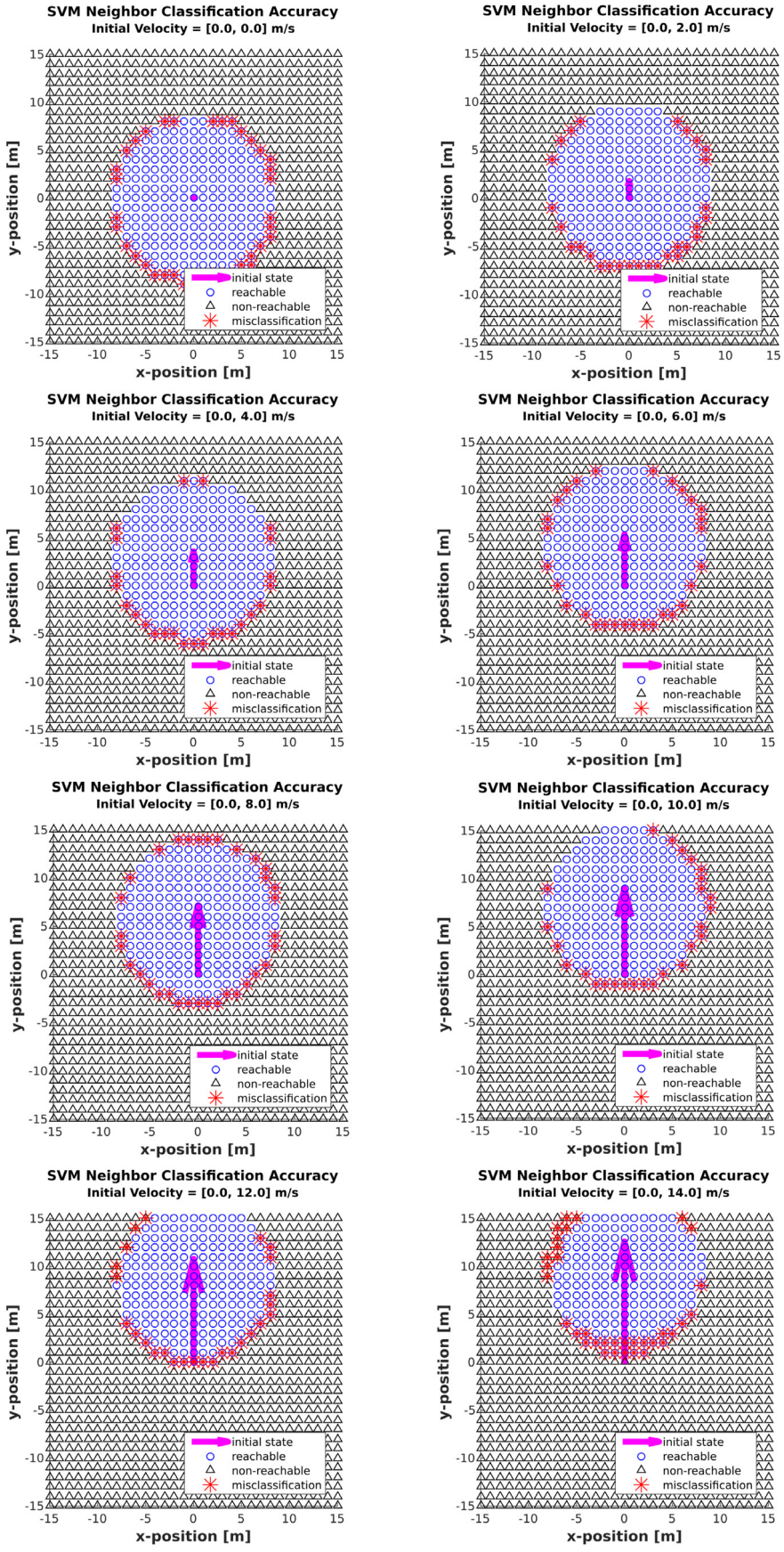

Fig. 16. Simplified, 2-dimensional reachable set classification. For all cases, the start state is at the origin with an initial velocity in the y-direction. The final states all have a velocity of zero.

single agent to multi-agent systems. Of particular interest is the possibility of emergent behavior due to many agents selfishly solving their own planning problem while avoiding collisions with other agents. Finally, game theory could be applied so that robotic agents can model dynamic obstacles as adversaries for more sophisticated avoidance maneuvers.

The code base for this work can be found at:
https://github.com/StanfordASL/KinoFMT.git

Videos of the demonstrations discussed in this paper can be found at:
https://www.youtube.com/watch?v=fv_6KB2eEFc

Funding Sources

This work was partially funded by United Technologies Research Center through the UTRC-2 Graduate Fellowship in Aerospace Systems (2012–2016).

References

[1] G.M. Hoffmann, H. Huang, S.L. Waslander, C.J. Tomlin, Quadrotor helicopter flight dynamics and control: Theory and experiment, in: Proc. of the AIAA Guidance, Navigation, and Control Conference, Vol. 2, 2007.
[2] S. Bouabdallah, A. Noth, R. Siegwart, Pid vs LQ control techniques applied to an indoor micro quadrotor, in: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, Vol. 3, IEEE, 2004, pp. 2451–2456.
[3] M. Hehn, R. D'Andrea, A flying inverted pendulum, in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 763–770.
[4] S.M. LaValle, Motion planning: Wild frontiers, IEEE Robot. Autom. Mag. 18 (2) (2011) 108–118.
[5] E. Frazzoli, M.A. Dahleh, E. Feron, Real-time motion planning for agile autonomous vehicles, AIAA J. Guidance Control Dyn. 25 (1) (2002) 116–129.
[6] P. Leven, S. Hutchinson, A framework for real-time path planning in changing environments, Int. J. Robot. Res. 21 (12) (2002) 999–1030.
[7] S. Singh, A. Majumdar, J.J.E. Slotine, M. Pavone, Robust online motion planning via contraction theory and convex optimization, 2017, Extended Version, Available at http://asl.stanford.edu/wp-content/papercite-data/pdf/Singh.Majumdar.Slotine.Pavone.ICRA17.pdf.
[8] R. Tedrake, I.R. Manchester, M. Tobenkin, J.W. Roberts, LQR-trees: feedback motion planning via sums-of-squares verification, Int. J. Robot. Res. 29 (8) (2010) 1038–1052.
[9] R. Allen, A. Clark, J. Starek, M. Pavone, A machine learning approach for real-time reachability analysis, in: IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, Chicago, IL, 2014, pp. 2202–2208.
[10] R. Allen, M. Pavone, Toward a real-time framework for solving the kinodynamic motion planning problem, in: Proc. IEEE Conf. on Robotics and Automation, Seattle, WA, 2015, pp. 928–934.
[11] R. Allen, M. Pavone, A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance, in: AIAA Conf. on Guidance, Navigation and Control, San Diego, CA, 2016, pp. 1–18.
[12] S.D. Pendleton, W. Liu, H. Andersen, Y.H. Eng, E. Frazzoli, D. Rus, M.H. Ang, Numerical approach to reachability-guided sampling-based motion planning under differential constraints, IEEE Robot. Autom. Lett. 2 (3) (2017) 1232–1239.
[13] C. Richter, A. Bry, N. Roy, Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments, in: International Symposium on Robotics Research, 2013.
[14] C. Richter, A. Bry, N. Roy, Polynomial trajectory planning for quadrotor flight, in: International Conference on Robotics and Automation, 2013.
[15] D. Mellinger, V. Kumar, Minimum snap trajectory generation and control for quadrotors, in: Proc. IEEE Conf. on Robotics and Automation, 2011, pp. 2520–2525.
[16] S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Robot. Res. 30 (7) (2011) 846–894.
[17] K. Sreenath, T. Lee, V. Kumar, Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load, in: Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on, IEEE, 2013, pp. 2269–2274.
[18] D. Mellinger, N. Michael, V. Kumar, Trajectory generation and control for precise aggressive maneuvers with quadrotors, Int. J. Robot. Res. 31 (5) (2012) 664–674.
[19] I.D. Cowling, O.A. Yakimenko, J.F. Whidborne, A.K. Cooke, Direct method based control system for an autonomous quadrotor, J. Intell. Robot. Syst. 60 (2) (2010) 285–316.
[20] I.D. Cowling, O.A. Yakimenko, J.F. Whidborne, A.K. Cooke, A prototype of an autonomous controller for a quadrotor UAV, in: Control Conference (ECC), 2007 European, IEEE, 2007, pp. 4001–4008.
[21] Y. Bouktir, M. Haddad, T. Chettibi, Trajectory planning for a quadrotor helicopter, in: Control and Automation, 2008 16th Mediterranean Conference on, IEEE, 2008, pp. 1258–1263.
[22] D.J. Webb, J. van den Berg, Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints, in: Proc. IEEE Conf. on Robotics and Automation, 2013, pp. 5054–5061.
[23] B. Landry, Planning and Control for Quadrotor Flight Through Cluttered Environments (Masters thesis), Massachusetts Institute of Technology, 2015.
[24] S. Grzonka, G. Grisetti, W. Burgard, A fully autonomous indoor quadrotor, IEEE Trans. Robot. 28 (1) (2012) 90–100.
[25] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, P. Abbeel, Finding locally optimal, collision-free trajectories with sequential convex optimization, in: Robotics: Science and Systems, Vol. 9, Citeseer, 2013, pp. 1–10.
[26] S. Lavalle, Planning Algorithms, Cambridge University Press, 2006.
[27] S.M. LaValle, J.J. Kuffner, Randomized kinodynamic planning, Int. J. Robot. Res. 20 (5) (2001) 378–400.
[28] L. Janson, E. Schmerling, A. Clark, M. Pavone, Fast marching tree: a fast marching sampling-based method for optimal motion planning in many dimensions, Int. J. Robot. Res. 34 (7) (2015) 883–921.
[29] Y. Li, Z. Littlefield, K.E. Bekris, Asymptotically optimal sampling-based kinodynamic planning, 2014, arXiv preprint arXiv:1407.2896.
[30] I.M. Ross, F. Fahroo, Issues in the real-time computation of optimal control, Math. Comput. Modell. 43 (9) (2006) 1172–1188.
[31] E. Schmerling, L. Janson, M. Pavone, Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics, in: Proc. IEEE Conf. on Decision and Control, 2015.
[32] S.S. Ge, Y.J. Cui, Dynamic motion planning for mobile robots using potential field method, Autonomous Robot. 13 (3) (2002) 207–222.
[33] E. Schmerling, L. Janson, M. Pavone, Optimal sampling-based motion planning under differential constraints: the driftless case, in: Proc. IEEE Conf. on Robotics and Automation, Seattle, WA, 2015, pp. 2368–2375.
[34] D.M. Stipanovic, I. Hwang, C.J. Tomlin, Computation of an over-approximation of the backward reachable set using subsystem level set functions, Dyn. Continuous Discrete Impuls. Syst. 11 (2004) 397–412.
[35] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[36] J. Nocedal, S.J. Wright, Numerical Optimization, second ed., Springer, 2006.
[37] F. Fahroo, I.M. Ross, Direct trajectory optimization by a chebyshev pseudospectral method, AIAA J. Guidance Control Dyn. 25 (1) (2002) 160–166.
[38] T. Lee, M. Leok, N.H. McClamroch, Nonlinear robust tracking control of a quadrotor UAV on SE (3), Asian J. Control 15 (2) (2013) 391–408.
[39] E. Schmerling, L. Janson, M. Pavone, Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics (extended version), 2015, 1–8, Available at http://arxiv.org/abs/1405.7421/.

Ross Allen is a member of the Control and Autonomous Systems Engineering Group at MIT Lincoln Laboratory. He earned his Ph.D. in Aeronautics & Astronautics from Stanford University in June 2016. At Stanford, he was the UTRC Fellow in Aerospace Systems. He has also worked as a mission designer in the GNC group at SpaceX, a roboticist at the NASA Jet Propulsion Laboratory, and one of the first employees at Lily Robotics developing an autonomous aerial camera. Ross received his M.S. in Aeronautics & Astronautics at Stanford University and his B.S. in Aerospace Engineering at Illinois Institute of Technology.

Dr. Marco Pavone is an Associate Professor of Aeronautics and Astronautics at Stanford University, where he is the Director of the Autonomous Systems Laboratory and Co-Director of the Center for Automotive Research at Stanford. Before joining Stanford, he was a Research Technologist within the Robotics Section at the NASA Jet Propulsion Laboratory. He received a Ph.D. degree in Aeronautics and Astronautics from the Massachusetts Institute of Technology in 2010. His main research interests are in the development of methodologies for the analysis, design, and control of autonomous systems, with an emphasis on self-driving cars, autonomous aerospace vehicles, and future mobility systems. He is a recipient of a number of awards, including a Presidential Early Career Award for Scientists and Engineers from President Barack Obama, an ONR YIP Award, an NSF CAREER Award, and a NASA Early Career Faculty Award. He was identified by the American Society for Engineering Education (ASEE) as one of America's 20 most highly promising investigators under the age of 40. His work has been recognized with best paper nominations or awards at the IEEE International Conference on Intelligent Transportation Systems, at the Field and Service Robotics Conference, at the Robotics: Science and Systems Conference, at the ROBOCOMM Conference, and at NASA symposia. He is currently serving as an Associate Editor for the IEEE Control Systems Magazine.