A Real-Time Framework for Kinodynamic Planning with Application to Quadrotor Obstacle Avoidance

Ross Allen and Marco Pavone*

The objective of this paper is to present a full-stack, real-time kinodynamic planning framework and demonstrate it on a quadrotor for collision avoidance. Specifically, the proposed framework utilizes an offline-online computation paradigm, neighborhood classification through machine learning, sampling-based motion planning with an optimal control distance metric, and trajectory smoothing to achieve real-time planning for aerial vehicles. The approach is demonstrated on a quadrotor navigating obstacles in an indoor space and stands as, arguably, one of the first demonstrations of full-online kinodynamic motion planning; exhibiting execution times under 1/3 of a second. 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.

I. 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, 2, 3]. A standing problem in the field of quadrotor control is the achievement of real-time, high-velocity obstacle avoidance. 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 quadrotor control [4]. This paper presents a full-stack approach for kinodynamic motion planning, trajectory smoothing, and trajectory control along with validating experiments. This is arguably one of the first, if not the first demonstration of truly real-time kinodynamic planning on a quadrotor system.

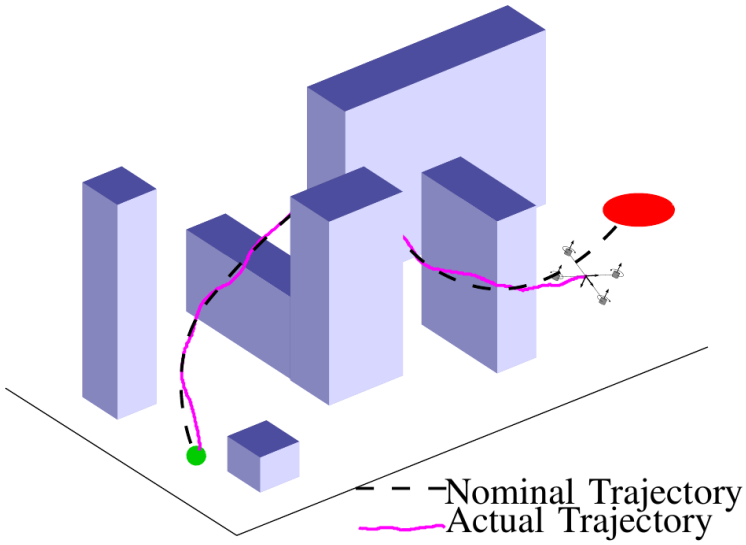

Figure 1. Conceptual diagram of a quadrotor tracking a kinodynamic motion plan through an obstructed environment.

*Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305. Email: {rallen10, pavone}@stanford.edu.

Related Work: To date, the most relevant and progressive work in obstacle avoidance and control of quadrotors is, arguably, that of Richter, Bry, and Roy [5, 6]. Relying on foundational work by Mellinger et. al. [7], Richter's work demonstrated aggressive maneuvers for quadrotors flying in obstructed indoor environments. This was accomplished by generating a set of waypoints through the configuration space and then developing a minimum-snap, polynomial trajectory connecting these waypoints. This minimum-snap trajectory produces a "graceful" flight pattern and guarantees dynamic feasibility [7]. Using the differentially flat dynamics of a quadrotor [7], the trajectory polynomials are used to generate analytical expressions for control inputs that are used in a feedforward fashion in the quadrotor flight controller [5].

While Richter's work represented an important step toward quadrotor planning and control, there remain several critical aspects yet to be achieved. Foremost, the planning algorithm used, RRT* [8], was not implemented in a real-time fashion. The planning phase was accomplished offline, with an a priori map of 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 [5]. 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 at online initiation.

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 [9]. 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 [10]. 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.

Frazzoli et. al. provided some of the pioneering work on real-time kinodynamic motion planning [11]. 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. Demonstrating 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 10s of milliseconds. The theory was even applied to dynamic obstacles; however computation times inflated to 10s of seconds. The major shortcoming of this approach is the restrictive nature of "trim trajectories" that prevents the motion planner from achieving *completeness* and is highly reliant of the user to select appropriate motion primitives. For the helicopter example in Frazzoli's work, 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 manuevers 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 "poorly designed" for this specific case. The work presented in this current paper does not require the user to select specifically tailored motion primitives, therefore remaining more applicable to arbitrary obstacle sets. Furthermore, it includes a notion of time optimality.

Other works have approached the topic of motion planning for quadrotors, even so far as real-time planning. Cowling et. al. [12, 13], and Bouktir et. al. [14] 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* [15]. This work avoided the explicit obstacle representation found in Bouktir et. al. and Cowling et. al. and demonstrated kinodynamic planning for a simulated quadrotor system with linearized dynamics. The Kinodynamic RRT* algorithm is shown to execute in 10s to 100s of seconds; therefore failing to achieve real-time execution.

An additional aspect is validation on a physical system. The papers Frazzoli et. al. [11], Cowling et. al. [12, 13], Bouktir et. al. [14], Webb and van den Berg [15] 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 [16]. Landry's work, however, is not real time, as it requires the entire problem to be solved ahead of time before online execution. 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 [17]. 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. In contrast, our work demonstrates a kinodynamic planner for quadrotor obstacle avoidance at high speeds.

Contribution: The contribution of this paper is a full-stack approach for achieving real-time, kinodynamic motion planning and trajectory control of a quadrotor navigating an arbitrary obstacle configuration. Our paper, arguably, provides the first demonstration of truly real-time kinodynamic planning and control of a quadrotor. Our approach and key intellectual contribution is to integrate three components of planning and control into one seamless architecture: the machine-learning-based, real-time, kinodynamic framework [18]; minimum snap trajectory generation [7]; and the nonlinear feedforward/feedback quadrotor controller [19].

Organization: The paper is structured as follows. Section II gives a formal definition of the kinodynamic planning problem we wish to solve. Section III presents the dynamical model of the quadrotor platform. Section IV develops the real-time kinodynamic planning framework and details how each component of the framework is tailored to the quadrotor system. Section V presents the experimental setup and results, validating the framework. Finally, in Section VI we draw our conclusions and presents directiosn for future research.

II. 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., $\boldsymbol{x}(t) \in \mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ (where $\mathcal{X}$ denotes the state space). 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:

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

where $\boldsymbol{f}_l$ and $\boldsymbol{f}_u$ are the lower and upper bounds for the system dynamic differential inclusion (note that, for generality, the dynamics are represented as a differential inclusion), t_{init} represents the given, fixed initial planning time, and t_{final} represents the *free* final time.

Note that if $\mathcal{X}_{\text{free}}$ can be explicitly represented, then the Optimal Kinodynamic Planning Problem may best be solved using existing optimal control methods, similar to what is presented in [20]. However, we are concerned with cases where $\mathcal{X}_{\text{free}}$ (a subset of the system *state* space) is difficult or not even possible to be *explicitly* represented (as is typical for kinodynamic planning problems [21]), and we are only allowed the ability to perform query-based collision checks.

For the quadrotor planning problem discussed in this paper, we choose a minimum-time cost function, that is:

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

In the following section we specialize the dynamical differential constraints, i.e., $\boldsymbol{f}[\dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{u}(t), t]$, to the case of a quadrotor system.

III. Quadrotor Dynamics

III.A. Nonlinear Dynamics

A quadrotor is modeled as an underactuated rigid body where net thrust is constrained along the $-\vec{z}_B$ axis (see Figure 2). The diagram given in Figure 2 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 Eqn. (3) [19]:

$$\begin{aligned} &\dot{\vec{\xi}}_B = \frac{d^W \vec{\xi}_B}{dt}, \qquad m\ddot{\vec{\xi}}_B = mg\vec{z}_W - u_1\vec{z}_B, \\ &\dot{R}_{BW} = R_{BW}\widehat{\Omega}_{BW}, \qquad J_B\dot{\vec{\Omega}}_{BW} = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \vec{\Omega}_{BW} \times J_B\vec{\Omega}_{BW}. \end{aligned} \tag{3}$$

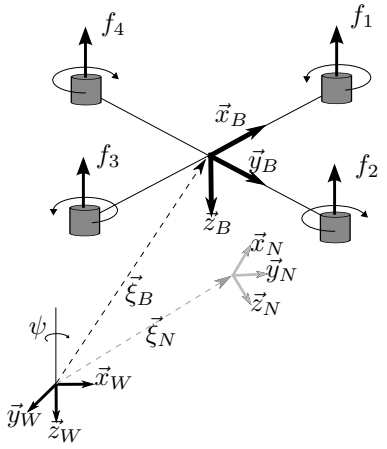

Figure 2. Diagram of quadrotor dynamics with world (inertial), body, and nominal reference frames.

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, $\vec{\Omega}_{BW}$ 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 $\mathbf{u} = [F_{z_B}, M_{x_B}, M_{y_B}, M_{z_B}] \in \mathbb{R}^4$ 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 $\mathbb{R}^3$) [19].

III.B. Approximate Dynamics

There are no known analytical solutions to the minimum-time optimal control problem under the quadrotor's nonlinear dynamics (3). While numerical solutions are possible [18], they are computationally expensive. 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 Section IV and IV.A for more details) [15]. 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 (Section IV.E) [7]. The double integrator dynamics are given as

$$\begin{aligned} &\dot{\mathbf{x}}(t) = A\mathbf{x} + B\mathbf{u} + \mathbf{c} \\ &\text{where: } A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ I \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 \\ g \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \vec{\xi}_B \\ \dot{\vec{\xi}}_B \end{bmatrix} \in \mathbb{R}^6, \quad \mathbf{u} = \ddot{\vec{\xi}}_B \in \mathbb{R}^3 \end{aligned} \tag{4}$$

Figure 3. The real-time framework for kinodynamic planning and control of a quadrotor.

IV. Real-Time Kinodynamic Planning Framework

Sampling-based planning algorithms have become the accepted approach for planning in high-dimensional spaces where state (obstacle) constraints are only implicitly represented [21]. 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" [21]. In this way, a complex trajectory control problem is broken down into a series of many smaller, simpler optimal boundary value problems (OBVP) [a] 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. This is prohibitively expensive [25].

To address this we wrap a sampling-based planner in a real-time framework, given in Fig. 3, that minimizes the number of OBVPs that need to be solved online. The "philosophy" of the framework can be condensed to:

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

To elaborate more, the framework (originally proposed in our earlier work [18]) 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 $N_{\text{pair}} \leq N_s\ (N_s - 1)$– 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`.

[a]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 propogator based on randomized or deterministically chosen control inputs [22]. These techniques are prone to "wander" through the state space, lacking the optimality guarantees of algorithms such as RRT*, PRM*, and FMT* [8, 23]. 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 100s of seconds which is too slow for real-time implementation [24].

The OBVP solution subroutine, `SolveOBVP`, which is often referred to as a "steering function" in the motion planning literature, is given in Section IV.A. The look-up table `Cost` can equivalently be thought of as an 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 IV.B. The cost threshold of the reachable set, often referred to a "neighborhood radius" in the motion planning literature, is the user defined value J_{th}.

Algorithm 1 Offline Phase for the Kinodynamic Motion Planning Framework

$V \leftarrow \texttt{Sample}(\mathcal{X}, N_s)$
$A \leftarrow \texttt{SampleData}(V, N_{\text{pair}}, \text{replace})$
$B \leftarrow \texttt{SampleData}(V, N_{\text{pair}}, \text{replace})$
$\texttt{Cost} \leftarrow \texttt{SolveOBVP}(A, B)$
$\texttt{NearSVM} \leftarrow \texttt{TrainClassifier}([A, B], \texttt{Cost}(A, B), J_{\text{th}})$

At the initiation of the online phase, obstacle data is presented along with the start state, x_{init}, and goal region, $\mathcal{X}_{\text{goal}}$[b]. 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 $N^{\text{out}}_{\text{init}}$ and storing in $N^{\text{in}}_{\text{goal}}$, respectively (see Section IV.B 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 solved to $O(1)$!

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 (AO) FMT* algorithm for its efficiency (see [23] for a detailed discussion of the advantages of FMT* over state-of-the-art counterparts) and kinodynamic extension [26]. The Kinodynamic Fast Marching Trees algorithm ($kino$-FMT) (adapted from [26]) 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 IV.C).

Algorithm 2 Online Phase for the Kinodynamic Motion Planning Framework

$X_{\text{goal}} \leftarrow \texttt{Sample}(\mathcal{X}_{\text{goal}}, N_{\text{goal}})$
$N^{\text{out}}_{\text{init}} \leftarrow \texttt{NearSVM}(x_{\text{init}}, V \backslash \{x_{\text{init}}\}, J_{\text{th}})$
$N^{\text{in}}_{\text{goal}} \leftarrow \texttt{NearSVM}(V \backslash \{X_{\text{goal}}\}, X_{\text{goal}}, J_{\text{th}})$
for $x \in V$ **do**
  if $x \in N^{\text{out}}_{\text{init}}$ **then**
    $\texttt{Cost} \leftarrow \texttt{SolveOBVP}(x_{\text{init}}, x)$
  if $x \in N^{\text{in}}_{\text{goal}}$ **then**
    $\texttt{Cost} \leftarrow \texttt{SolveOBVP}(x, X_{\text{goal}})$
$\texttt{Path} \leftarrow kino\text{–}\texttt{FMT}\,(V, \texttt{Cost}, x_{\text{init}}, X_{\text{goal}})$
return $\texttt{SmoothPath}(\texttt{Path})$

Finally the sequence of states generated by $kino$-FMT is used as a set of waypoints for a path smoothing algorithm that generates a minimum-snap, dynamically feasible trajectory for the quadrotor (see Section IV.D). Mapping the differentially flat output variables from the smooth trajectory back to the full state and control space (Section IV.E), we can provide feedforward terms to the flight controller (Section IV.F).

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).

IV.A. Analytical Solution to OBVP

As explained in Section II, we minimize computations by approximating our system as the double integrator given in Eqn (4). This approximation enables 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 IV.E. The results in this section come from the works [15, 27].

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

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

$$\mathcal{J}[\mathbf{u},\tau] = \int_0^\tau 1 + \mathbf{u}[t]^{\mathrm{T}} R_u \mathbf{u}[t]\,\mathrm{d}t, \tag{5}$$

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 Eqn. (6) where $R_u = w_R I$ and w_R is the control penalty weight [15, 27]:

$$\mathcal{J}^*[\tau] = \tau + \|\mathbf{x} - \bar{\mathbf{x}}[\tau]\|^{\mathrm{T}} \mathbf{d}[\tau]. \tag{6}$$

The corresponding control and state trajectories as functions of time t, for a fixed final time τ, are given in Eqn. (7), respectively [15, 27]:

$$\begin{aligned} \mathbf{u}[t] &= R_u^{-1} B^{\mathrm{T}} \exp\left[A^{\mathrm{T}}(\tau - t)\right] \mathbf{d}[\tau], \\ \mathbf{x}[t] &= \bar{\mathbf{x}}[t] + G[t] \exp\left[A^{\mathrm{T}}(\tau - t)\right] \mathbf{d}[\tau], \end{aligned} \tag{7}$$

where

$$\begin{aligned} \mathbf{d}[\tau] &= G[\tau]^{-1}\left(\mathbf{x} - \bar{\mathbf{x}}[\tau]\right), \\ G[t] &= \frac{1}{w_R} \begin{bmatrix} t^3/3 & 0 & 0 & t^2/2 & 0 & 0 \\ 0 & t^3/3 & 0 & 0 & t^2/2 & 0 \\ 0 & 0 & t^3/3 & 0 & 0 & t^2/2 \\ t^2/2 & 0 & 0 & t & 0 & 0 \\ 0 & t^2/2 & 0 & 0 & t & 0 \\ 0 & 0 & t^2/2 & 0 & 0 & t \end{bmatrix}, \\ \bar{\mathbf{x}}[t] &= \exp\left[At\right] \mathbf{x}_0 + \left[0, 0, gt^2/2, 0, 0, gt\right]^{\mathrm{T}}, \end{aligned} \tag{8}$$

Note that Eqns. (6) and (7) require a *fixed* final time τ The optimal final time τ^* is found as argmin $\mathcal{J}[\tau]$,which can be solved for via a bisection search of Eqn. (6).

IV.B. Machine Learning of Neighborhoods

When the boundary 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 boundary states with their nearest neighbors, as defined by the cost-limited reachable set (see Figure 4). By limiting edge connections from the boundary 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.*

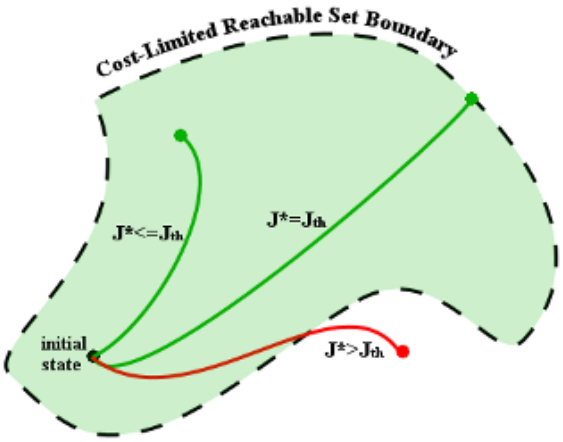

Figure 4. Conceptual representation of a cost-limited reachable set for a notional 2D dynamical system. Formally, a (forward) cost-limited reachable set is the set of states that can be reached from a given state with a cost bounded above by a given threshold (denoted as J_{th}).

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:

$$R^{\text{out}}\left(\boldsymbol{x}_a, \mathcal{U}, J_{\text{th}}\right) := \{\boldsymbol{x}_b \in \mathcal{X} \mid \exists \boldsymbol{u} \in \mathcal{U} \text{ and } \exists t' \in [t_0, t_f] \text{ s.t. } \boldsymbol{x}\left(t'\right) = \boldsymbol{x}_b \text{ and } \mathcal{J}^* \leq J_{\text{th}}\}, \tag{9}$$

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 [28], 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 [29], 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 $\boldsymbol{x}_\mathbf{b}$ 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 Eqn. (10) [30]:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{N_{\text{train}}} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N_{\text{train}}} y^{(i)} y^{(j)} \alpha_i \alpha_j K\left(\mathbf{p}^{(i)}, \mathbf{p}^{(j)}\right) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \ i = 1, \ldots, N_{\text{train}} \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{10}$$

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*, c is a weighting parameter between first and second order terms, and p is 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

$$\texttt{NearSVM}\left(\sum_{i=1}^{N_{\text{train}}} \alpha_i y^{(i)} K\left(\mathbf{p}^{(i)}, \tilde{\mathbf{p}}\right) + b\right). \tag{11}$$

where b is a bias term that is determined as a function of the Lagrange multipliers [30].

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 IV.C. Results on training and testing of the SVM classifier for a quadrotor system are presented in Section V.

IV.C. 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 [23], 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 backward reachable set of x, N_x^{in}. The sample $y_{\text{min}} \in Y_{\text{near}}$ represents the optimal connection point (assuming no obstacles) between x and the existing tree. If the connection from y_{min} and x is free of collisions with obstacles, 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 H ever becomes empty, then $kino$-FMT reports failure. The (asymptotic) optimality properties of FMT* (and its kinodynamic variants) are discussed in [23, 31, 26].

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

IV.D. 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's work [7], Richter et. al. [5] formulate the spline determination as an unconstrained quadratic programming problem that minimizes the integral of the square of the *snap* (i.e. the 4^{th} derivative of position); see Eqn. (12). 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 as they are used in our current approach [5, 6].

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 4^{th} derivative and passes through the sampled states, or "nodes", of the solution trajectory determined in Section IV.C. 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

$$J_{\text{snap}} = \int_0^T P^{(4)}(t)^2 \, dt = \boldsymbol{p}^{\text{T}} Q(T) \boldsymbol{p}, \tag{12}$$

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. Without derivation, the Hessian matrix is given as[c]

$$\begin{aligned} Q_{i,j}(T) &= 2\left(\prod_{k=0}^{3}(i-k)(j-k)\right)\frac{T^{i+j-7}}{i+j-7} \quad \text{for: } i \geq 4 \wedge j \geq 4, \\ Q_{i,j}(T) &= 0 \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{otherwise.} \end{aligned} \tag{13}$$

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 IV.C. 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

$$A\boldsymbol{p} = \boldsymbol{d} \tag{14}$$

$$A = \begin{bmatrix} A_0 \\ A_T \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_T \end{bmatrix} \tag{15}$$

where the terms are given as

$$A_{0_{i,j}} = \begin{cases} \prod_{k=0}^{i-1}(i-k) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{16}$$

$$\boldsymbol{d}_{0_i} = P^{(i)}(0) \tag{17}$$

$$A_{T_{i,j}} = \begin{cases} \left(\prod_{k=0}^{i-1}(i-k)\right)T^{i-j} & \text{if } i \geq j \\ 0 & \text{if } i < j \end{cases} \tag{18}$$

$$\boldsymbol{d}_{T_i} = P^{(i)}(T) \tag{19}$$

Numerical stability can be achieved by reformulating the constrained problem represented in Eqns. (12) and (15) as an unconstrained optimization [5, 6]. This is achieved by optimizing over the polynomial derivatives at the terminal points instead of the polynomial coefficients. Under this reformulation, Eqns. (12) and (15) become

$$J_{\text{snap}} = \boldsymbol{d}^{\mathrm{T}} A^{-\mathrm{T}} Q(T) A^{-1} \boldsymbol{d}, \tag{20}$$

and the polynomial coefficients are determined, a posteriori, via inversion of Eqn. (14).

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

$$\boldsymbol{d}_{\text{total}} = \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}. \tag{21}$$

With this reordering of the derivative vector in Eqn. (21), 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

$$J_{\text{snap}} = \boldsymbol{d}_{\text{total}}^{\mathrm{T}} C A_{1\ldots M}^{-\mathrm{T}} Q_{1\ldots M} A_{1\ldots M} C^{\mathrm{T}} \boldsymbol{d}_{\text{total}}. \tag{22}$$

[c]Note that we diverge from Richter by only considering the minimization on the 4^{th} derivative, where Richter leaves the formulation more general as a weighted sum of squares of derivatives. Furthermore, due to the fact that Richter uses a geometric planner to determine waypoints, his approach requires a time allocation optimization to determine polynomial segment times, T [5, 6]. In contrast, our work determines the polynomial segment times during the time-minimizing kinodynamic planning; see Section IV.C.

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 Eqn. (22) can be written

$$J_{\text{snap}} = \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}. \tag{23}$$

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

$$\boldsymbol{d}_{\text{free}}^{*} = -H_{22}^{-1} H_{12}^{\mathrm{T}} \boldsymbol{d}_{\text{fix}}. \tag{24}$$

Now that the derivatives at each waypoint are determined, the polynomial coefficients can be determined by inverting Eqn. (14). 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 IV.E.

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 [5, 6].

IV.E. Differentially Flat Mapping

The trajectory smoother from Section IV.D produces polynomial splines for position and yaw that are continuous up to their fourth derivative. Mellinger et. al. 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 Eqn. (3) represents a differentially flat system with flat output variables $\vec{\xi}_N$ and ψ_N [7]. This mathematical property proves that the smoothed trajectory from Section IV.D 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 et. al. 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 [7], 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

$$u_{1_{ff}} = -\vec{z}_B \cdot \vec{F}_N, \quad \text{where: } \vec{F}_N = m\ddot{\vec{\xi}}_N - mg\vec{z}_W \tag{25}$$

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

$$\vec{R}_N = \left[{}^{W}\vec{x}_N, {}^{W}\vec{y}_N, {}^{W}\vec{z}_N \right], \tag{26}$$

where

$$\begin{aligned} \vec{z}_N &= -\frac{\vec{F}_N}{\|\vec{F}_N\|} \\ \vec{y}_S &= [-\sin\psi_N, \cos\psi_N, 0]^{\mathrm{T}} \\ \vec{x}_N &= \frac{\vec{y}_S \times \vec{z}_N}{\|\vec{y}_S \times \vec{z}_N\|} \\ \vec{y}_N &= \vec{z}_N \times \vec{x}_N. \end{aligned} \tag{27}$$

The nominal angular velocity vector is given by

$$\vec{\Omega}_{NW} = p_N \vec{x}_N + q_N \vec{y}_N + r_N \vec{z}_N \tag{28}$$

where the individual components of nominal angular velocity are

$$
\begin{aligned}
p_N &= -\vec{h}_\Omega \cdot \vec{y}_N \\
q_N &= \vec{h}_\Omega \cdot \vec{x}_N \\
r_N &= \dot{\psi}_N \vec{z}_W \cdot \vec{z}_N
\end{aligned} \tag{29}
$$

For compactness we have defined

$$
\vec{h}_\Omega = \frac{m}{u_{1_{ff}}} \left(\left(\vec{\xi}_N^{(3)} \cdot \vec{z}_N \right) \vec{z}_N - \vec{\xi}_N^{(3)} \right) \tag{30}
$$

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

$$
\dot{\vec{\Omega}}_{NW} = \alpha_{1_N} \vec{x}_N + \alpha_{2_N} \vec{y}_N + \alpha_{3_N} \vec{z}_N \tag{31}
$$

where the individual components of nominal angular acceleration are

$$
\begin{aligned}
\alpha_{1_N} &= -\vec{h}_\alpha \cdot \vec{y}_N \\
\alpha_{2_N} &= \vec{h}_\alpha \cdot \vec{x}_N \\
\alpha_{3_N} &= \left(\ddot{\psi}_N \vec{z}_N - \dot{\psi}_N \vec{h}_\Omega \right) \cdot \vec{z}_W
\end{aligned} \tag{32}
$$

Again for compactness we give

$$
\begin{aligned}
\vec{h}_\alpha = -\frac{1}{u_{1_{ff}}} \Big(& m \vec{\xi}_N^{(4)} + \ddot{u}_{1_{ff}} \vec{z}_N + 2 \dot{u}_{1_{ff}} \vec{\Omega}_{NW} \times \vec{z}_N \\
& + \vec{\Omega}_{NW} \times \vec{\Omega}_{NW} \times \vec{z}_N \Big)
\end{aligned} \tag{33}
$$

The derivative of the net thrust, which appear in Eqn (33), are derived to be

$$
\begin{aligned}
\dot{u}_{1_{ff}} &= -m \vec{\xi}_N^{(3)} \cdot \vec{z}_N \\
\ddot{u}_{1_{ff}} &= -\left(m \vec{\xi}_N^{(4)} + \vec{\Omega}_{NW} \times \vec{\Omega}_{NW} \times \vec{z}_N \right) \cdot \vec{z}_N
\end{aligned} \tag{34}
$$

Note that the equations presented in this section are taken almost directly from Mellinger et. al. [7] but are stated here for completeness of our approach.

IV.F. Flight Controller

The flight controller is based on work by Lee et. al. and can be consider a form of feedforward/feedback control [19]. Feedforward inputs, denoted with subscript ff, are generated from the differentially flat mapping in Section IV.E and feedback terms, denoted with subscript fb, are generated via proportional-derivative (PD) tracking of position, velocity, orientation and angular velocity. Equation (35) gives the net thrust control input.

$$
\begin{aligned}
u_1 &= u_{1_{ff}} + u_{1_{fb}} \\
&= -\vec{z}_B \cdot \left(m \ddot{\vec{\xi}}_N - m g \vec{z}_W + K_\xi \vec{e}_x i + K_v \vec{e}_v \right)
\end{aligned} \tag{35}
$$

Equation (36) presents the control inputs for the moments about the body axes.

$$
\begin{aligned}
[u_2, u_3, u_4]^T &= [u_2, u_3, u_4]_{ff}^T + [u_2, u_3, u_4]_{fb}^T \\
&= J_B \left(R_B^T R_N \dot{\vec{\Omega}}_{BW} - \vec{\Omega}_{BW} \times \left(R_B^T R_N \vec{\Omega}_{BW} \right) \right) \\
&+ \vec{\Omega}_{BW} \times J_B \vec{\Omega}_{BW} + K_R \vec{e}_R + K_\Omega \vec{e}_\Omega
\end{aligned} \tag{36}
$$

The error terms for feedback control are given by Eqn. (37) [19]

$$
\begin{aligned}
\vec{e}_\xi &= \vec{\xi}_N - \vec{\xi} \\
\vec{e}_v &= \dot{\vec{\xi}}_N - \dot{\vec{\xi}} \\
\vec{e}_R &= \frac{1}{2}\left(R_B^{\mathrm{T}} R_D - R_D^{\mathrm{T}} R_B\right)^\vee \\
\vec{e}_\Omega &= R_B^{\mathrm{T}} R_D \vec{\Omega}_D - \vec{\Omega_B}
\end{aligned}
\tag{37}
$$

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

V. Experimental Results

V.A. Experimental Setup

The real-time framework is demonstrated on a Pixhawk autopilot flown on a DJI F-450 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. 5. Table 1 gives detailed information on the computational platform and programming language for each of the major components of the framework discussed in Section IV. Future work will convert all portions of the online phase (see Alg. 2) to C++ to be run on an embedded processor on the quadrotor.

Table 1. Computational platform and programming language for the major components of the real-time framework.

Process	Reference	Processor	Language
localization	NA	workstation	C++
precomputations	IV	workstation	MATLAB
neighborhood estimation	IV.B	workstation	MATLAB
OBVP solutions	IV.A	workstation	MATLAB
sampling-based planning	IV.C	workstation	C++
min-snap smoothing	IV.D	workstation	MATLAB
flat-to-nonlinear mapping	IV.E	Pixhawk	C/C++
flight control	IV.F	Pixhawk	C/C++

The quadrotor is navigating an indoor environment with dimensions of approximately $3\text{m} \times 4\text{m} \times 3\text{m}$. The obstacle set consists of two parrallel walls with 1.5m openings at opposite ends and a 1.5m separating corridor. This obstacle set is designed to be similar in form to that presented by Webb and van den Berg [15].

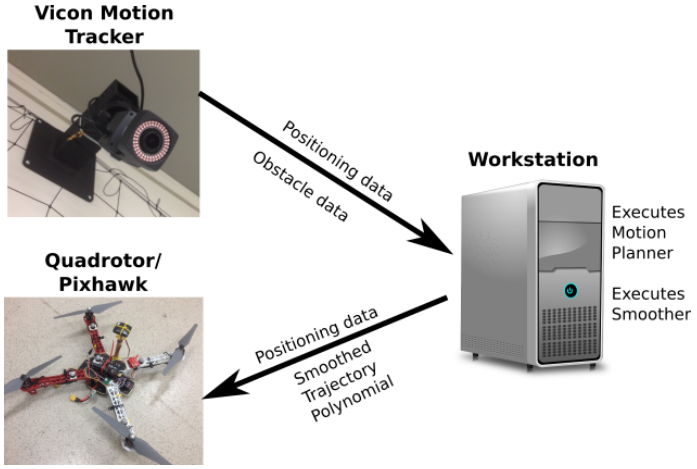

Figure 5. Communication/computation structure for flight tests.

V.B. Numerical Results and Flight Data

The real-time kinodynamic planner was successfully demonstrated in flight tests. The first image in Fig 6 gives an overhead view of the exploration tree generated by $kino$-FMT during execution with the final solution shown in blue. The second image in Fig 6 compares the minimum-time, sampling-based motion plan; minimum-snap smoothed trajectory of the differentially flat output variables; and the flight trajectory that was physically flown by the quadrotor. Fig 7 shows a set of screen captures from a recording of the flight.

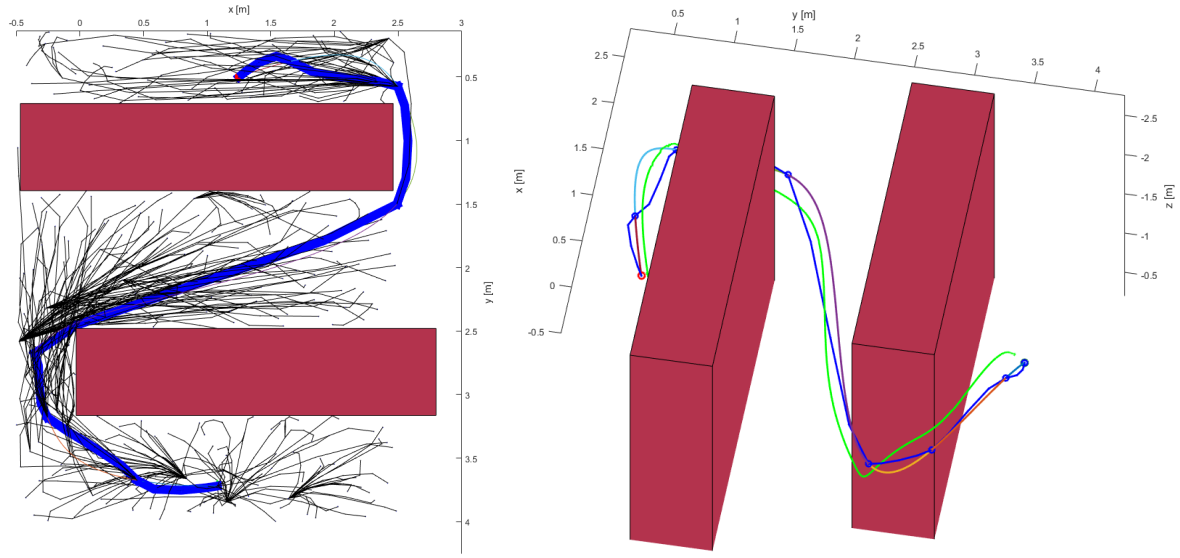

Figure 6. ***(left)*** **Tree explored by the $kino$-FMT algorithm with optimal solution in blue.** ***(right)*** **Motion plan (blue), smoothed trajectory (multi-colored), and flight data (green) shown with the parallel wall obstacles.**

The primary goal of this work was to prove that the entire planning framework can be executed in a real-time environment. The computational timing data and path cost are given in Table 2 for a range of sampled states. It is shown that the entire kinodynamic planning and control problem can be solved in under $1/3$ of a second for 500 sampled nodes. Even with 3000 sampled states, the computation time for the entire planner is under 2 seconds.

To compare this to existing results, Webb and van den Berg simulate an almost identical problem; however they do not perform any path smoothing or communication to a physical quadrotor [15]. With 1000 sampled states Webb and van den Berg's solution takes 51.603 seconds; *i.e.* 120x, or 2 orders of magnitude, slower than the technique presented here. Richter et. al. do not state the computation time for motion plan demonstrated in their work [5]. They do, however, give the computation time for a simplified, 2-dimensional problem that incorporates geometric path planning and minimum-snap path smoothing. Richter's simplified, 2D planning problem takes 3 seconds of computation time; *i.e.* 9.6x, or 1 order of magnitude, slower than the fastest 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 spherical objects [11]. Computation times for a parallel wall obstacle set rise into the 10s of seconds, however Frazzoli is considering the more challenging situation of dynamic obstacles. Direct comparison with Frazzoli's work is more difficult because it only seeks 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.

In Table 2 the computation time is broken down into percentages for the major components of the framework: neighborhood classification for the terminal states (see Sec. IV.B); neighborhood OBVP solutions for the terminal states (see Sec. IV.A); sampling-based motion planning (see Sec. IV.C); path smoothing to generate a minimum-snap, dynamically feasible trajectory (see Sec. IV.D); and communication (see Sec. V.A). 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 seconds per OBVP solution. In comparison, the average `NearSVM` classification time is

Table 2. Path cost and computation time breakdown for the Real-Time Kinodynamic Framework for differing numbers of sampled states

# of Samples	Path Cost [s]	Computation Time [s]	Neighbor Classifier [%]	Neighbor OBVPs [%]	Kino-FMT* [%]	Smoothing [%]	Comms [%]
500	5.4958	0.3125	6.42	37.73	9.43	30.43	25.29
1000	5.4721	0.4293	5.31	41.01	13.60	32.70	4.24
2000	5.2382	0.9242	4.56	41.65	19.81	26.40	3.38
3000	5.2910	1.789	3.08	53.69	29.74	9.65	1.63

1.95×10^{-5} seconds per classification; roughly 1200 times, or three orders of magnitude, faster than OBVP solution. This rapid approximation of neighborhood sets as –opposed to explicit determination via OBVP solutions– is the critically enabling component for real-time implementation.

Table 3. 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	$\lvert\Delta z\rvert$	$(\Delta z)^2$	$(\Delta z)^3$	$\sqrt{(\Delta x)^2+(\Delta y)^2+(\Delta z)^2+(\Delta \dot{x})^2+(\Delta \dot{y})^2+(\Delta \dot{z})^2}$
$\dot{x}_1$	$\dot{x}_2$	$\lvert\Delta \dot{x}\rvert$	$(\Delta \dot{x})^2$	$(\Delta \dot{x})^3$	
$\dot{y}_1$	$\dot{y}_2$	$\lvert\Delta \dot{y}\rvert$	$(\Delta \dot{y})^2$	$(\Delta \dot{x})^3$	
$\dot{z}_1$	$\dot{z}_2$	$\lvert\Delta \dot{z}\rvert$	$(\Delta \dot{z})^2$	$(\Delta \dot{x})^3$	

Due to the reliance on machine-learning of neighbor sets, it is important to determine the classification accuracy of the `NearSVM` algorithm. The feature vector is a 33-element vector, given in Table 3, composed of nonlinear mappings of the boundary value state variables. A third order kernel function is chosen; therefore $p = 3$ in Eqn. (11). 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 10^{th} quantile of all OBVP costs; which for this test campaign evaluated to neighbor cost threshold of roughly 0.69 seconds. In other words, for a given state, roughly 10% of all other states are within 0.69 seconds 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 second cost threshold. On average less than one training error occurred per the 20000 training examples. The algorithm was tested against 30000 additional OBVP examples to ensure that the SVM was not *over-trained* to the training set [d]. 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 [29]. Table 4 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 10^{th} 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[e].

Table 4. 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	Testing Error [%]
20000	0.6	30000	2693	26600.6	371.8	334.6	2.35

[d]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.

[e]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 the classifier is trivial. Therefore a well trained classifier should not be biased toward one type of error.

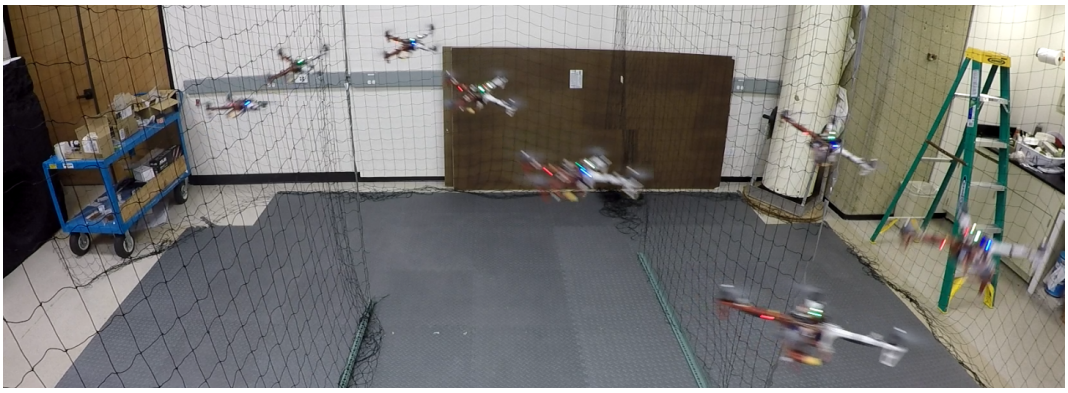

Figure 7. Timelapse of quadrotor navigating obstacles.

VI. Conclusions

This work presents a full-stack, quadrotor planning architecture that is shown to reduce online computation times 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 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.

While this work is targeted at demonstration of real-time planning for a quadrotor system, much of the presented framework remains generally applicable to motion planning for an arbitrary dynamical system. Therefore the presented work is relevant for a wide range of planning and control problems; e.g. spacecraft, automobile, robotic arms, naval craft, etc.

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. To deal with the more challenging scenario of dynamic obstacles, the entire framework will be wrapped in a model predictive control structure to re-plan as obstacle data evolves. Finally, 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.

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

References

[1] Gabriel M Hoffmann, Haomiao Huang, Steven L Waslander, and Claire J Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, volume 2, 2007.

[2] Samir Bouabdallah, Andre Noth, and Roland 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*, volume 3, pages 2451–2456. IEEE, 2004.

[3] Markus Hehn and Raffaello D'Andrea. A flying inverted pendulum. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 763–770. IEEE, 2011.

[4] S. M. LaValle. Motion planning: Wild frontiers. *IEEE Robotics Automation Magazine*, 18(2):108–118, 2011.

[5] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *International Symposium on Robotics Research*, 2013.

[6] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for quadrotor flight. In *International Conference on Robotics and Automation*, 2013.

[7] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. IEEE Conf. on Robotics and Automation*, pages 2520–2525, 2011.

[8] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[9] Koushil Sreenath, Taeyoung Lee, and Vipin 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*, pages 2269–2274. IEEE, 2013.

[10] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, page 0278364911434236, 2012.

[11] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, 25(1):116–129, 2002.

[12] Ian D Cowling, Oleg A Yakimenko, James F Whidborne, and Alastair K Cooke. Direct method based control system for an autonomous quadrotor. *Journal of Intelligent & Robotic Systems*, 60(2):285–316, 2010.

[13] Ian D Cowling, Oleg A Yakimenko, James F Whidborne, and Alastair K Cooke. A prototype of an autonomous controller for a quadrotor UAV. In *Control Conference (ECC), 2007 European*, pages 4001–4008. IEEE, 2007.

[14] Y Bouktir, M Haddad, and T Chettibi. Trajectory planning for a quadrotor helicopter. In *Control and Automation, 2008 16th Mediterranean Conference on*, pages 1258–1263. IEEE, 2008.

[15] D. J. Webb and J. van den Berg. Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints. In *Proc. IEEE Conf. on Robotics and Automation*, pages 5054–5061, 2013.

[16] Benoit Landry. Planning and Control for Quadrotor Flight Through Cluttered Environments. Master's thesis, Massachusetts Institute of Technology, 2015.

[17] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. A fully autonomous indoor quadrotor. *Robotics, IEEE Transactions on*, 28(1):90–100, 2012.

[18] Ross Allen and Marco Pavone. Toward a Real-Time Framework for Solving the Kinodynamic Motion Planning Problem. In *Proc. IEEE Conf. on Robotics and Automation*, Seattle, WA, May 2015. In Press.

[19] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Nonlinear robust tracking control of a quadrotor UAV on SE (3). *Asian Journal of Control*, 15(2):391–408, 2013.

[20] John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization. In *Robotics: Science and Systems*, volume 9, pages 1–10. Citeseer, 2013.

[21] S. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006.

[22] S. M. LaValle and J. J. Kuffner. Randomized Kinodynamic Planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[23] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast Marching Tree: A Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions. *International Journal of Robotics Research*, 34(7):883–921, 2015.

[24] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *arXiv preprint arXiv:1407.2896*, 2014.

[25] I Michael Ross and Fariba Fahroo. Issues in the real-time computation of optimal control. *Mathematical and computer modelling*, 43(9):1172–1188, 2006.

[26] E. Schmerling, L. Janson, and 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. In Press.

[27] Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal Sampling-Based Motion Planning under Differential Constraints: the Drift Case with Linear Affine Dynamics (Extended Version). Available at `http://arxiv.org/abs/1405.7421/`, May 2015.

[28] D. M. Stipanovic, I. Hwang, and C. J. Tomlin. Computation of an Over-Approximation of the Backward Reachable Set Using Subsystem Level Set Functions. *Dynamics of Continuous Discrete and Impulsive Systems*, 11:397–412, 2004.

[29] Ross Allen, Ashley Clark, Joseph Starek, and Marco Pavone. A Machine Learning Approach for Real-Time Reachability Analysis. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 2202–2208, Chicago, IL, September 2014.

[30] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[31] E. Schmerling, L. Janson, and M. Pavone. Optimal Sampling-Based Motion Planning under Differential Constraints: the Driftless Case. In *Proc. IEEE Conf. on Robotics and Automation*, pages 2368–2375, 2015.