Toward a Real-Time Framework for Solving the Kinodynamic Motion Planning Problem

Ross Allen, Marco Pavone

***Abstract*— In this paper we propose a framework combining techniques from sampling-based motion planning, machine learning, and trajectory optimization to address the kinodynamic motion planning problem in real-time environments. This framework relies on a look-up table that stores precomputed optimal solutions to boundary value problems (assuming no obstacles), which form the directed edges of a precomputed motion planning roadmap. A sampling-based motion planning algorithm then leverages such a precomputed roadmap to compute online an obstacle-free trajectory. Machine learning techniques are employed to minimize the number of online solutions to boundary value problems required to compute the neighborhoods of the start state and goal regions. This approach is demonstrated to reduce online planning times up to six orders of magnitude. Simulation results are presented and discussed. Problem-specific framework modifications are then discussed that would allow further computation time reductions.**

I. Introduction

Geometric motion planning, i.e., path optimization and obstacle avoidance for systems *without differential constraints*, has become a mature field in the last two decades [1]. The generalization of geometric planning to kinodynamic motion planning, i.e., systems *with differential constraints* such as momentum or bounded curvature, has been an area of intense research during the last several years. Kinodynamic motion planning is widely seen as an open problem, especially when it comes to computing motion plans in a real-time environment [2]. This is mostly due to the prohibitively high computation times when dealing with complicated dynamics or lack of accuracy/optimality that arise from simplifying assumptions. In this paper we present a framework for solving such problems that drastically reduces the amount of online computations necessary without needing to simplify the dynamics, thus progressing the field of kinodynamic motion planning significantly closer to real-time applications.

Related Work: A standard distinction for motion planning algorithms is whether they are sampling-based (i.e., they avoid an explicit construction of the configuration space) or not [1]. Sampling-based kinodynamic planning received its first formal investigation in 2001 by LaValle and Kuffner [3], where the authors proposed the Rapidly-exploring Random Trees (RRT) algorithm and demonstrated their approach on simulated hovercraft (4- and 6-dimensional state space examples) and spacecraft (6- and 12-dimensional state space examples). Hsu et al. developed the probabilistic roadmap (PRM) algorithm for kinodynamic systems in dynamic environments and demonstrated it on physical robots [4]. Karaman and Frazzoli used results from differential geometry to extend a variation of RRT, RRT*, to a class of nonholonomic dynamic systems [5]. Several recent publications have focused on applying RRT to systems with linear or linearized dynamics such as double-integrators and linear quadcopter models [6, 7, 8]. We note that there has also been some research on *non*-sampling-based kinodynamic planning [9, 10]. These works focus on posing the motion planning problem as an optimal control problem, encompassing obstacles as part of the constraints, and solving the problem with techniques from trajectory optimization such as pseudospectral methods [9] or sequential convex programming [10]. In this paper we adopt a sampling-based approach, arguably the predominant paradigm in the field of planning under differential constraints. However we will also leverage a number of techniques from the work in non-sampling-based trajectory optimization, as detailed below.

Focusing on sampling-based algorithms, work from varying authors identified that two of the key challenges of kinodynamic planning are: 1) the determination of an accurate *distance metric* (also referred to as a cost function) and 2) selection of an appropriate *steering function* [3, 11, 8]. The distance metric quantifies the cost of a motion plan and determines the set of states the are in the neighborhood, or *cost-limited reachable set*, of a given state. The steering function is the function that evaluates the control inputs, and corresponding state trajectory, to connect one sampled node to another. In the foundational work by LaValle and Kuffner, weighted Euclidean distance was employed as the distance metric. The steering function applied to each example is simply a forward propagation of the state using random, admissible controls [3]. Work by Perez et. al. identified that the distance metric and steering function must closely reflect the optimal cost-to-go and optimal trajectory [8]. They attempted to achieve this by linearizing the dynamics and applying a linear-quadratic regulator for distance and steering. Paranjape et. al. developed a steering function for a kinodynamic planning problem that relies on motion primitives [12]. The analytical treatment of these primitives require that they be rigorously tailored to a single, specific set of dynamics. Previous work by the authors demonstrated that machine learning can be effectively used to rapidly approximate distance metrics and neighborhoods dynamic systems, however this work did not go so far as to actually solve any planning problems [13]. This paper adopts the approach of taking optimal cost and optimal trajectories as distance and steering functions, respectively, and proposes to combine offline computation (akin, but not equivalent, to motion primitives) with machine learning techniques for real-time computations.

Perhaps the most relevant prior work is that of Pivtoraiko et. al. who explored the use of state lattices, a notion similar to motion primitives, for solving differentially constrained planning problems [14]. State lattices blur the notions of a steering function and a state sampling method. They involve precomputing a set of dynamically feasible trajectories for the system and then regularly repeating this set of trajectories

Ross Allen and Marco Pavone are with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305 {rallen10, pavone}@stanford.edu.

at the termination of every prior trajectory. The end of each trajectory represents a "sampled" state and the repetition of trajectories form the lattice throughout the state space. Given a start and goal lattice point, graph search algorithms can be used to connect, or *steer through*, lattice points in the state space, thus solving a motion planning problem. Since the admissible states in a state lattice do not span the state space, an arbitrary goal state will be non-reachable and an approximation to the nearest lattice point is necessary. In high-dimensional problems the lattice may be too sparse and this approximation may be unacceptable. Furthermore, since the admissible controls do not span the control space the method will not generally converge to an optimal solution even as the density of lattice points increases. The framework posed in this paper is similar in that it uses precomputed trajectories between states, but the states are sampled randomly instead of regularly repeated. Furthermore, the admissible controls span the entire control space and a portion of online computation time is committed to optimally connecting the start and goal states to the precomputed graph.

Contribution: In this paper we propose a framework that enables the real-time solution to the kinodynamic motion planning problem; the framework being represented in Fig. 1. This framework is designed to be problem-agnostic and therefore can be applied to systems ranging in complexity from linear models, to non-linear dynamics and non-convex constraints, all the way to differential inclusion systems. The framework is described as *enabling* real-time solutions because it drastically reduces the amount of online computation required for any given problem, but does not necessarily provide real-time performance on its own. To achieve truly real-time solutions for a given problem, the subcomponents of the framework must be tailored to leverage specific characteristics of the given planning problem.

The framework employs an *offline-online* computation paradigm, and considers the exact[1] *optimal cost-to-go* and *optimal* trajectories as distance metric and steering function, respectively. The final component of the framework is the kinodynamic variant of the sampling-based Fast Marching Trees algorithm (FMT*) [15], referred to as $kino$-FMT . This variant handles the asymmetry of directed edge connections between two states. We note that the framework is general enough such that other planning algorithms, such as PRM or even RRT*, could be used in place of $kino$-FMT ; however we allocate a brief discussion to $kino$-FMT so as to contribute this novel variant of FMT* to the literature. At the core, the framework works by precomputing a large number of unobstructed, optimal boundary value problems (OBVP) for a given system. This information is then used to train machine learning algorithms that predict cost-limited reachability of newly posed OBVPs. When run in a real-time environment the machine learning algorithms enable rapid approximations of state neighborhoods, minimizing the number of online OBVPs to be solved. Due to the precomputed optimal trajectories, $kino$-FMT 's tree-growth process just consists of a call to a look-up table followed by an online collision check. The framework can be summarized by the philosophy: *exploration through machine learning, decision making through optimal control, precomputation when possible*.

[1]Up to a user-defined tolerance of the optimization solver.

This work marks the unification, extension, and application of several algorithms and techniques previously developed by the authors; including the machine learning based reachability analysis [13] and optimal path planning [15]. While these previous works were developed with the intention of solving kinodynamic planning problems, this is the first work in which this is actually accomplished.

To demonstrate our approach, we apply our framework to two non-linear systems: a simple model of a fixed-wing unmanned aerial vehicle (UAV) navigating a forest and a small spacecraft in zero-gravity navigating a space station. Results show that the framework reduces online computation time by up to 6 orders of magnitude. We then propose techniques for tailoring the framework around specific problem characteristics that would accelerate computation and provide truly real-time results.

Organization: This paper is structured as follows. In Section II we define the optimal motion planning problem. In Section III we present our framework for solving generic problems along with some of the vital subcomponents of the framework. Section IV gives results from numerical experiments using the generic framework and discusses the benefits of the proposed approach. Discussion is then given to potential *tailoring* techniques that could accelerate computation for specific problems. Finally, in Section V we draw some conclusions and discuss directions for future work.

II. Problem Statement

In this section we state the problem we wish to solve. We start with the definition of the geometric path planning problem , i.e. a motion planning problem without differential constraints. Let $\mathcal{X}$ be the configuration space. Let $\mathcal{X}_{\text{obs}}$ be the obstacle region. The obstacle-free space is defined as $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$. The initial condition x_{init} is an element of $\mathcal{X}_{\text{free}}$, and the goal region $\mathcal{X}_{\text{goal}}$ is a subset of $\mathcal{X}_{\text{free}}$. A motion planning problem is denoted by a triplet $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$. A path is denoted by a function $\boldsymbol{x} : [0,1] \to \mathcal{X}$. A path is said to be *collision-free* if $\boldsymbol{x}(\tau) \in \mathcal{X}_{\text{free}}$ for all $\tau \in [0,1]$. A path is said to be a *feasible path* for the planning problem $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$ if it is collision-free, $\boldsymbol{x}(0) = x_{\text{init}}$, and $\boldsymbol{x}(1) \in \mathcal{X}_{\text{goal}}$. Let Σ denote the set of all paths. The path planning problem can then be defined as [15] (some regularity assumptions are avoided here for the sake of brevity):

> **Optimal Path Planning Problem**: Given a path planning problem $(\mathcal{X}_{\text{free}}, x_{\text{init}}, \mathcal{X}_{\text{goal}})$ and an arc length function $c : \Sigma \to \mathbb{R}_{\geq 0}$, find a feasible path $\boldsymbol{x}^*$ such that $c(\boldsymbol{x}^*) = \min\{c(\boldsymbol{x}) : \boldsymbol{x} \text{ is feasible}\}$. If no such path exists, report failure.

In a kinodynamic planning problem, also referred to as simply a motion planning problem, a path is indeed the state evolution of an underlying dynamical system. For the kinodynamic problem, therefore, the problem is extended from choosing an optimal, obstacle-free path to choosing a control function, $\boldsymbol{u}(t)$, that generates an obstacle-free trajectory, $\boldsymbol{x}(t)$, while minimizing a cost function, J. In this sense it is easier to cast the Optimal Motion Planning Problem as a Constrained Optimal Control Problem with obstacles, as shown below.

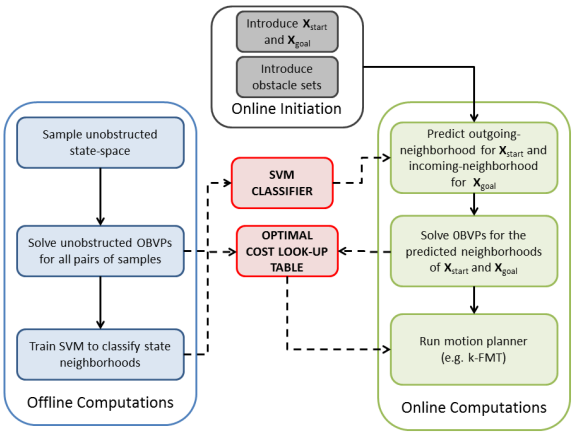

Fig. 1. Flowchart of the Kinodynamic Motion Planning Framework. This diagram also illustrates the extension of prior work where the SVM Classifier and Motion Planner blocks correlate to [13] and [15], respectively.

Optimal Kinodynamic Planning Problem:

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

Where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the trajectory of the state vector in a n-dimensional state space, $\boldsymbol{u}(t) \in \mathbb{R}^m$ is the control function vector in a m-dimensional control space, $\mathcal{U}$ is the admissible control set, and $\boldsymbol{f}$ represents the system dynamics. The problem is kept as general as possible by representing the dynamics as a differential inclusion with $\boldsymbol{f}_l$ and $\boldsymbol{f}_u$ being the lower and upper bound of the inclusion, respectively.

Note that if $\mathcal{X}_{\text{free}}$ can be explicitly represented, then Problem 1 may best be solved using existing optimal control theory, similar to what is presented in [10]. However, we are concerned with cases where $\mathcal{X}_{\text{free}}$ is very difficult to describe. In our problems of interest the best we can hope for is the ability to check the intersection of a given state, $\boldsymbol{x}$, and $\mathcal{X}_{\text{free}}$, during run-time, motivating our adoption of a sampling-based approach. The goal of this paper is to develop a framework that is general enough to solve Problem 1 while minimizing the amount of online computations necessary.

III. A Framework for Optimal Kynodynamic Motion Planning

To solve the Optimal Kynodynamic Planning Problem we apply techniques from optimal control, trajectory optimization and machine learning in an offline/online computation paradigm. The idea is that much of the computationally expensive elements of the motion planning problem can be executed offline in the "laboratory" environment in an effort to minimize the online, "in field" computations. The whole framework is illustrated in Fig. 1.

A. Offline Computations

During the offline computation phase, which is outlined in Algorithm 1, it is assumed that there is a known set of system dynamics and constraints - possibly non-linear - and an unobstructed configuration space. The algorithm begins by randomly or quasi-randomly sampling a large number, N_{sample}, of states from the unobstructed state space, $\mathcal{X}$, which is accomplished by the subroutine `Sample`. The offline phase then proceeds to randomly select a set of states, of size N_{pair}, from the set of sampled states V (with replacement and $N_{pair} \leq N_{sample}^2$) which is accomplished by the `SampleData` subroutine. These sets are then used to solve many, if not all, of the optimal 2-point boundary value problems (OBVP)[2] between each of the sampled state pairs, represented by the sets A and B. The solution of these pairs is accomplished with the `SolveOBVP` subroutine. The optimal cost and optimal trajectory for each of these unobstructed OBVPs is stored in a look-up table, denoted by the subroutine `Cost`, for use during online computations. In this sense, `Cost` may be loosely compared with the notion of motion primitives, but in a form that is handled numerically instead of analytically, and can be applied to an arbitrary system. The optimal cost, along with the associated boundary values, are also used to train a machine learning classification algorithm, such as a support vector machine, to predict whether a given state is cost-limited reachable from another given state [13]. The training is accomplished with the subroutine `TrainClassifier` where J_{th} is the cost threshold for cost-limited reachability classification. The machine learning classifier is denoted `Near`. The trained SVM, `Near`, and the optimal cost look-up table, `Cost`, become the tools for online computation. The use of these online tools can be summarized as "Exploration through machine learning, decision making through optimal control".

Algorithm 1 Offline Computations for the Kinodynamic Motion Planning Framework

1 $V \leftarrow$ `Sample`$(\mathcal{X}, N_{sample})$
2 $A \leftarrow$ `SampleData`$(V, N_{pair}, \text{replace})$
3 $B \leftarrow$ `SampleData`$(V, N_{pair}, \text{replace})$
4 `Cost` $\leftarrow$ `SolveOBVP`(A, B)
5 `Near` $\leftarrow$ `TrainClassifier`$([A, B], \texttt{Cost}(A, B), J_{th})$

B. Online Computations

Upon initiation of online computation the algorithm is presented with the start state, x_{init}, goal region, $\mathcal{X}_{\text{goal}}$, and the obstacles, $\mathcal{X}_{\text{obs}}$, which are all assumed to be unknown beforehand and the obstacles being too difficult to be represented explicitly[3]. By applying the trained SVM, the algorithm proceeds to identify previously sampled states that are in the cost-limited, outgoing-neighborhood of x_{init} and the incoming-neighborhood of $\mathcal{X}_{\text{goal}}$. For each state classified in the neighborhoods of x_{init} or $\mathcal{X}_{\text{goal}}$ an OBVP is solved and recorded in the look-up table. Finally, *kino*-FMT is called to optimally and efficiently connects x_{init} to $\mathcal{X}_{\text{goal}}$ using the `Near` and `Cost` subroutines. The online phase is outlined in Algorithm 2. Its subroutines are discussed in detailed next.

C. `SolveOBVP`: Optimal 2-Point Boundary Value Problem Solver

Many solution techniques exist for approximating the solution for a general optimal control problem, a few of the

[2]Also known as optimal control problems or steering problems.

[3]It is noted that if the start state, goal state, and obstacles are known ahead of time, then the entire motion planning problem can be solved offline and computation time is irrelevant.

Algorithm 2 Online Computations for the Kinodynamic Motion Planning Framework

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

most prominent approaches being: linearization of dynamics and solution via analytical results [8], shooting methods, and direct or indirect trajectory optimization methods [16, 10]. For a general optimal control problem it is not assumed that linearization of dynamics is an acceptable approximation. Shooting methods lack guarantees on enforcement of final conditions for 2-point boundary value problems, a critical need for the sampling-based algorithm presented in this paper. Indirect trajectory optimization methods are often more computationally costly than direct methods as they require solutions to the costate variables in addition to the state variables [17]. This leaves direct trajectory optimization techniques as the favored choice for the presented work.

The method implemented for `SolveOBVP` solves optimal boundary value problems (OBVP) by discretizing the continuous-time problem using the Chebyshev pseudospectral method, transforming it into a nonlinear programming problem (NLP), and solving it using sequential quadratic programming (SQP). While `SolveOBVP` considers state and control constraints, it ignores obstacles which are assumed to be unknown until the online phase of kinodynamic planning.

Without loss of generality we can state that the objective of `SolveOBVP` is the determination of the control trajectory, $\boldsymbol{u}(t)$, and corresponding state trajectory, $\boldsymbol{x}(t)$, that solves the *continuous Bolza problem* [18]:

$$\begin{aligned}
\text{minimize: } & J[\boldsymbol{x}(t), \boldsymbol{u}(t), t_b] = M[\boldsymbol{x}(t_b), t_b] + \\
& \int_{t_a}^{t_b} L[\boldsymbol{x}(t), \boldsymbol{u}(t), t]\, dt \\
\text{subject to: } & \boldsymbol{f}_l \leq \boldsymbol{f}[\dot{\boldsymbol{x}}(t), \boldsymbol{x}(t), \boldsymbol{u}(t), t] \leq \boldsymbol{f}_u \\
& \boldsymbol{g}_l \leq \boldsymbol{g}[\boldsymbol{x}(t), \boldsymbol{u}(t), t] \leq \boldsymbol{g}_u \\
& \psi_l \leq \psi[\boldsymbol{x}(t_a), \boldsymbol{x}(t_b), (t_b - t_a)] \leq \psi_u \\
\text{where: } & t \in [t_a, t_b], \boldsymbol{x}(t) \in \mathbb{R}^n, \boldsymbol{u}(t) \in \mathbb{R}^m
\end{aligned} \tag{2}$$

Note the use of differential inclusions to describe the dynamics $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^n$; also, the state and control constraints $\boldsymbol{g} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^r$ and boundary conditions $\psi : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^p$ are generalizations of the more commonly seen equality constraints. Equality constraints can easily be enforced by setting the upper and lower bounds of any of these equations equal. A key advantage of this formulation is that it permits a wider range of dynamics including differential algebraic equations (DAE), instead of just ordinary differential equations (ODE).

In general, Eqn. 2 requires a numerical treatment. In this paper we apply a solution technique that begins by time-discretizing Eqn. 2 into N segments using the Chebyshev pseudospectral method and then transforming it into a nonlinear programming problem. For brevity, the details of this step are not presented here but are well described by Fahroo and Ross [18]. The result is a discrete problem of the form:

$$\begin{aligned}
\text{minimize: } & J^N[\boldsymbol{X}, \boldsymbol{U}, t_b] = M[\boldsymbol{x}_N, t_b] + \\
& \frac{t_b - t_a}{2} \sum_{k=0}^{N} L[\boldsymbol{x}_k, \boldsymbol{u}_k, t_k]\, w_k \\
\text{subject to: } & \text{for } k = 0, ..., N \\
& \boldsymbol{f}_l \leq \boldsymbol{f}\left[\frac{2}{t_b - t_a}\boldsymbol{d}_k, \boldsymbol{x}_k, \boldsymbol{u}_k, t_k\right] \leq \boldsymbol{f}_u, \\
& \boldsymbol{g}_l \leq \boldsymbol{g}[\boldsymbol{x}_k, \boldsymbol{u}_k, t_k] \leq \boldsymbol{g}_u, \\
& \boldsymbol{\phi}_l \leq \boldsymbol{\phi}[\boldsymbol{x}_0, \boldsymbol{x}_N, (t_b - t_a)] \leq \boldsymbol{\phi}_u
\end{aligned} \tag{3}$$

where $\boldsymbol{X} = [\boldsymbol{x}_0^{\mathrm{T}}, ..., \boldsymbol{x}_N^{\mathrm{T}}]^{\mathrm{T}}$ and $\boldsymbol{U} = [\boldsymbol{u}_0^{\mathrm{T}}, ..., \boldsymbol{u}_N^{\mathrm{T}}]^{\mathrm{T}}$. Now posed in the discrete form, a solution to Eqn. 3 can be attempted using sequential quadratic programming. Fundamentally, SQPs are a heuristic for solving non-convex optimization problems and make no guarantees on solutions. In practice, however, they tend to provide highly reliable results - assuming a reasonable initial guess is provided by the user - and are commonly used [19, 10].

D. `TrainClassifier` and `Near`: Machine Learning for Reachability Set Determination

The `Near` function is critical in enabling real-time execution of the kinodynamic motion planner as it identifies nearby sampled states and reduces the number of edge connection attempts from a given state. In geometric path planning problems, `Near` is very often the Euclidean distance between two states; therefore very inexpensive to calculate. For kinodynamic motion planning, however, `Near` becomes the cost-limited reachable set from a given state. The cost-limited reachable set of state $\boldsymbol{x}_a$ under admissible controls $\mathcal{U}$ and cost threshold J_{th} is defined as R in Eqn 4.

$$\begin{aligned}
R(\boldsymbol{x}_a, \mathcal{U}, J_{th}) = \{\boldsymbol{x}_b \in \mathcal{X} \mid \exists \boldsymbol{u} \in \mathcal{U} \text{ and} \\
\exists t' \in [t_0, t_f] \text{ s.t. } \boldsymbol{x}(t') = \boldsymbol{x}_b \text{ and } J^* \leq J_{th}\}.
\end{aligned} \tag{4}$$

The reachable sets for dynamic systems can be very difficult to assess, often necessitating numerical approximations. The complexity of these numerical approximations is exponential in the dimension of the state space, making them impractical for real-time applications [20].

This motivates a novel approach to reachability analysis for dynamical systems. To this end, machine learning techniques are applied to estimate the cost-limited reachable set, or 'neighborhood', of a given state. This approach is a direct extension of the authors' recent publication on machine learning for reachability analysis [13]. In this paper, a support vector machine (SVM) is used to predict if the optimal cost of traversing from one given state to another is less than or greater than a specified cost threshold. The SVM, which is given the title '`Near`', takes as inputs start and end states and a classification threshold, J_{th}. `Near` is trained with the same data that is used to generate the cost look-up table. Specifically, a training example for `Near` would take in the values of initial and final states, $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, as input variables and the optimal cost to traverse from $\boldsymbol{x}_a$ to $\boldsymbol{x}_b$, J^*, as the

target variable. It has been shown that this technique can classify reachability sets with an error of less than 10% [13].

E. *kino-FMT: Sampling-Based Motion Planner*

In this section we pose a slight variation of the Fast Marching Trees algorithm to accommodate our kinodynamic framework [15]. Alg. 2 calls the *kino*-FMT planning algorithm with the same inputs that FMT* would take: a set of sampled states, V, a neighborhood function, `Near`, and a cost function, `Cost`. The critical change is the distinction of incoming-neighborhoods and outgoing-neighborhoods; that is, the set of states for which a given state is cost-limited reachable (in-neighborhood) and the set of states that are cost-limited reachable from a given state (out-neighborhood). In geometric planning, where the distance metric is often Euclidean distance, the distinction is not necessary since the cost to connect state a to state b is identical to the cost of connecting b to a. This is not the case for kinodynamic planning thus the difference must be reflected in the planning algorithm.

Since the majority of the algorithm is unaffected by this change, only a brief description is given, leaving the thorough discussion of Fast Marching Trees to the original work by Janson and Pavone [15]. The original FMT* algorithm works by building tree-like structures of motion paths using lazy dynamic programming recursion. These trees grow steadily outward in the cost-to-come space. For *kino*-FMT , "paths" are replaced with "trajectories" and the notion of neighborhoods incorporates incoming- and outgoing-neighborhoods. We highlight the change in notation from the original FMT* that N_x^{in} represents the set of sampled states in the incoming-neighborhood of state x, and N_z^{out} represents the outgoing-neighborhood of state z.

IV. Numerical Experiments and Discussion

We apply the real-time kinodynamic framework to two example cases that demonstrate the power and flexibility of this approach. The example cases include a simple model of a fixed wing UAV and a gravity-free spacecraft – i.e., no strong gravitational influence that would produce orbital motion. These examples demonstrate the flexibility of the framework by incorporating characteristics that are typically stumbling blocks for kinodynamic planners. These characteristics include non-linear dynamics (trigonometric terms for turning the UAV), non-linear cost functions (time-optimization causes the final time, t_b, to appear in the denominator of the dynamics), and non-linear control constraints (norm of the thrust is constrained to unity). Not only do these examples illustrate the range of problems that can be handled, they also illustrate the vast reduction in computation time when compared with more naive approaches.

A. *Fixed-wing UAV*

The fixed-wing UAV problem seeks to compute a time-optimal path for a point-mass robot that has Dubins-like dynamics in the xy-plane and single integrator dynamics along the z-axis [21]. This creates a 4-dimensional state space: 3-dimensional position plus xy-heading. The system has a 2-dimensional control space: z-velocity and heading turn rate. Rigid body dynamics, gravity, and aerodynamic effects are ignored. An illustration of the system is given in Fig. 2. The equations of motion and control constraints are given as

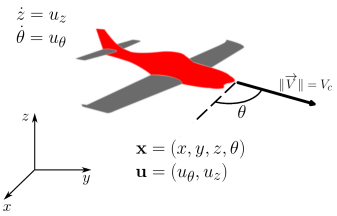

Fig. 2. Simplified fixed-wing UAV Model.

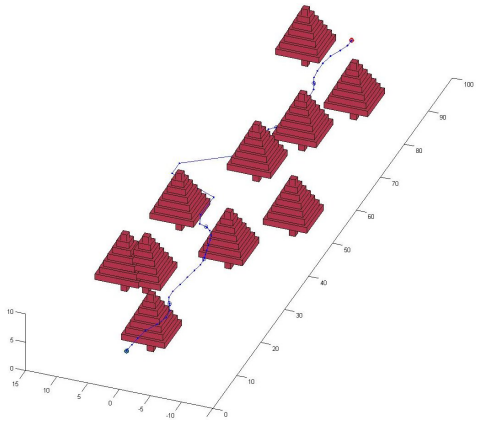

Fig. 3. Time-optimized path for fixed-wing UAV navigating through a forest. Sharp corners are artifacts of plotting, not true representations of the trajectory.

$$\begin{aligned} \dot{x} &= V_c \cos\theta & \dot{z} &= u_z \\ \dot{y} &= V_c \sin\theta & \dot{\theta} &= u_\theta \\ |u_\theta| &\leq \phi_{\max} & |u_z| &\leq V_{z,\max} \end{aligned} \tag{5}$$

where V_c is the constant horizontal speed, ϕ_{max} is the maximum turning rate, $V_{z,\max}$ is the maximum climbrate, and θ is the heading angle with respect to the interial x-axis.

This problem exhibits non-linear dynamics due to the trigonometric terms in x and y. Additionally, the minimization over time, $J = t_b$, necessitates a *free final time* that appears non-linearly in the discrete differentiation operator (See Fahroo and Ross for details [18]).

For the given example problem, the UAV has constant horizontal speed of 10 m/s, max turning rate of π rad/s, and max climb rate of 2.5 m/s. The obstacle set is a simple representation of a pine forest with quasi-randomized tree positions. Fig. 3 gives an aerial view of the optimal trajectory as solved by the kinodynamic planning framework. It is noted that the seemingly sharp corners of the optimal trajectory are actually just artifacts of the MATLAB plotting routine. In fact the Chebyshev pseudospectral method connects intermediate nodes of an optimal trajectory with high order polynomials [18].

B. *Gravity-free spacecraft*

Like the UAV problem, the spacecraft problem seeks a time-optimal path for a point-mass. This problem has the added complexity of a 6-dimensional state space - position and velocity in 3D - along with a 4-dimensional control space - throttle and 3-dimensional pointing vector that is norm-constrained to unity. Rigid body dynamics and gravitational fields are ignored and the change in mass from propulsion

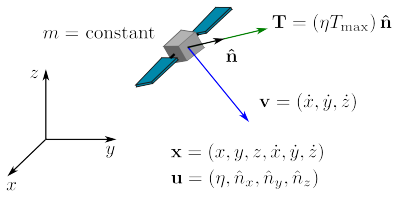

Fig. 4. Dynamics for the gravity-free spacecraft.

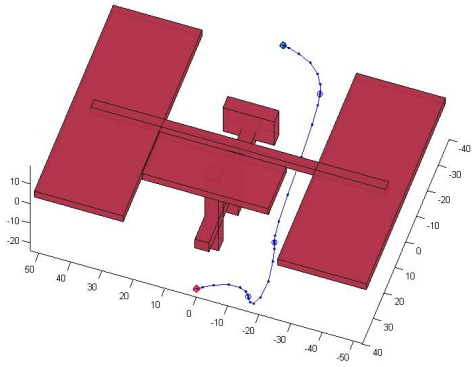

Fig. 5. Time-optimized path for spacecraft navigating around ISS.

is assumed to be negligible. The system is illustrated in Fig. 4 and the equations of motion and control constraints are given as

$$\begin{aligned} \dot{x} &= v_x & \dot{v}_x &= T_x/m \\ \dot{y} &= v_y & \dot{v}_y &= T_y/m \\ \dot{z} &= v_z & \dot{v}_z &= T_z/m \\ \mathrm{T} &= (\eta T_{max})\hat{\mathrm{n}} & 0 \leq \eta &\leq 1 \end{aligned} \tag{6}$$

where T is the thrust vector in 3D, T_{max} is the maximum thrust, η is the throttle, and m is the mass. While the dynamics and constraints appear to be linear and convex, respectively, the problem is in fact non-convex due to the norm constrained pointing vector and the appearance of the free final time in the differentiation operator - as was the case with the UAV.

The obstacle set along with the start and goal states are meant to be a crude representation of a maneuver around the International Space Station. Fig. 5 displays the time-optimal trajectory of the planning problem.

C. Results Summary

For both the UAV and Spacecraft problems, the real-time kinodynamic motion planning framework demonstrates orders-of-magnitude decrease in computation time when compared to naive approaches. Table I summarizes the computation times for differing levels of complexity of the framework. Both problems where run with $N_{sample} = 5000$. The time unit is normalized by the average time it took to solve a single OBVP for the system.

The Naive framework relies on computing a complete digraph of the n sampled states; involving n^2 edge connections[4]. To run a planning algorithm such as $kino$-FMT, we must have knowledge of the neighborhood of a given state, however a neighborhood cannot be defined until there is information on the cost to reach every other state; therefore the naive approach must calculate all n^2 OBVPs to establish neighborhoods. The Neighborhood Learning framework does not use precomputation of OBVPs but reduces the number of necessary online OBVP solutions by estimating a given state's neighborhood with an SVM classifier, and then only attempting an edge connection for those states in the estimated neighborhood. The Real-time framework, which is the complete framework as detailed in Sec. III, reduces computation further by precomputing most of the OBVPs in the offline phase.

[4]an 'edge connection' is identical in meaning to a 'OBVP solution', and a 'node' is identical to a 'sampled state'

TABLE I

COMPARISON OF NORMALIZED, ONLINE COMPUTATION TIMES FOR DIFFERING LEVELS OF FRAMEWORK COMPLEXITY.

	Naive	Neighborhood Learning	Real-time Framework
UAV	2.50×10^7	4.62×10^3	197
Spacecraft	2.50×10^7	4.85×10^3	79

TABLE II

FINAL PATH COST COMPARED WITH OPTIMAL, UNOBSTRUCTED COST.

	Unobstructed [sec]	Real-time Framework [sec]	% Difference
UAV	9.40	11.07	17.77%
Spacecraft	57.445	140.90	145.3%

It is noted that only 10.2% and 25.3% of the online computation time for UAV and Spacecraft problem, respectively, are due to solving OBVPs. The rest of the computation time is dominated by collision checking. The OBVPs solved online represent the edge connections between x_{init} and its out-neighborhood and $\mathcal{X}_{\text{goal}}$ its in-neighborhood and can all be solved in parallel. Therefore parallel processors could reduce the online computation time for OBVPs down to the average solution time of a single OPBVP which is sub-one second for both examples. Further reductions of online computation time is then left to improvements in collision checking algorithms, which is outside the scope of this paper.

To get a measure of how close the real-time framework comes to solving for the true optimal solution, we compare the final path cost with the optimal cost of the unobstructed start-to-goal trajectory. The optimal cost of the unobstructed problem is necessarily a lower limit on the optimal cost of the obstructed problem. Results are given in Table II.

D. Comparison with Existing Techniques

It is difficult to draw direct comparisons with existing techniques because we are proposing a framework, not an explicit algorithm. Existing algorithms for kinodynamic motion planning, such PRM [4] and RRT* [5], can be substituted in place of $kino$-FMT, however they would still just comprise a subcomponent of the framework - not a full alternative to the framework. Comparing these subcomponents has already been performed in prior work [15] and is not relevant to to evaluating the framework as a whole.

In future work, the generic framework will be tailored to a specific problem and applied to a physical system. This will then allow for direct comparison with other techniques that have already been demonstrated on the specific system.

E. Problem Tailoring

The objective of this work was to design a generic, problem-agnostic framework. Starting from this generalized approach, the user can realize further improvements by tailoring aspects of the framework to a specific problem. Here we give a brief discussion of potential tailoring techniques.

The largest room for improvement comes from the large amount of online computation time committed to collision checking. Along with devising a more efficient collision checking algorithm, this can also be improved by wrapping the framework with a model predictive control algorithm such that smaller subproblems are solved in a sequence. Far fewer trajectories would need collision checking in each subproblem.

It was previously noted that parallel processing can greatly reduce the online computation of OBVPs. For the given example problems, anywhere from 10s to 100s of parallel processors could be used, depending on the desired computation time and size of in- and out-neighborhoods of $\mathcal{X}_{\text{goal}}$ and x_{init}.

We have purposefully chosen examples where *favorable* dynamics were not present or ignored (e.g. linearity, convexity, differential flatness, etc.) in an effort to demonstrate the framework on an arbitrary system. If a given problem exhibits any of these favorable characteristics, the `SolveOBVP` subcomponent of the framework may be significantly accelerated. Furthermore, a customized SQP solver can be devised for a given set of dynamics that utilizes the sparsity pattern of the disctretized NLP problem; further reducing computation time [22].

V. Conclusions

We have proposed a framework for solving arbitrary (i.e. non-linear, non-convex, etc.) kinodynamic motion planning problems that moves the field significantly closer to real-time execution times. This framework capitalizes on recent advances in machine learning and trajectory optimization to rapidly explore the state space of a planning problem while ensuring that chosen trajectories obey all dynamic and control constraints; therefore there is no need to linearize the system. This approach is shown to reduce the online computation time of a classic motion planner by up to six orders of magnitude. The major bottle neck in online computation times is the time required to perform collision detection between a robot and the obstacles. In this sense, the presented framework reduces the kinodynamic planning problem to essentially a fast collision-checking problem.

Extensions of this work are currently being pursued. These extension include incorporation of a, efficient collision-checking algorithm, conjunction with model predictive control to produce a closed-loop controller, development of customized constrained optimization solvers, and demonstration on a physical robotic system.

VI. Acknowledgements

This work was supported by an Early Career Faculty grant from NASAs Space Technology Research Grants Program (Grant NNX12AQ43G) and the United Technologies Research Center Graduate Fellowship.

References

[1] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

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

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

[4] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[5] S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Proc. IEEE Conf. on Robotics and Automation*, 2013.

[6] G. Goretkin, A. Perez, R. Platt Jr., and G. Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *Proc. IEEE Conf. on Robotics and Automation*, 2013.

[7] 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*, 2013.

[8] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *Proc. IEEE Conf. on Robotics and Automation*, pages 2537–2542, 2012.

[9] L. R. Lewis and I. M. Ross. A pseudospectral method for real-time motion planning and obstacle avoidance. Technical report, DTIC Document, 2007.

[10] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

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

[12] A. A. Paranjape, K. C. Meier, X. Shi, S. J. Chung, and S. Hutchinson. Motion primitives and 3-D path planning for fast flight through a forest. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2940–2947. IEEE, 2013.

[13] R. Allen, A. Clark, J. Starek, and M. Pavone. A machine learning approach for real-time reachability analysis. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014.

[14] M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[15] L. Janson, E. Schmerling, A. Clark, and M. Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *International Journal of Robotics Research*, 2015. In Press.

[16] J. T. Betts. Survey of numerical methods for trajectory optimization. *AIAA Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[17] B. A. Conway. *Spacecraft Trajectory Optimization*, volume 32. Cambridge University Press, 2010.

[18] F. Fahroo and I. M. Ross. Direct trajectory optimization by a Chebyshev pseudospectral method. *AIAA Journal of Guidance, Control, and Dynamics*, 25(1):160–166, 2002.

[19] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2 edition, 2006.

[20] 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.

[21] M. Hwangbo, J. Kuffner, and T. Kanade. Efficient two-phase 3D motion planning for small fixed-wing UAVs. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1035–1041. IEEE, 2007.

[22] J. Mattingley and S. Boyd. Cvxgen: a code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.