A REAL-TIME FRAMEWORK FOR KINODYNAMIC PLANNING WITH APPLICATION TO QUADROTOR OBSTACLE AVOIDANCE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF
AERONAUTICS AND ASTRONAUTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Ross E. Allen

June 2016

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Any time a robotic system must navigate through an obstructed world to achieve some objective, it is necessarily solving some form of *motion planning problem*. These types of problems are ubiquitous in cutting edge technologies; e.g. an autonomous car navigating through traffic, an aerial delivery drone avoiding buildings and airspaces, a robotic spacecraft performing a safe docking manuever, a humanoid robot acting in close proximity to people, etc. Figure 1.1 gives a few examples of robotic systems that rely on motion planning, but many more could posed. Let us use the quadrotor helicopter as an illustrative example.

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 [4, 5, 6]. 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). In fact, real-time kinodynamic planning represents an open challenge in robotics, in general [7].

Figure 1.1: Current technologies that require the solution to motion planning problems[1].

The challenge of real-time kinodynamic planning can be formulated into two themes that serve to motivate and guide the work presented in this thesis.

> **Motivating Theme 1:** The development of an algorithmic framework that provides a real-time approach for solving the kinodynamic motion planning problem for a general dynamical system.

> **Motivating Theme 2:** Demonstration of real-time motion planning on a physical, kinodynamic system navigating a dynamic environment.

The first theme addresses an existing gap in theory and practice that is present in current literature. The second theme addresses an existing gap in robot experimentation.

In response to such motivating themes, this thesis presents a full-stack approach for kinodynamic motion planning which includes: an offline-online computation paradigm,

[1] *Sources*: (*upper-left*) wired.com/2012/02/autonomous-vehicles-q-and-a/. (*upper-right*) amazon.com/b?node=8037720011. (*lower-left*) geek.com/wp-content/uploads/2014/05/dragon2-2.jpg. (*lower-right*) https://upload.wikimedia.org/wikipedia/commons/9/92/TOPIO-3.jpg

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

1.2 Problem Statement

In this section we mathematically formulate 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 (discussed in more detail in Section 2.1.2). 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 $\mathbf{x}_{\text{init}}$ 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 $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathcal{X}_{\text{goal}})$. A path is denoted by a function $\mathbf{x} : [0, 1] \to \mathcal{X}$. A path is said to be *collision-free* if $\mathbf{x}(\tau) \in \mathcal{X}_{\text{free}}$ for all $\tau \in [0, 1]$. A path is said to be a *feasible path* for the planning problem $(\mathcal{X}_{\text{free}}, \mathbf{x}_{\text{init}}, \mathcal{X}_{\text{goal}})$ if it is collision-free, $\mathbf{x}(0) = \mathbf{x}_{\text{init}}$, and $\mathbf{x}(1) \in \mathcal{X}_{\text{goal}}$. Let Σ denote the set of all paths. The path planning problem can then be defined as [3]:

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

In contrast, the optimal *kinodynamic* planning problem must account for systems with differential constraints. The optimal kinodynamic planning problem consists of the determination of a control function $\mathbf{u}(t) \in \mathbb{R}^m$, and corresponding state trajectory $\mathbf{x}(t) \in \mathbb{R}^n$, that minimizes a cost function $\mathcal{J}(\cdot)$ while obeying control constraints, $\mathbf{u}(t) \in \mathcal{U}$, dynamical (differential) constraints, $\boldsymbol{f}[\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), t]$, and state (obstacle) constraints, i.e. , $\mathbf{x}(t) \in \mathcal{X}_{\text{free}}(t) \subseteq \mathcal{X}$ (where $\mathcal{X}$ denotes the state space and $\mathcal{X}_{\text{free}}(t)$ is the obstacle-free space which is a function of time to account for moving

obstacles). The state at the final time must belong to a given goal region, i.e. , $x(t_{\text{final}}) \in \mathcal{X}_{\text{goal}} \subseteq \mathcal{X}$. Formally, the problem can be posed as a continuous Bolza problem:

Optimal Kinodynamic Planning Problem:

$$\begin{aligned}
\text{Find: } & \mathbf{u}(t) \\
\text{and corresponding: } & \mathbf{x}(t) \\
\text{that minimizes: } & \mathcal{J}[\mathbf{x}(t), \mathbf{u}(t), t_{\text{final}}] \\
\text{subject to: } & \mathbf{u}(t) \in \mathcal{U} && \forall t \in [t_{\text{init}}, t_{\text{final}}] \\
& \mathbf{x}(t) \in \mathcal{X}_{\text{free}} && \forall t \in [t_{\text{init}}, t_{\text{final}}] \\
& \boldsymbol{f}_l \leq \boldsymbol{f}[\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), t] \leq \boldsymbol{f}_u && \forall t \in [t_{\text{init}}, t_{\text{final}}] \\
& \mathbf{x}(t_{\text{final}}) \in \mathcal{X}_{\text{goal}}
\end{aligned} \tag{1.1}$$

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

Note that if $\mathcal{X}_{\text{free}}(t)$ 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 [8]. However, we are concerned with cases where $\mathcal{X}_{\text{free}}(t)$ is difficult or not even possible to be *explicitly* represented (as is typical for kinodynamic planning problems [1]), and we are only allowed the ability to perform query-based collision checks.

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

$$\mathcal{J}[\mathbf{x}(t), \mathbf{u}(t), t_{\text{final}}] = t_{\text{final}}. \tag{1.2}$$

In Chapter 5 we specialize the dynamical differential constraints, i.e. , $\boldsymbol{f}[\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), t]$, to the case of a quadrotor system.

1.3 Related Work

Throughout this thesis we will detail each component of the full-stack planning framework and discuss its relation to the two motivating themes presented in Section 1.1. First, however, let us build a foundation of prior work that sought to answer similar themes. There are two bodies of complementary, yet distinct, literature that are relevant to the work presented here: those works that address real-time motion planning in a general sense and those that focus on planning and control for quadrotors, specifically. We begin by discussing generalized planning and then move onto quadrotor-specific works.

Frazzoli et. al. [9] provided some of the pioneering work on real-time kinodynamic motion planning. 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 on 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 thesis 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.

Leven and Hutchinson developed a real-time path planning framework for changing environments [10]. Their framework, which appears to be tailored to multi-link manipulator robots, relied on a *preprocessing phase* that generated a roadmap of the unobstructed configurations space (i.e. configuration space with no obstacles present). It then developed a mapping from nodes in the unobstructed configuration space to discrete cells in the workspace. When the online phase of the planner, referred to as the *query phase*, was initialized and obstacles were introduced, obstructed cells in the workspace could be mapped to corresponding nodes in the configuration space. These nodes were then removed from the roadmap and planning could occur on this augmented roadmap. This approach yielded impressive online planning times of less than one second.

While Leven's framework is the most similar in form to that presented in our current work - consisting of a framework with offline and online phases to minimize the real-time computations - there are several key differences. Foremost, Leven's work centered on kinematically-constrained, but not dynamically-constrained, robot manipulators. Furthermore they implement a "local planner" that consists of straight-line connections between sampled nodes in the configuration space, thus neglecting some of the kinematic constraints that are fundamental to the manipulators. For the straight-line connection assumption to be valid, they spend considerable time developing a *distance metric* that measures the "closeness" between two configurations. Leven states that the ideal distance metric is swept volume, yet this is too expensive to calculate in real-time so a set of norms in the configuration and workspace are used instead [10]. In our work presented here, we seek to address differentially-constrained systems. To do so we must avoid straight-line approximations for state connections, instead relying on solving an optimal control problem between states (see Section 2.3). Therefore our distance metric indeed becomes optimal cost. As with Leven's work, we face the problem that computing optimal cost maybe too expensive to allow real-time computation. To this end we implement a machine-learning algorithm to approximate the optimal cost when real-time calculations are necessary (see Sections

2.4 and 3.3.3). If our approach were applied to Leven's work, they could directly estimate swept volume instead of relying on norm-based alternatives.

In the context of the second motivating theme, the most relevant and progressive work in obstacle avoidance and control of quadrotors is, arguably, that of Richter, Bry, and Roy [11, 12]. Relying on foundational work by Mellinger et. al. [13], Richter's work demonstrated aggressive maneuvers for quadrotors flying in obstructed indoor environments. This was accomplished by generating a set of waypoints through the workspace and then developing a minimum-snap, polynomial trajectory connecting these waypoints. This minimum-snap trajectory produces a "graceful" flight pattern and guarantees dynamic feasibility [13]. Using the differentially flat dynamics of a quadrotor [13], the trajectory polynomials are used to generate analytical expressions for control inputs that are used in a feedforward fashion in the quadrotor flight controller [11].

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* [14], was not implemented in a real-time fashion. The planning phase was accomplished offline, with an a priori map of obstacles. This leaves Richter's approach unable to handle dynamic obstacles, which is illustrated in their demonstrations that only feature static obstacles. Furthermore, the RRT* algorithm used a simple straight-line metric for the initial planning phase to connect start and goal states; it did not account for the differential motion constraints of the quadrotor [11]. 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 thesis overcomes these shortfalls by employing a *kinodynamic* planner in a truly *real-time* fashion, with obstacle information only available during online execution.

Other works have made significant contributions to the theory of quadrotor control. Sreenath et. al. developed a controller for a quadrotor carrying a cable-suspended load [15]. Hehn and D'Andrea demonstrated stabilization of an inverted

pendulum balanced on a quadrotor [6]. Mellinger et. al. devised a hybrid controller capable of perching a quadrotor on an over-vertical surface [16]. 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 thesis takes kinodynamic planning and flight control as subcomponents of a single problem and proposes a method for addressing both simultaneously.

Several papers have approached the topic of motion planning for quadrotors, even so far as real-time planning. Cowling et. al. [17, 18], and Bouktir et. al. [19] 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 this thesis 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* [20]. This work avoided the explicit obstacle representation found in Bouktir et. al. [19] and Cowling et. al. [17, 18] 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 implementation.

An additional, important aspect in this field is validation on a physical system. The papers Frazzoli et. al. [9], Leven and Hutchinson [10], Cowling et. al. [17, 18], Bouktir et. al. [19], Webb and van den Berg [20] 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 [21]. Landry's work, however, is not real-time, as it requires the entire problem to be solved ahead of time before online execution. As with Richter's

work, Landry's work is, therefore, limited to static obstacles. Grzonka et. al. developed an autonomous quadrotor system capable of navigating highly obstructed indoor environments that executed a variant of the A* algorithm for real-time motion planning [22]. 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.

This thesis is the culmination of the author's prior works. In Allen et. al. [2] we introduce the concept of machine learning for rapid estimation of reachable sets for dynamical systems. In our current work we extend this approach to the control-penalized double integrator (see Section 2.3.4) and show improved estimation accuracy. In Allen and Pavone we first introduce the generalized framework for kinodynamic planning and show how online computation times can be reduced by several orders of magnitude for simulated systems [23]. The subsequent paper applied the kinodynamic planning framework to a quadrotor system and demonstrated real-time planning on a physical system [24]. In our current work we extend the real-time framework to dynamic obstacles by developing an effective, event-based replanning scheme. Furthermore we provide extended simulation results to test the framework in a wider variety of obstacle sets than is possible in a laboratory environment, allowing statistical analysis of the framework performance.

1.4 Contributions

In the pursuit of addressing the two motivational themes, this work resulted in three key contributions. These contributions can be roughly separated into three categories of theoretical, practical, and experimental components.

Theoretical

We provide a novel machine learning technique for the approximation of cost-limited reachable sets of dynamical systems. This technique represents a pivotal element of

real-time kinodynamic planning as it dramatically decreases the amount of online computation necessary for the planning process.

Practical

The novel synthesis of existing theoretical results into a coherent, full-stack framework for kinodynamic planning. It is argued that the framework that emerges from this synthesis is greater than the sum of its parts; realizing online planning times for dynamical systems that had yet to be achieved.

Experimental

We produce, arguably, the first demonstration of truly real-time kinodynamic planning on a physical quadrotor. Demonstrations were performed in environments with static, dynamic, and even adversarial obstacles.

Chapter 2: Mathematical Foundations.

In this chapter we present a set of mathematical tools and concepts upon which the rest of the thesis is built. We introduce the field of sampling-based motion planning; a field that has evolved for the purpose of solving the Optimal Path Planning Problem. We then introduce a general form of a dynamical system. This system represents the differential constraints of the kinodynamic planning problem. We then present a set of example dynamical systems, including the quadrotor system, Dubins Vehicle, a fixed-wing UAV, and the double integrator system. These example systems serve as test cases for the kinodynamic planning framework.

Following on from dynamical systems, we introduce the optimal boundary value problem (OBVP); discussing analytical and numerical results for general and specific OBVPs. The concept of a cost-limited reachable set is then introduced.

Chapter 3: Machine Learning for Real-Time Reachability Analysis.

In this chapter we show how machine learning, particularly support vector machines and weighted linear regression, can be used to approximate the cost-limited reachable set of a dynamical system. We demonstrate this technique on two simulated robotic systems and show testing error of less than 10% accompanied with a reduction in computation time by up to 4 orders of magnitude when compared to numerical solutions for reachable sets. Training data generation and feature vector selection are also discussed.

Chapter 4: A Real-Time Framework for Kinodynamic Planning.

In this chapter 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, performing collision checking in real-time. The machine learning techniques discussed in Chapter 3 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 6 orders of magnitude. Simulation results are presented and discussed. Problem-specific framework modifications are then discussed that would allow further computation time reductions.

Chapter 5: Real-Time Quadrotor Planning and Control.

In this chapter we tailor the real-time planning framework to a quadrotor system and demonstrate it on a physical robotoic platform. Along with the offline-online computation paradigm, machine learning of reachable sets, and the sampling-based planning, the tailored framework also employs trajectory smoothing to achieve real-time planning for aerial vehicles. This framework accounts for dynamic obstacles with an event-based replanning structure and a locally reactive control layer that minimizes replanning events. The approach is demonstrated on a quadrotor navigating moving obstacles in an indoor space and stands as, arguably, one of the first demonstrations of full-online kinodynamic motion planning; exhibiting execution cycles of 3-5 Hz. For the quadrotor, a simplified dynamics model is used during the planning phase to accelerate online computation. A trajectory smoothing phase, which leverages the differentially flat nature of quadrotor dynamics, is then implemented to guarantee a dynamically feasible trajectory.

Chapter 2

Mathematical Foundations

2.1 Fundamentals of Motion Planning

The Optimal Path Planning Problem given in Section 1.2 has been the focus of research for several decades. As a result many techniques for addressing such problems have been developed. This section presents a illustrative example of a planning problem, the foundational concepts and terminology used to discuss planning, and a general class of planning algorithms that are central to this thesis.

2.1.1 Piano Movers Problem

To help guide our discussion on motion planning, let us first consider a illustrative example. Imagine you are tasked with moving a piano from one position in a cluttered apartment to another position. During this move, you must ensure that the piano does not collide with other furniture, walls, doors, the floor, or the ceiling. This example is illustrated in Figure 2.1.

The Piano Movers Problem is indeed a canonical motion planning problem. The concepts of this specific, intuitive problem can be extended to aid discussion of motion planning in general. Borrowing terminology from LaValle [1], we will refer to the piano as a *robot*, symbolized as $\mathcal{A}$ which is a subset of $\mathbb{R}^3$. The apartment can be considered the *world* or *workspace*, $\mathcal{W} \in \mathbb{R}^3$. The furniture, walls, ceiling, and floor represent

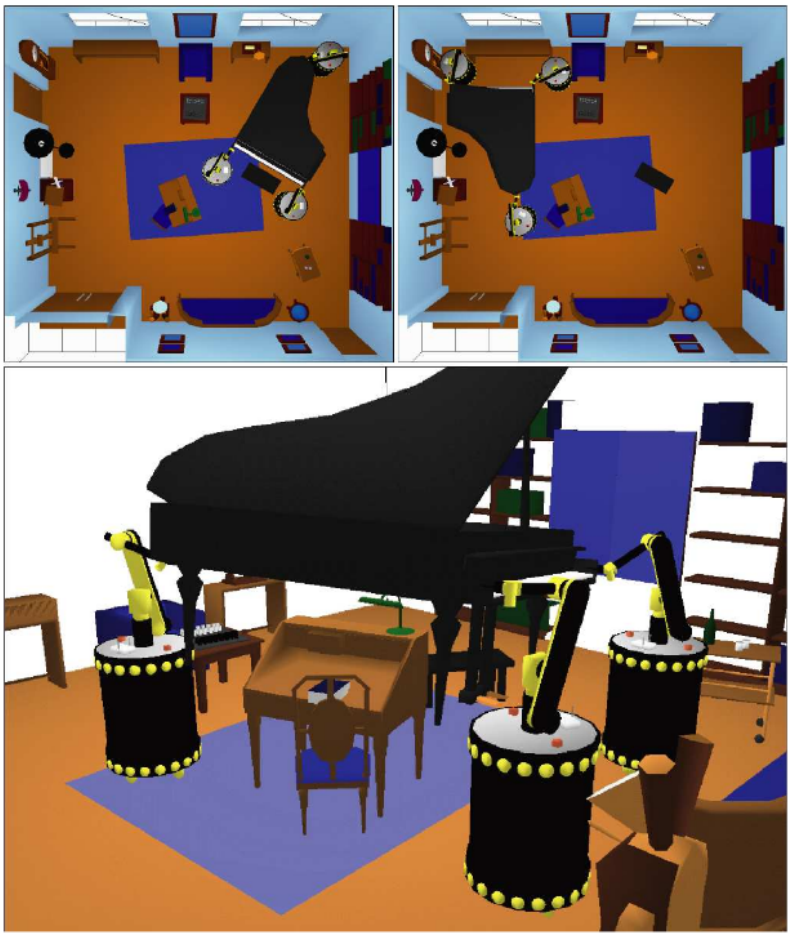


Figure 2.1: Illustration of the piano movers problem. *Source*: Cortes, Juan. Motion planning algorithms for general closed-chain mechanisms. Diss. 2003.

the *obstacle region*, $\mathcal{O} \subset \mathcal{W}$. The motion planning problem can then be discussed as the determination of translations and rotations that move $\mathcal{A}$ through $\mathcal{W} \setminus \mathcal{O}$ from an initial condition to a goal condition or region.

It is important to note that there are several characteristics absent in the Piano Movers Problem that are central to the types of planning problems discussed in this thesis. First of which is the idea of differential constraints. It was assumed that there were no constraints on the motion of the piano so long as it does not interfere with any obstacles. Many robotic systems – such as cars, spacecraft, or quadrotors – have constraints on their motion that are dictated by their kinematics and/or dynamics; see Section 2.2 for more discussion. Furthermore, the Piano Movers problem implied static obstacles where many important planning problems must address the case of moving, dynamic obstacles.

2.1.2 Configuration Space

Now that we have an illustrative example of a planning problem and some initial terminology, we can begin to discuss concepts for solving such problems.

This first important concepts are that of the *state* of the robot, $\mathbf{x}$, and the *state space*, $\mathcal{X}$. The state is a mathematical representation of a specific situation or configuration or "state" of the robot. For example, the state of the piano might be defined by six values; three to define its position and three to define its orientation. The state of a car robot may be defined as the 2D position, heading angle, and velocity. A quadrotor robot may require 12 variables to define its state: 3 for position, 3 for velocity, 3 for orientation, and 3 for angular rates. The *state space* is the set of all possible states that the robot can realize, ignoring obstacles. The field of topology allows for more formal definitions and discussion of these spaces, but such detail is left to LaValle's book [1]. A note on terminology: in the planning algorithms literature the state and state space are commonly referred to as the configuration, q, and configuration space, $\mathcal{C}$, respectively. We will use these terms interchangeably throughout this work.

The configuration space allows us to map our robot – which may be a 3D object

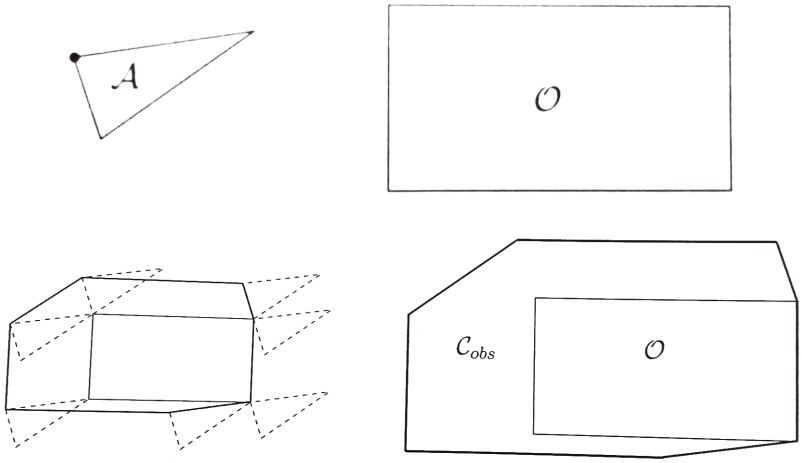


Figure 2.2: Constructing a configuration obstacle for rectangular obstacle and triangular robot, where only translation is allowed for the robot. *Source*: LaValle 2006 [1].

in a 3D world – to a single point in the configuration space. This is a vital concept because now all planning problems, regardless of the shape of the robot, can be solved by finding a feasible/optimal point trajectory through the configuration space. To establish an equivalence between planning in the workspace, $\mathcal{W}$, and the configuration space, $\mathcal{C}$, obstacles must mapped to their appropriate regions in the configuration space, known as *configuration obstacles*, $\mathcal{C}_{obs}$. LaValle gives a simplistic example of such mapping for a 2D, polygonal obstacle in a 2D world where a 2D, polygonal robot is only capable of translation, not rotation [1]. Figure 2.2 illustrates how the robot, $\mathcal{A}$, is "slid" along the boundary of the obstacle, $\mathcal{O}$, to determine the configuration obstacle, $\mathcal{C}_{obs}$.

Extending $\mathcal{C}_{obs}$ construction to a more general case is not as simple as Figure 2.2 may imply. If we extend this analysis to a 3D world while maintaining the restrictive assumptions of translation-only motion and polyhedron robots and obstacles, a configuration obstacle can be constructed in $O(nm)$ time, as a worst case, where n is the number of faces of $\mathcal{A}$ and m is the number of faces of $\mathcal{O}$ [1]. When we relax the translation-only assumption, thus increasing the dimension of the configuration space, the configuration obstacle becomes considerably more complex. To illustrate, if the simple robot in Figure 2.2 is allowed to rotate, the configuration obstacle can

be formed as a set of 73 algebraic primitives; which is a surpisingly large number considering it is a triangular robot and rectangular obstacle [1]. For the more general and applicable case of a non-polyhedral robot with freedom of rotation, construction of configuration obstacles is prohibitively computationally expensive. This difficulty of forming $\mathcal{C}_{obs}$ in higher dimensions motivates the use of sampling-based planning algorithms.

2.1.3 Sampling-Based Planning

Sampling-based motion planners have become the favored approach for planning in high-dimensional configuration spaces. This is due to the fact that sampling-based approaches avoid an explicit representation of the configuration space and instead rely on sampling configurations and checking for collisions with obstacles using a "black box" collision checker. Kinodynamic planning problems tend to be high-dimensional due to the need to capture velocities and orientation rates in the state of the robot; therefore a sampling-based planner is the logical approach for this type of problem. This section gives a brief overview of sampling-based planning. For details on the specific sampling-based algorithm used in this work, refer to Section 4.1.4.

In general sampling-based planners work by randomly or quasi-randomly sampling configurations from $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$ and connecting the samples in a graph structure such that a feasible/optimal path can be found from an initial state, $\mathbf{x}_{\text{init}}$, to a goal region, $\mathcal{X}_{\text{goal}}$. Sampling-based algorithms can be broadly classified into groups based on whether they generate a tree graph (preferred for single-query planning problems), or a roadmap graph (preferred for multi-query problems). Figure 2.3 gives a conceptual illustration of the graph structures that may arise from implementing these algorithms on a simple, 2D planning problem.

There are many variations of sampling-based planners that primarily differ in logic or sequence in which nodes are added to the graph in the configuration space. Perhaps the most widely known sampling-based planners are the rapidly exploring random tree (RRT and RRT*) and the probabilistic roadmap (PRM) algorithms; the work discussed in this thesis implemented the fast marching tree (FMT*) algorithm

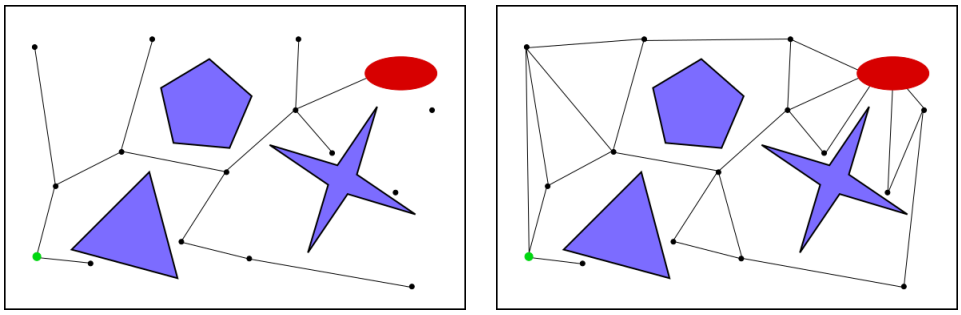


Figure 2.3: Tree (left) and roadmap (right) graphs generated by sampling-based planning algorithms for a simplistic 2D problem. The graphs generated can be used to find a feasible/optimal path from the initial state, shown in green, to the goal region, shown in red, while avoiding the configuration obstacles, shown in blue. A collision checking module prevents edges that would intersect obstacles from being incorporated into the graph.

which has performance advantages over RRT* and PRM [3]. The details of the RRT and PRM algorithms are left to existing literature, such as LaValle [1], and the FMT* algorithm is disucssed in Section 4.1.4.

There are three important concepts that all of these algorithms have in common: state connections, distance metric, and neighborhoods. State connections – also referred to as sample connections, node connections, configuration connections, or graph edges – refer to the way in which edges are formed in the graph between sampled states in the configuration space. State connections, or graph edges, have an associated cost or length, commonly referred to as a *distance metric*. Figure 2.3 represents graph edges as straight lines. For many *geometric* planning problems, i.e. planning without differential constraints, state connections would indeed be straight lines in the configuration space; therefore implying a *distance metric* of Euclidean distance. If there are differential constraints, however, state connections must represent dynamically feasible trajectories for the robot in the configuration space. In such cases the distance metric is associated with the cost of an optimization problem. For example, state connections may represent a minimum time or minimum energy trajectory between states. Therefore, in this work, state connections can be thought

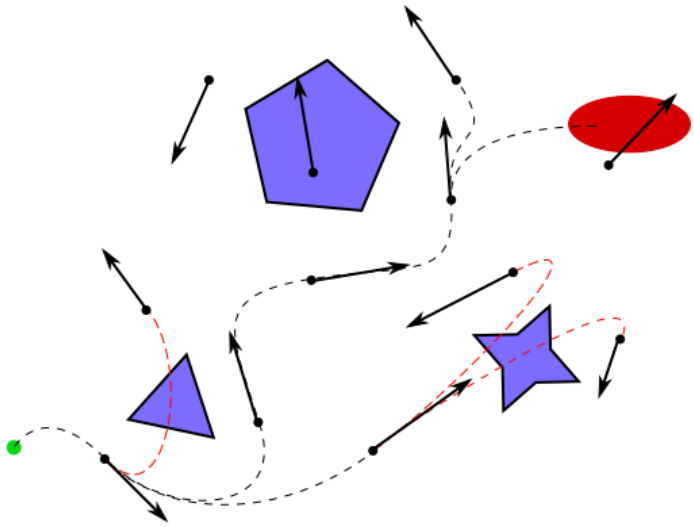


Figure 2.4: A tree structure for a differentially constrained planning problem. State connections interfering with obstacles, as identified by the collision checker, are indicated in red and would be pruned by the sampling-based planning algorithm.

as synonymous without optimal boundary value problems (OBVPs), discussed in Section 2.3. Figure 2.4 illustrates a tree structure for a differentially constrained planning problem where sampling occurs in 2D position and 2D velocity.

Another important concept is that of *state neighborhoods* or *cost-limited reachable sets*. This concept, discussed more in Section 2.4, represents the set of states that are within a given distance or cost of a given state. For sampling-based planners, state connections are only attempted for states within a neighborhood. This is done for computational efficiency: if state connections are attempted between highly separated states, there is a high probability of obstacle collision; thus it represents a "wasted" computation. The size of the neighborhood, often referred to as the *neighborhood radius*, can be user-defined or adaptively generated. Generally speaking, the size of the neighborhood trades off between optimality of the planner solution and computational efficiency. A small neighborhood radius will avoid wasted state connections that are invalid due to obstacle collision but will use more intermediate nodes causing the solution to be less optimal. A large neighborhood radius will find better state connections, short-cutting intermediate nodes, but will result in more obstacle collisions, thus higher computational cost.

2.2 Dynamical Systems

This section introduces the concept and general form of a dynamical system along with the property of differential flatness. Dynamical systems are the distinguishing feature between kinodynamic motion planning and the more simple problem of geometric planning. We then present the equations of motion, i.e. *dynamics*, for the quadrotor (Section 2.2.3), double integrator (Section 2.2.4), Dubins vehicle (Section 2.2.5), fixed-wing UAV (Section 2.2.6), and gravity-free spacecraft (Section 2.2.7) robotic systems. These robotic systems appear throughout this thesis as test cases for the techniques devloped in Chapters 3, 4, and 5 .

2.2.1 Differential Equations

While the concept of dynamical systems encompasses a wider range of mathematical systems – including difference equations, integral equations, etc. – we limit our scope to deterministic differential equations with a single independent variable. Within this scope, the most general form of a dynamical system we discuss is that of a differential inclusion, given in the form

$$\boldsymbol{f}_l \leq \boldsymbol{f}[\dot{\mathbf{x}}(t), \mathbf{x}(t), \boldsymbol{u}(t), t] \leq \boldsymbol{f}_u \tag{2.1}$$

.

Note that, in Equation (2.1), if $\boldsymbol{f}_l = \boldsymbol{f}_u$, then our differential inclusion reduces to a differential algebraic equation and can be written as

$$\boldsymbol{f}[\dot{\mathbf{x}}(t), \mathbf{x}(t), \boldsymbol{u}(t), t] = 0 \tag{2.2}$$

The specific systems, discussed in Sections 2.2.3 - 2.2.6, all are the form of an ordinary differential equation which can generally be expressed as

$$\dot{\mathbf{x}}(t) = \boldsymbol{f}[\mathbf{x}(t), \mathbf{u}(t), t] \tag{2.3}$$

2.2.2 Differential Flatness

We now introduce the concept of *differential flatness* which can be a powerful tool in the analysis and treatment of certain dynamical systems; particularly for the quadrotor system. A system is said to be *differentially flat* if its state and control variables can be represented as explicit functions of a set of "output variables" and their derivatives [25]. These output variables $\mathbf{y}$, termed the *flat output*, are represented mathematically as

$$\mathbf{y} = \boldsymbol{c}\left(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \ddot{\mathbf{u}}, ..., \mathbf{u}^{(\alpha)}\right) \quad \mathbf{y} \in \mathbb{R}^{N_o}, \tag{2.4}$$

where N_o is the number of output variables. For the system to be differentially flat, we must be able to find some functions $\boldsymbol{a}$ and $\boldsymbol{b}$, such that [25]

$$\begin{aligned} \mathbf{x} &= \boldsymbol{a}\left(\mathbf{y}, \dot{\mathbf{y}}, ..., \mathbf{y}^{(\beta)}\right) \\ \mathbf{u} &= \boldsymbol{b}\left(\mathbf{y}, \dot{\mathbf{y}}, ..., \mathbf{y}^{(\beta+1)}\right). \end{aligned} \tag{2.5}$$

Therefore, if a differentially-constrained system is differentially flat, then it can be equivalently represented as a non-differentially-constrained system. This implies that any function in the output space that are differentiable to order $\beta + 1$ is guaranteed to be dynamically feasible in the state and control space. This is a powerful property because trajectory generation in the flat output space is an unconstrained problem, therefore simpler than generating a valid trajectory in the state and control space. Furthermore, the flat output space may be in a reduced dimensional space when compared to the state and control space.

While recasting an optimal control problem using the flat output space may remove dynamic constraints, numerical complications may arise from the nonlinear transformation of the boundary conditions according to Equation 2.5. The cost function also undergoes a similar nonlinear tranformation. Instead of solving an optimization with, potentially, nonlinear dynamic constraints, the differentially flat mapping can produce an optimization problem with nonlinear boundary conditions. Therefore, motion planning for differentially flat systems is not *necessarily* a simplified problem. In Chapter 5 we show how, when properly applied, differential flatness can

be leveraged to accelerate computation of motion plans for a quadrotor system.

2.2.3 Quadrotor

A quadrotor is modeled as an underactuated rigid body where net thrust is constrained along the $-\vec{z}_B$ axis. The diagram given in Figure 2.5 represents the relevant coordinate frames and variables for the quadrotor planning and control problem. The world frame, W, is an inertial frame, which is implemented in our case with a North-East-Down (NED) orientation. The body-fixed frame, B, translates and rotates with the quadrotor. The nominal frame, N, is a target frame for trajectory tracking; therefore in perfect trajectory tracking $B = N$. The quadrotor dynamics are given in Eqn. (2.6) [26]:

$$\begin{aligned}
\dot{\vec{\xi}}_B &= \frac{d^W \vec{\xi}_B}{dt}, \\
\ddot{\vec{\xi}}_B &= mg\vec{z}_W - u_1\vec{z}_B, \\
\dot{R}_{BW} &= R_{BW}\widehat{\Omega}_{BW}, \\
J_B\dot{\vec{\Omega}}_{BW} &= \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \vec{\Omega}_{BW} \times J_B\vec{\Omega}_{BW}.
\end{aligned} \tag{2.6}$$

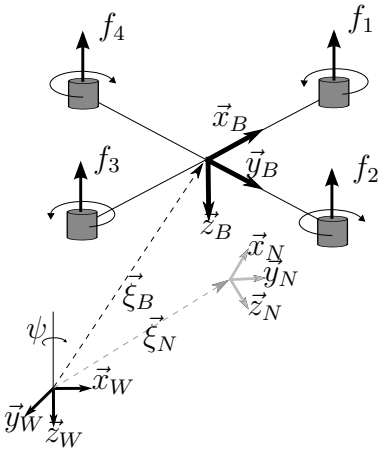


Figure 2.5: Diagram of quadrotor dynamics with world (inertial), body, and nominal reference frames.

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, and $\vec{\Omega}_{BW}$ is the angular velocity of the body frame with respect to the world frame. Gravity acceleration is given by g and 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 $\hat{\cdot}$ denotes the *hat-map* (i.e. , an isomorphism between 3×3 skew-symmetric matrices and vectors in $\mathbb{R}^3$) [26].

It has been shown that quadrotor dynamics represent a differentially flat system [13]. It can be shown that a 4D flat output space can be generated with position in 3D and yaw; i.e. $\mathbf{y} = \left[\vec{\xi}_B, \psi\right]$. The mapping between the state and control variables and the flat output variables is discussed in the context of how it is implemented during flight control in Chapter 5.

2.2.4 Double Integrator

There are no known analytical solutions to the minimum-time optimal control problem under the quadrotor's nonlinear dynamics (2.6). While numerical solutions are possible [23], 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 2.3.4 for more details) [20]. 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

5.1.4) [13]. The double integrator dynamics are given as

$$\dot{\mathbf{x}}(t) = A\mathbf{x} + B\mathbf{u} + \mathbf{c}$$
$$\text{where: } A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ I \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 \\ g \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \vec{\xi}_B \\ \dot{\vec{\xi}}_B \end{bmatrix} \in \mathbb{R}^6, \quad \mathbf{u} = \ddot{\vec{\xi}}_B \in \mathbb{R}^3. \tag{2.7}$$

Note that this approximator-corrector approach for the quadrotor dynamics is one of several trade-offs that arise from trying to address, simultaneously, both motivational themes stated in Section 1.1. The framework presented in Chapter 4 is indeed general enough to accommodate the fully non-linear dynamics of the quadrotor, however, it is desirable to apply an approximation (along with the later correction) to improve online performance during physical demonstrations.

2.2.5 Dubins Vehicle

The Dubins car models a simple non-holonomic car-like land vehicle that is constrained by a maximum turning angle $\phi_{\max}$, and has a fixed forward speed $v \in \mathbb{R}_{++}$. This maximum steering angle imposes a minimum turning radius $\rho_{\min}$ on the vehicle. The dynamics of a Dubins car [27] are given as:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = u = \frac{v}{L}\tan\phi,$$

where the set of admissible controls is $\mathcal{U} = \left[-\frac{v}{L}\tan\phi_{\max}, \frac{v}{L}\tan\phi_{\max}\right]$. The state is given by $\mathbf{x} = [x, y, \theta]$. The control task is concerned with minimizing the total path length, which is equivalent to minimizing the traversal time $\mathcal{J} = t_f - t_0$ since speed is constant. The scenario is depicted visually in Figure 2.6.

This model is used as a simulation test case for the machine learning techniques developed in Chapter 3 and the general planning framework developed in Chapter 4.

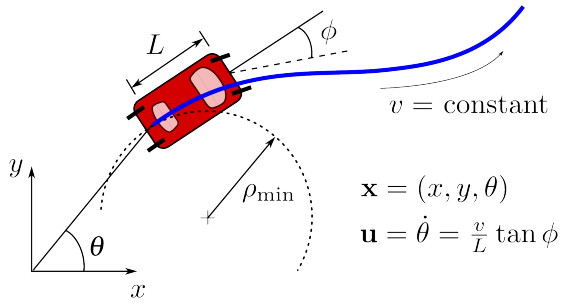


Figure 2.6: Dynamics for the Dubins car model.

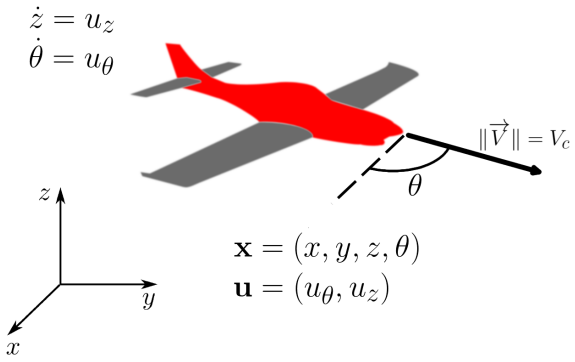


Figure 2.7: Simplified fixed-wing UAV Model.

2.2.6 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 [28]. 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. 2.7. The equations of motion and control constraints are given as

$$\begin{aligned} \dot{x} &= V_c \cos\theta \qquad \dot{z} = u_z \\ \dot{y} &= V_c \sin\theta \qquad \dot{\theta} = u_\theta \\ |u_\theta| &\leq \phi_{\max} \qquad |u_z| \leq V_{z,\max} \end{aligned} \tag{2.8}$$

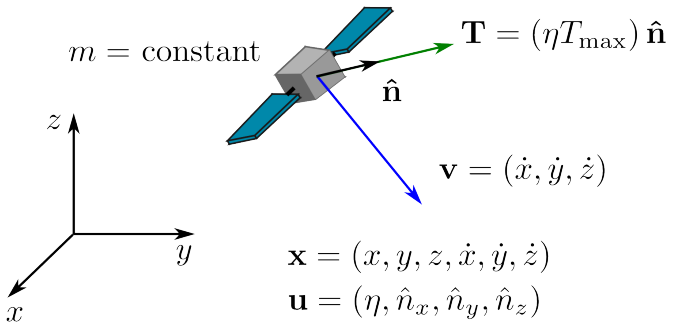


Figure 2.8: Dynamics for the gravity-free spacecraft.

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 [29]).

2.2.7 Gravity-Free Spacecraft

The gravity-free spacecraft has similar dynamics to the double integrator presented in Section 2.2.4, however constraints are now imposed on the control variables. The state space is 6-dimensional - 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 is assumed to be negligible. The system is illustrated in Figure 2.8.

The equations of motion for the gravity-free spacecraft are given in Equation (2.9).

$$\begin{aligned} \dot{x} &= v_x & \dot{v}_x &= {}^{T_x}/_{m} \\ \dot{y} &= v_y & \dot{v}_y &= {}^{T_y}/_{m} \\ \dot{z} &= v_z & \dot{v}_z &= {}^{T_z}/_{m} \\ \mathrm{T} &= (\eta T_{max})\hat{\mathrm{n}} & 0 \leq \eta &\leq 1 \end{aligned} \tag{2.9}$$

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.

2.3 Optimal Boundary Value Problems

As discussed in Section 2.1, sampling-based motion planning consists of generating a graph structure from a set of sampled states that can be searched for an optimal path. For kinodynamic planning problems, the edges of the the graph structure represent solutions to optimal control problems, also referred to as *steering problems* or *optimal boundary value problems*[1] (OBVPs). Here we discuss the the general form of an OBVP and relevant results from the study of optimal control. In Chapters 4 and 5 we will use these results to create a computational subroutine we title `SolveOBVP` for solving boundary value problems. The optimal control or OBVP problem is expressed in Equation (2.10)

[1] we make a distinction with the commonly used term 'boundary value problem' (BVP) which may only refer to a feasibility problem

Optimal Boundary Value Problem:

$$\begin{aligned} \text{Find: } & \mathbf{u}^*(t),\, \mathbf{x}^*(t) \\ \text{that minimizes: } & \mathcal{J}(\mathbf{u}) = h\left(\mathbf{x}(t_f), t_f\right) + \int_{t_0}^{t_f} g(\mathbf{x}(t), \mathbf{u}(t), t)dt \\ \text{subject to: } & \dot{\mathbf{x}}(t) = \boldsymbol{f}(\mathbf{x}(t), \mathbf{u}(t), t) \\ & \mathbf{u}(t) \in \mathcal{U} \\ & \mathbf{x}(t) \in \mathcal{X} \\ & \mathbf{x}^*(t_0) = \mathbf{x}_0,\ \mathbf{x}^*(t_f) = \mathbf{x}_f \end{aligned} \tag{2.10}$$

where t_0 is a specified initial time, t_f is a *free* final time, $\mathbf{u}^*(t)$ is the optimal control function as a function of time, and $\mathbf{x}^*(t)$ is the corresponding optimal state trajectory as a function of time.

Equation (2.10) appears very similar in form to that of the optimal kinodynamic planning problem, Equation (1.1), but there are several important distinctions. The most important difference is that of the admissible state trajectory, $\mathbf{x}(t) \in \mathcal{X}$. This imposes state constraints, e.g. a bounding box on the trajectory, but ignores configuration obstacles, $\mathcal{C}_{obs}$ or $\mathcal{X}_{\text{obs}}$, which represent the crux of the planning problem. If the configuration obstacles could be explicitly represented, the optimal kinodynamic planning problem of Equation (1.1) could indeed be mapped to the optimal boundary value problem of Equation (2.10) and existing optimal control theory could be applied to solve. As discussed in Section 2.1, constructing the configuration obstacles from workspace obstacles can be an intractable problem. Therefore, when performing edge connections in the configuration graph, we resort to solving OBVPs without consideration of configuration obstacles and then perform an a posteriori collision check on the solution.

To discuss analytical results on Equation (2.10), we've applied some assumptions on the form of the cost functional and system dynamics. We've constrained the dynamics to an ordinary differential equation, $\dot{\mathbf{x}}(t) = \boldsymbol{f}(\mathbf{x}(t), \mathbf{u}(t), t)$, as opposed to the more general differential inclusion that was presented in Equation (1.1). For the cost functional we've assumed an integral plus a scalar-valued function of the final

conditions: $h(\mathbf{x}(t_f), t_f) : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$. The functional integrates over a scalar-valued function of the time, state, and control variables: $g(\mathbf{x}(t), \mathbf{u}(t), t) : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}$. This cost functional is general enough to address almost all real-world applications [30].

2.3.1 Analytical Results

There is no known general solution for the optimization problem given in Equation (2.10). We can, however, develop a set of necessary conditions for a solution to Equation (2.10). These necessary conditions can be powerful tools for developing numerical techniques for approximate solutions or deriving analytical solutions to specific problems. We briefly present these conditions here, leaving further detail to Kirk's book on optimal control [30].

First we introduce the notion of *costate variables*[2], $\mathbf{p}(t)$, that arise as Lagrange multipliers when reforming $\mathcal{J}$ into an augmented cost functional

$$\mathcal{J}_a(\mathbf{u}) = \int_{t_0}^{t_f} \Bigg\{ g(\mathbf{x}(t), \mathbf{u}(t), t) + \left[\frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}(t), t) \right]^{\mathrm{T}} \dot{\mathbf{x}}(t) + \frac{\partial h}{\partial t}(\mathbf{x}(t), t) + \mathbf{p}^{\mathrm{T}}(t) \left[\boldsymbol{f}(\mathbf{x}(t), \mathbf{u}(t), t) - \dot{\mathbf{x}}(t) \right] \Bigg\} dt .$$

Now we can define a useful function, $\mathscr{H}$, known as the *Hamiltonian*[30], which is given as

$$\mathscr{H}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}(t), t) = g(\mathbf{x}(t), \mathbf{u}(t), t) + \mathbf{p}^{\mathrm{T}}(t) \left[\boldsymbol{f}(\mathbf{x}(t), \mathbf{u}(t), t) \right]. \tag{2.11}$$

Now we have the required tools to state, without derivation, the necessary conditions for a solution to the optimal boundary value problem, Equation (2.10):

[2] not to be confused with the notation used for the attribute vector that appears in the machine learning portion of this work, Chapter 3

Necessary Conditions for OBVP Solution:

$$\begin{aligned}
\dot{\mathbf{x}}^*(t) &= \frac{\partial \mathscr{H}}{\partial \mathbf{p}}\left(\mathbf{x}^*(t), \mathbf{u}^*(t), \mathbf{p}^*(t), t\right) \\
\dot{\mathbf{p}}^*(t) &= -\frac{\partial \mathscr{H}}{\partial \mathbf{x}}\left(\mathbf{x}^*(t), \mathbf{u}^*(t), \mathbf{p}^*(t), t\right) \\
\mathscr{H}\left(\mathbf{x}^*(t), \mathbf{u}^*(t), \mathbf{p}^*(t), t\right) &\leq \mathscr{H}\left(\mathbf{x}^*(t), \mathbf{u}^*(t), \mathbf{p}(t), t\right)
\end{aligned} \tag{2.12}$$

for all $t \in [t_0, t_f]$ and the third line, known as *Pontryagin's Minimum Principle*, holds for all admissible $\mathbf{u}(t)$, i.e. $\mathbf{u}(t) \in \mathcal{U}$. Assuming a fixed final state, yet free final time, satisfaction of the boundary values are addressed by the condition

$$\mathscr{H}(\mathbf{x}^*(t_f), \mathbf{u}^*(t_f), \mathbf{p}^*(t_f), t_f) + \frac{\partial h}{\partial t}(\mathbf{x}^*(t_f), t_f) = 0. \tag{2.13}$$

It is important to note that the necessary conditions do not represent solutions to the optimal boundary value problem. For one they are necessary but often not sufficient. Furthermore they represent a set of partial differential equations that they themselves may require numerical treatment to solve. Instead the necessary conditions can be useful for verifying a proposed solution to a optimal control problem and developing numerical techniques for approximate solutions. For a small subset of problems, we can indeed use the necessary conditions to derive analytical solutions. We now discuss some numerical techniques and analytical results for the optimal boundary value problem that are relevant to this thesis.

2.3.2 Numerical Techniques

Many solution techniques exist for approximating the solution to Equation (2.10), a few of the most prominent approaches being: linearization of dynamics and solution via analytical results [31], shooting methods, direct trajectory optimization, and indirect trajectory optimization [32, 8]. 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 thesis. 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 [33]. This leaves direct trajectory optimization techniques as the favored choice for the presented work.

Approximate solutions to Equation (2.10) can be achieved 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). As previously noted, when solving Equation (2.10) we consider state and control constraints, yet ignore obstacle constraints which are checked a posteriori.

Pseudospectral Methods

The necessary conditions for a solution to Equation (2.10) are presented in Section 2.3.1, but even with these tools only simplified cases lend themselves to analytical solution [30]. In general, Equation (2.10) requires a numerical treatment. In this thesis we apply a solution technique that begins by time-discretizing Equation (2.10) 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 [29]. The result is a discrete problem of the form:

$$\begin{aligned}
\text{minimize: } & J^N\left[\mathbf{x}, \boldsymbol{U}, t_b\right] = M\left[\mathbf{x}_N, t_b\right] + \\
& \frac{t_b - t_a}{2} \sum_{k=0}^{N} L\left[\mathbf{x}_k, \boldsymbol{u}_k, t_k\right] w_k \\
\text{subject to: } & \text{for } k = 0, ..., N \\
& \boldsymbol{f}_l \leq \boldsymbol{f}\left[\frac{2}{t_b - t_a}\boldsymbol{d}_k, \mathbf{x}_k, \boldsymbol{u}_k, t_k\right] \leq \boldsymbol{f}_u, \\
& \boldsymbol{g}_l \leq \boldsymbol{g}\left[\mathbf{x}_k, \boldsymbol{u}_k, t_k\right] \leq \boldsymbol{g}_u, \\
& \boldsymbol{\phi}_l \leq \boldsymbol{\phi}\left[\mathbf{x}_0, \mathbf{x}_N, (t_b - t_a)\right] \leq \boldsymbol{\phi}_u
\end{aligned} \tag{2.14}$$

where $\mathbf{x} = [\mathbf{x}_0^{\mathrm{T}}, ..., \mathbf{x}_N^{\mathrm{T}}]^{\mathrm{T}}$ and $\boldsymbol{U} = [\boldsymbol{u}_0^{\mathrm{T}}, ..., \boldsymbol{u}_N^{\mathrm{T}}]^{\mathrm{T}}$.

Sequential Convex Programming

Now posed in the discrete form, a solution to Eqn. 2.14 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 [34, 8]. SQPs work by iteratively 'convexifying' Eqn. 2.14 around local trust-region, solving the convex problem, adjusting the trust-region, and repeating until some user-defined tolerances are achieved [34].

2.3.3 Dubins Vehicle

The Dubins vehicle introduced in Section 2.2.5 has a known analytical solution for the optimal path between any two states or configurations. It can be shown that any two states for a Dubins vehicle can be optimally connected using one of six trajectory types or *words*, each trajectory consisting of three maneuvers. The six possible optimal trajectories can be represented as

$$\{L_\alpha R_\beta L_\gamma,\, R_\alpha L_\beta R_\gamma,\, L_\alpha S_d L_\gamma,\, L_\alpha S_d R_\gamma,\, R_\alpha S_d L_\gamma,\, R_\alpha S_d R_\gamma\}\,, \tag{2.15}$$

where L represents a left-turning maneuver of minimum radius, R represents a right turning maneuver of minimum radius, and S represents a straight path. The angles $\alpha,\, \gamma \in [0,\, 2\pi)$ and $\beta, \in (\pi,\, 2\pi)$ represent the duration of the respective turn and $d \geq 0$ is the distance of the straight path [27]. Knowing that the optimal trajectory will be described by one of the paths in Equation (2.15) the task then becomes determining which curve with which parameters. A simple approach is to try all such trajectories with minimum radius turns and bitangent straight sections.

2.3.4 Control-Penalized Double Integrator

As explained in Section 2.2.4, we minimize computations by approximating our system as the double integrator given in Eqn (2.7). 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 5.1.4. The results in this section come from the works [20, 35].

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

$$\mathcal{J}[\mathbf{u}, \tau] = \int_0^\tau 1 + \mathbf{u}[t]^{\mathrm{T}} R_u \mathbf{u}[t] \, \mathrm{d}t, \tag{2.16}$$

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. (2.17) where $R_u = w_R I$ and w_R is the control penalty weight [20, 35]:

$$\mathcal{J}^*[\tau] = \tau + \|\mathbf{x} - \bar{\mathbf{x}}[\tau]\|^{\mathrm{T}} \mathbf{d}[\tau]. \tag{2.17}$$

The corresponding control and state trajectories as functions of time t, for a fixed final time τ, are given in Eqn. (2.18), respectively [20, 35]:

$$\begin{aligned} \mathbf{u}[t] &= R_u^{-1} B^{\mathrm{T}} \exp\left[A^{\mathrm{T}}(\tau - t)\right] \mathbf{d}[\tau], \\ \mathbf{x}[t] &= \bar{\mathbf{x}}[t] + G[t] \exp\left[A^{\mathrm{T}}(\tau - t)\right] \mathbf{d}[\tau], \end{aligned} \tag{2.18}$$

where

$$\begin{aligned} \mathbf{d}[\tau] &= G[\tau]^{-1} \left(\mathbf{x} - \bar{\mathbf{x}}[\tau]\right), \\ G[t] &= \frac{1}{w_R} \begin{bmatrix} t^3/3 & 0 & 0 & t^2/2 & 0 & 0 \\ 0 & t^3/3 & 0 & 0 & t^2/2 & 0 \\ 0 & 0 & t^3/3 & 0 & 0 & t^2/2 \\ t^2/2 & 0 & 0 & t & 0 & 0 \\ 0 & t^2/2 & 0 & 0 & t & 0 \\ 0 & 0 & t^2/2 & 0 & 0 & t \end{bmatrix}, \\ \bar{\mathbf{x}}[t] &= \exp\left[At\right] \mathbf{x}_0 + \left[0, 0, gt^2/2, 0, 0, gt\right]^{\mathrm{T}}. \end{aligned} \tag{2.19}$$

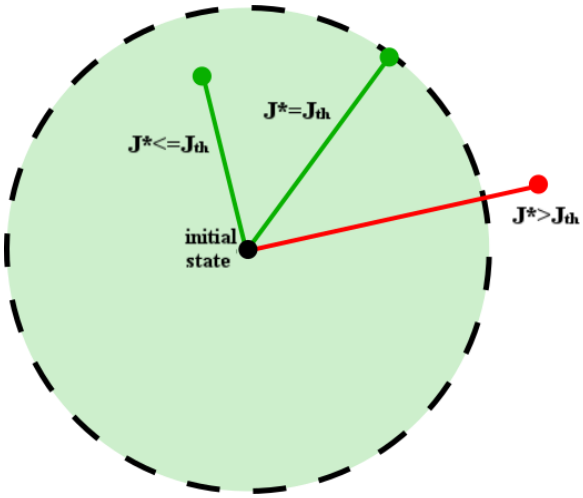


Figure 2.9: Cost-limited reachable set for a two-dimensional system with no differential constraints of the type that would appear in a geometric planning problem.

Note that Eqns. (2.17) and (2.18) require a *fixed* final time τ. The optimal final time, $\tau^* = \text{argmin}\, \mathcal{J}[\tau]$, can be solved for via a bisection search of Eqn. (2.17).

2.4 Reachable Sets for Dynamical Systems

As mentioned in Section 2.1 sampling-based motion planners rely on the concept of *neighborhoods* to efficiently generate a graph of the configuration space to solve a planning problem. For *geometric* planning problems, where the distance metric is Euclidean distance, a state neighborhood is an n-dimensional sphere. Mathematically the neighborhood, $\mathcal{R}$, of state $\mathbf{x}_a$ is given as

$$\mathcal{R}(\mathbf{x}_a) = \{\mathbf{x}_b \in \mathcal{X} \mid \|\mathbf{x}_a - \mathbf{x}_b\| \leq \mathcal{J}_{th}\} \tag{2.20}$$

where n is the dimensionality of the configuration space and $\mathcal{J}_{th}$ is a user-defined radius or "threshold". The cost-limited reachable set is illustrated in Figure 2.9.

For a differentially-constrained system, such as those discussed in Section 2.2 the cost-limited reachable set, or neighborhood, of state $\mathbf{x}_a$ may be considerably more complex. In fact there is no general analytical expressions for the cost-limited

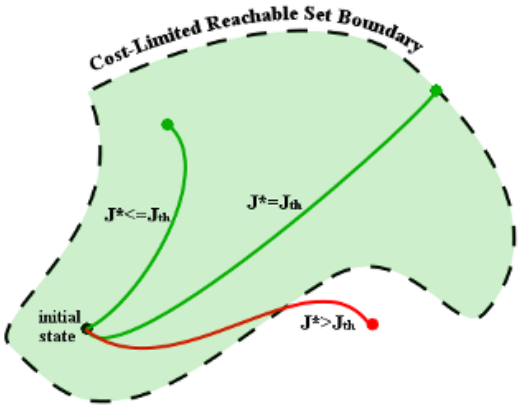


Figure 2.10: Conceptual representation of a cost-limited reachable set for a notional 2D dynamical system. Formally, a (forward) cost-limited reachable set is the set of states that can be reached from a given state with a cost bounded above by a given threshold (denoted as $\mathcal{J}_{\text{th}}$).

reachable sets for systems described by Equation (2.3).

While we may not be able to give an anayltical expression for an arbitrary reachable set, we may give a rigorous mathematical definition for a cost-limited reachable set. First we note that two types of reachable sets may be defined: the *forward* reachable set and the *backward* reachable set. The mathematical definition of the forward cost-limited reachable set, or "outgoing neighborhood", of a state $\mathbf{x}_a$ is:

$$\begin{aligned}\mathcal{R}^{\text{out}}\left(\mathbf{x}_a, \mathcal{U}, \mathcal{J}_{\text{th}}\right) := \{\mathbf{x}_b \in \mathcal{X} \mid \exists \mathbf{u} \in \mathcal{U} \text{ and } \\ \exists t' \in [t_0, t_f] \text{ s.t. } \mathbf{x}\left(t'\right) = \mathbf{x}_b \text{ and } \mathcal{J}^* \leq \mathcal{J}_{\text{th}}\},\end{aligned} \tag{2.21}$$

where $\mathcal{J}_{\text{th}}$ is a user-defined cost threshold. In words, Equation 2.21 states that the forward reachable set is the union of all states $\mathbf{x}_b \in \mathcal{X}$ such that the optimal cost, $\mathcal{J}^*$, to steer the system from $\mathbf{x}_a$ to $\mathbf{x}_b$ is less than the cost threshold $\mathcal{J}_{\text{th}}$. We can use Equation (2.21) to define the related concept of a backward reachable set or "incoming neighborhood". The backward reachable set of state $\mathbf{x}_b$ is the union of all states, $\mathbf{x}_a$, such that $\mathbf{x}_b$ is in the forward reachable set of $\mathbf{x}_a$. A conceptual diagram of a cost-limited reachable set, i.e. *neighborhood*, of a given state is represented in Fig. 2.10.

As previously noted, the determination of cost-limited reachable sets is a critical need for sampling-based motion planning, however this can be a computationally challenging task. The computational complexity of numerical approximations are exponential in the dimension of the configuration space [36]. In Chapter 3 we develop a unique application of machine learning for rapidly approximating Equation 2.21.

Chapter 3

Machine Learning for Real-Time Reachability Analysis

As was discussed in Section 2.4, the determination of reachability sets for dynamical systems is a computationally-expensive problem [36]. For problems where real-time analysis of reachable sets is required, such as for real-time sampling-based motion planning, we must devise a rapid approximation scheme. In this chapter[1], we show that supervised machine learning techniques [37] can be used to accurately classify – in a binary, query-based fashion – the cost-limited reachable sets of dynamical systems in real-time. To elaborate, we seek a function that makes a simple, binary discrimination:

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

To demonstrate, we use two different machine learning algorithms and compare the accuracy and efficiency of each. The algorithms employed are (1) a support vector machine learning algorithm for binary classification of a state's reachable set, and (2) a locally-weighted regression algorithm for predicting cost function values between query points $\mathbf{x}_a$ and $\mathbf{x}_b$, which determines if $\mathbf{x}_b$ is cost-limited reachable from $\mathbf{x}_a$. The

[1] the content of this chapter is the product of collaboration between the author, Ashley Clark, and Joseph Starek with guidance from Marco Pavone [2].

approach generalizes to any type of system dynamics so long as cost training data can be generated. In this chapter we restrict our attention to continuous, nonlinear, ordinary dynamical systems, and demonstrate our learning strategies on several systems of interest: the Dubins car model (see Section 2.2.5), a gravity-free spacecraft model (see Section 2.2.7), and the control-penalized double integrator (see Section 2.2.4). These techniques can asses a classification query on the order of milliseconds; a improvement of up to 4 orders-of-magnitude over an exact classification.

3.1 SVM Classification of Reachable Sets

To develop such a function, we must first generate training data in the form of a set of *training examples*. Since our goal is to predict whether two states are within some cost threshold of one another, our training data will be generated from solving a large set of optimal boundary value problems. A training example consists of a initial state $\mathbf{x}_a$, final state $\mathbf{x}_b$, 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, $\mathcal{J}_{\text{th}}$, 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. (3.1) [37]:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{N_{\text{train}}} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N_{\text{train}}} y^{(i)} y^{(j)} \alpha_i \alpha_j K\left(\mathbf{p}^{(i)}, \mathbf{p}^{(j)}\right) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \ i = 1, \ldots, N_{\text{train}} \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{3.1}$$

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

the *support vectors*. For this work the *kernel function*, K, has the form

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

where ϕ is a nonlinear mapping of the attribute vector to a *feature vector* (see Tables 3.1 and 3.2 for examples of feature vectors used), 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

$$\operatorname{sgn}\left(\sum_{i=1}^{N_{\text{train}}} \alpha_i y^{(i)} K\left(\mathbf{p}^{(i)}, \tilde{\mathbf{p}}\right) + b\right). \tag{3.2}$$

where b is a bias term that is determined as a function of the Lagrange multipliers [37].

3.2 Regression Estimation of Optimal Cost

For a given system (i.e. , fixed cost function, dynamics, control constraints and state constraints) each optimal boundary value problem in Equation (2.10) differs only in the initial and final conditions. As a result, the optimal cost, $\mathcal{J}^*$, can be thought of as a function of the boundary states, $\mathbf{x}_a$ and $\mathbf{x}_b$ (also referred to as a query pair), parameterized by system dynamics and constraints. Accordingly we write $\mathcal{J}^* = \mathcal{J}^*(\mathbf{x}_a, \mathbf{x}_b; f, \mathcal{X}, \mathcal{U})$. The idea then is interpolate neighboring pre-computed queries using regression techniques. Due to its robustness and simplicity, we demonstrate the idea using locally-weighted linear regression, implemented as:

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{m} w^{(i)} \left(\mathcal{J}^{*(i)} - \theta^{\mathrm{T}} \phi(\tilde{\mathbf{x}}^{(i)})\right)^2$$

where $\tilde{\mathbf{x}}^{(i)} \in \mathbb{R}^d$ is the i^{th} training example (query pair), m is the number of training examples, $\theta \in \mathbb{R}^n$ is the training parameter vector, $\phi : \mathbb{R}^d \to \mathbb{R}^n$ is the feature mapping, $\mathcal{J}^{*(i)}$ is the optimal cost of the i^{th} training query, and $w^{(i)}$ is the weight

given to the i^{th} residual. As it is well known, the solution is

$$\theta^* = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{W}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{W}\mathbf{J}^*,$$

which yields

$$\hat{\mathcal{J}} = \theta^{*\mathrm{T}}\phi(\tilde{\mathbf{x}})$$

as the estimated cost, where $\mathbf{W} \in \mathbb{R}^{k\times k}$ is a diagonal matrix of weights $w^{(i)}$, $\mathbf{J}^* \in \mathbb{R}^k$ is the vector of optimal costs, and $\mathbf{\Phi} \in \mathbb{R}^{k\times n}$ is the matrix of all k feature vectors.

Model Selection

To improve interpolation performance, feature vector components are normalized to unit scaling by defining the weights $w^{(i)}$ as:

$$w^{(i)} = \exp\left(\frac{\left|\left|\mathbf{v}^{(i)}\right|\right|^2}{2\tau^2}\right), \text{ with } \mathbf{v}_k^{(i)} = \frac{\phi\left(\tilde{\mathbf{x}}^{(i)}\right)_k - \phi\left(\tilde{\mathbf{x}}\right)_k}{\text{range}\left(\phi\left(\tilde{\mathbf{x}}\right)_k\right)},$$

where $\tau > 0$ is the bandwidth parameter, and $\text{range}\left(\phi\left(\tilde{\mathbf{x}}\right)_k\right)$ is the maximum extent of the kth feature ($k = 1, \ldots, n$). Due to the tradeoff between over- and under-fitting as τ is varied, we ran k-fold cross-validation [37, Ch. 1] to find the value of τ that yielded the lowest average percent error over all k training queries.

Feature Selection

As previously mentioned, OBVP query pairs are mapped to feature vectors. In order to more effectively approximate the true cost function, one should carefully choose a mapping that produces linearly-independent features that are relevant to the given OBVP (e.g. , endpoint norms, energy values, ratios of boundary states, etc). In an effort to identify features with the most significant impact on cost approximation accuracy, we ran a backward feature selection search as part of our cost-prediction algorithm training. This not only illustrates the trade-off in approximation error and feature vector size, but also provides feature vector design intuition for the particular application.

3.3 Numerical Test Cases

3.3.1 Dubins Vehicle

The cost-limited reachable set for a given Dubins car is known analytically [38], the boundary of which can be described by the intersection of two congruent cardiods. parameterized by:

$$x(\theta) = \rho\left(2\sin\theta - \sin(2\theta - vt/\rho)\right),$$
$$y(\theta) = -\rho\left(2\cos\theta - 1 - \cos(2\theta - vt/\rho)\right).$$

The cardiods translate and rotate depending upon the time horizon under consideration. For time horizons and rotation angles that satisfy the inequality $0 \leq \theta \leq vt/\rho$, the boundary can be simplified and described by:

$$\begin{aligned} x(\theta) &= \rho\sin\theta + (vt - \rho\theta)\cos\theta, \\ y(\theta) &= -\rho(1 - \cos\theta) + (vt - \rho\theta)\sin\theta. \end{aligned} \tag{3.3}$$

The projection of this simplified set into the x-y plane is depicted in Figure 3.1. While heading constraints are accounted for in the analytical solution, they are not displayed in the figure for the sake of simplicity. The dynamics of the Dubins car are invariant with respect to its initial state. Therefore, without loss of generality, we may assume the initial state $\mathbf{x}_a$ is always the origin, and set the feature vector to consist of combinations of the target state only.

The results of our training and feature selection for the regression-based approach, Section 3.2, are listed in Table 3.1 with the features listed in the order of relevance, from most relevant feature to least relevant feature. The corresponding average cost estimation error for each group of n most relevant features are shown in Figure 3.2. Results show that one can achieve 5.3% error[2] from the true cost with just twelve features. Note that, even though the learning algorithm does not have prior knowledge

[2]Here, error is defined as the difference between the true and predicted costs divided by the true cost averaged over all training examples.

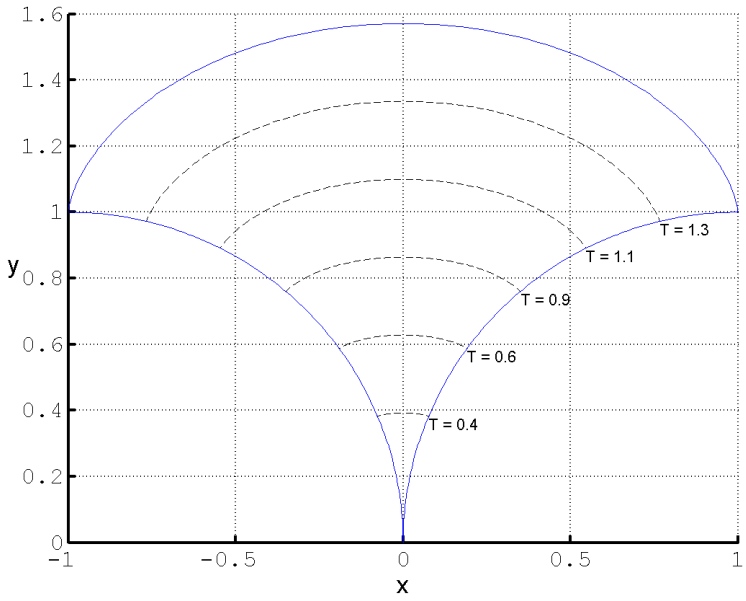


Figure 3.1: Reachability sets for an instance of the Dubins car vehicle ($\rho_{\min} = 1, v = 1, L = 1$) for cost threshold horizon times $T \leq \frac{\pi}{2}$.

of the analytical form of reachable set as given in Equation (3.3), the feature selection process gives high importance to the terms that appear in the analytical form. This is a crucial result as it implies that the machine learning approach is robust enough to generate accurate fits for unknown, nonlinear dynamics and cost functions.

Given the same OBVP training examples as for cost prediction, reachability was assessed using a nonlinear SVM classifier, Section 3.1, with a 4th-order polynomial kernel. The classification results on a test set of 100 new query points are shown in Figure 3.3. Several kernel functions were attempted and the 4-th order kernel proved to be the most accurate for this system. Test errors[3] were 4.17%, 7.29%, and 3.13% respectively for each of three cost thresholds used to establish the cost-reachable boundary, namely, $\mathcal{J}_1 = \mu - \sigma, \mathcal{J}_2 = \mu,$ and $\mathcal{J}_3 = \mu + \sigma$, where μ was the training set mean cost and σ the standard deviation.

Due to the complexity of displaying the final θ-heading values in addition to the x-y

[3]Here, test error is defined as the number of misclassifications divided by total number of test examples.

Table 3.1: Dubins car features from most important to least important.

1-12	13-24	25-36
$\cos\theta$	$y\cos\theta$	$1/\tan\theta$
$\sin^2\theta$	$y\theta$	$x/\tan\theta$
$\cos^2\theta$	θ	$1/\cos\theta$
$\lvert\theta\rvert$	$\theta\sqrt{x^2+y^2}$	$x/\cos\theta$
θ^2	$y\sin\theta$	$y/\tan\theta$
$\sqrt{x^2+y^2+\theta^2}$	$x\sin\theta$	$1/\sin\theta$
$\sqrt{x^2+y^2}$	$\lvert x\rvert$	$x/\sin\theta$
x	xy	$y/\sin\theta$
$x\cos\theta$	y^2	$y\tan\theta$
$\sin\theta$	x^2	$y/\cos\theta$
$x\theta$	$\lvert y\rvert$	$tan^2\theta$
y	$\tan\theta$	$x\tan\theta$

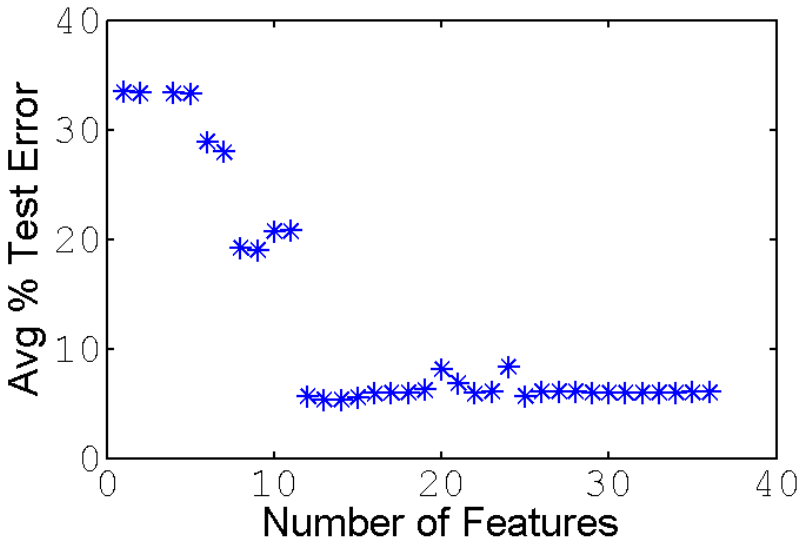


Figure 3.2: Results of feature selection for the Dubins car showing the average test error percentage vs number of features used.

position values, these results are not shown directly superimposed on the reachability set in Figure 3.1. Instead, Figure 3.3 plots the test classifications generated for both the linear regression and SVM approach with the cost threshold, $\mathcal{J}_{\text{th}}$, shown as a solid black line. It can be seen that most misclassifications occur in the direct cost-vicinity of the cost threshold. In simple terms this means that even when the algorithms misclassify points, they aren't too far from the actual cost.

One noticeable outlier occurs for the lowest threshold data where the SVM misclassifies a high-cost test case. This misclassification occurred because the final heading happens to point back toward the initial state for this testing example. Note that feature selection revealed a high importance of $|\theta|$ and θ^2. Since $-\pi \leq \theta \leq \pi$, the anti-parallel alignment can spoof the SVM into thinking it's nearly a straight-line path to the end point when, in fact, it requires almost an entire loop to reach the final state. Altering the feature vector eliminates this outlier.

3.3.2 Gravity-Free Spacecraft

The gravity-free spacecraft model presented in Section 2.2.7 was used as a second test case for the machine learning of reachable sets. The features examined for the spacecraft are shown in Table 3.2, listed in order of importance as determined by our backwards-search feature selection algorithm. The average weighted-linear regression estimation error obtained from increasingly large sets of the top-priority features is depicted in Figure 3.4. Interestingly, it appears that only the top 5 features out of the original 26 in our feature vector are required for the average cost-estimation error to fall below 10%, for an unseen OBVP problem.

For the full feature set, the SVM reachability analysis was again implemented with a 4th-order polynomial kernel, similar to the Dubins car. In both cases, the 4th-order polynomial kernel seemed to balance the trade-off between bias and variance for the classifier. The classification of a 100-point test set of new 2PBVP query points can be seen in Figure 3.5, yielding test errors 8.08%, 11.11%, and 7.07%, respectively, again for cost thresholds $\mathcal{J}_1 = \mu - \sigma, \mathcal{J}_2 = \mu,$ and $\mathcal{J}_3 = \mu + \sigma$. Figure 3.5 directly compares the classifications made be the linear regression and SVM algorithms. Again, most

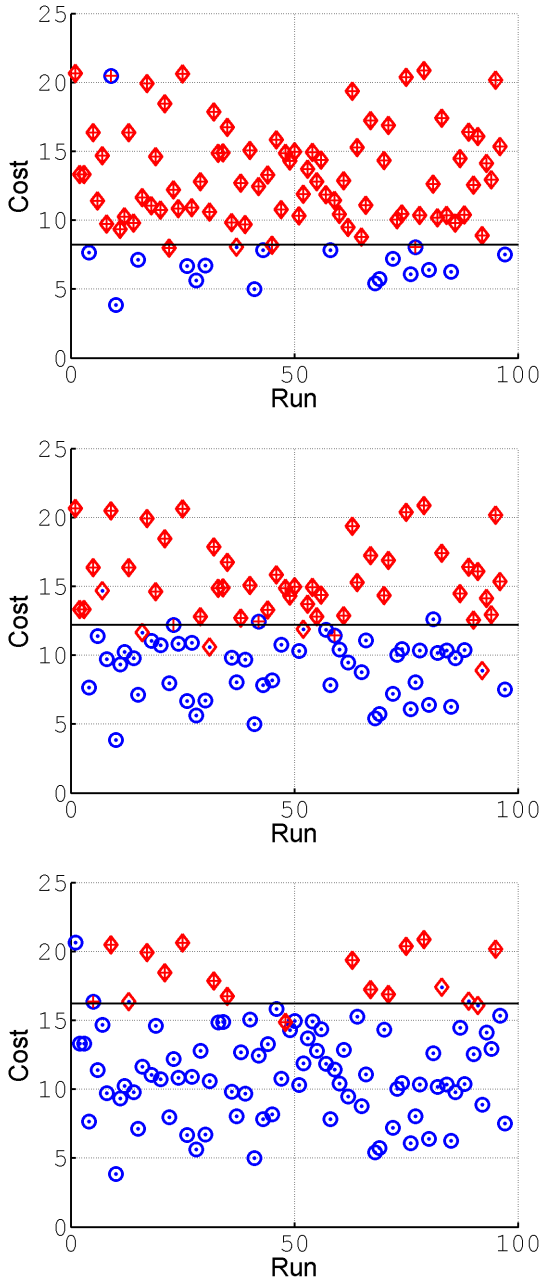


Figure 3.3: Predicted reachability set plotted with true cost for the Dubins Car with 3 different cost thresholds (black lines). Blue circles = SVM predicted reachable, red diamonds = SVM predicted non-reachable, blue dot = linear regression predicted reachable, red cross = linear regression predicted non-reachable.

Table 3.2: Deep space spacecraft features, from most important to least important

1-9	10-18	19-26
$(x/\dot{x})^2$	y^2	$\dot{z}^2$
$(y/\dot{y})^2$	$\|\|x,y,z\|\| \,/\, \|\|\dot{x},\dot{y},\dot{z}\|\|$	$\dot{y}^2$
$(z/\dot{z})^2$	z^2	$\|\|\dot{x},\dot{y},\dot{z}\|\|$
$\|\|x/\dot{x},y/\dot{y},z/\dot{z}\|\|$	y	$\dot{x}$
$x/\dot{x}$	$\|\dot{z}\|$	$\dot{y}$
$y/\dot{y}$	x	$\dot{z}$
$z/\dot{z}$	$\|\dot{x}\|$	$\|\|x,y,z\|\|$
z	$\|\dot{y}\|$	$\|\|x,y,z,\dot{x},\dot{y},\dot{z}\|\|$
x^2	$\dot{x}^2$	

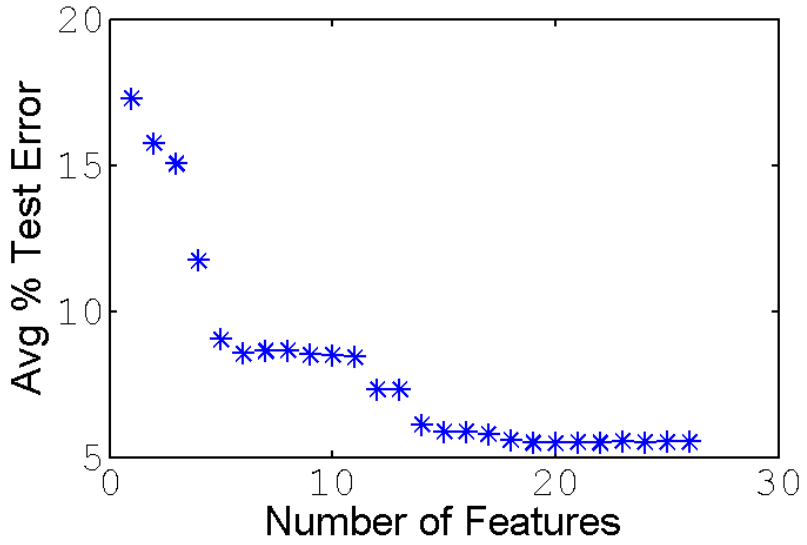


Figure 3.4: Results of feature selection for the deep-space spacecraft showing the average test error percentage vs number of features used.

Table 3.3: Average computation time and percent of misclassification for the two-point boundary value problem solver, the linear regression cost estimation (best fit model using all features), and SVM classification (average over all 3 cost thresholds).

System	2PBVP Solve		Lin. Reg.		SVM	
	Time	% Err	Time	% Err	Time	% Err
Dubins	1.23 s	0.0	9.4 ms	3.8	0.44 ms	4.9
Spacecraft	10.3 s	0.0	9.0 ms	5.8	0.40 ms	8.8

misclassifications occur near the cost threshold.

3.3.3 Control-Penalized Double Integrator

The control-penalized double integrator system discussed in Section 2.2.4 was also used as a test case for the machine learning of reachable sets. Due to the relevance to quadrotor planning, results and discussion of the machine learning of reahable sets for the double integrator system are left to Section 5.3.

3.3.4 Execution Time and Accuracy

Table 3.3 compares the computation time and classification accuracy of each of the three approaches: 1) truth-value determined by a 2PBVP solver[4], 2) locally-weighted linear regression cost estimation, and 3) support vector machine classification. These results confirm that machine learning techniques are able to drastically reduce the computation time for reachability analysis by as much as four orders of magnitude – which comes at the cost of misclassifying some queries.

By definition, the truth-value calculation produces no misclassifications but take seconds or tens-of-seconds to perform each classification. Locally-weighted linear regression produces the most accurate classifications of the machine learning techniques but can take 20x longer per query to compute when compared to the SVM approach. The relatively large computation time for linear regression is due to the $n \times n$ matrix

[4]As previously mentioned, the techniques used to determine the true classifications are not guaranteed to be absolutely correct, but they do represent the state of the art in solving optimal 2PBVPs. See Section 2.3.2 for more details.

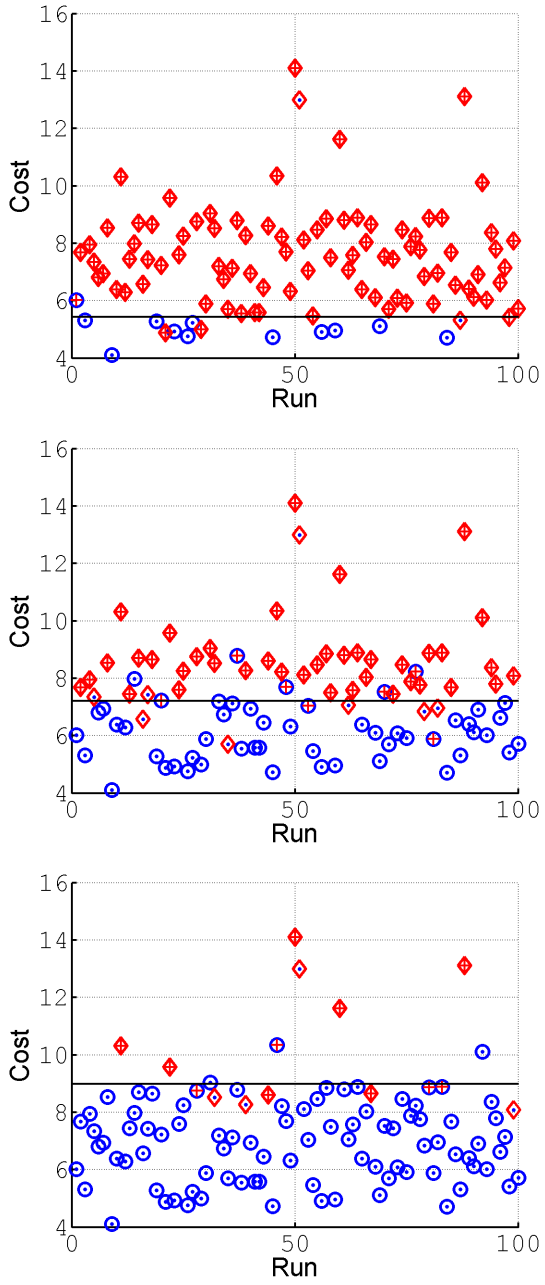


Figure 3.5: Predicted reachability set plotted with true cost for the deep-space spacecraft with 3 different cost thresholds (black lines). Blue circles = SVM predicted reachable, red diamonds = SVM predicted non-reachable, blue dot = linear regression predicted reachable, red cross = linear regression predicted non-reachable

inversion required for each classification query. This matrix inversion can be avoided if a non-locally-weighted regression scheme is used, but accuracy of classification is severely impacted (results are omitted due to page limitations). Linear regression techniques have the benefit of generating more information by providing an estimate of the optimal cost. The SVM only returns a binary, true-false result. This cost estimation may be valuable beyond reachable set analysis, depending on application.

Chapter 4

A Real-Time Framework for Kinodynamic Planning

In Chapter 1 we introduced the Optimal Kinodynamic Planning Problem in Equation (1.1). In this chapter we develop a technique for addressing such problems in real-time environments.

4.1 Real-Time Framework for Kinodynamic Planning

To solve the Optimal Kinodynamic 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. 4.1. Section 4.1.1 discusses how a priori information, such as system dynamics and control constraints, can be leveraged to precompute a large number of boundary value problems which are then stored in a look-up table for rapid queries during online computations. As discussed in Section 4.1.2, information that only becomes available at the initiation of online execution – such as obstacle

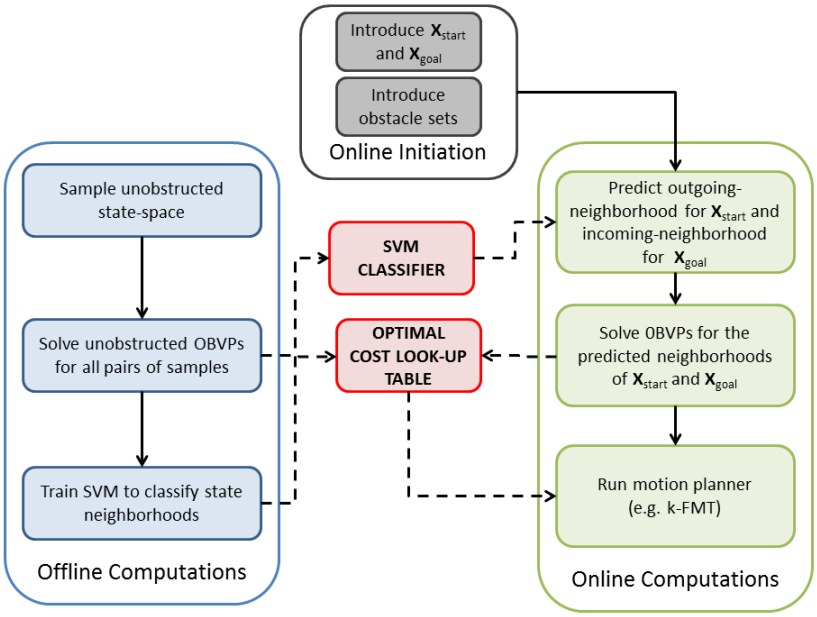


Figure 4.1: Flowchart of the Kinodynamic Motion Planning Framework. This diagram also illustrates the extension of prior work where the SVM Classifier and Motion Planner blocks correlate to [2] and [3], respectively.

data and terminal states – can then be fused with the precomputed look-up table using machine learning. Section 4.1.3 discusses the subroutines of this framework such as the solutions to boundary value problems, machine learning algorithms, and sampling-based planning scheme.

It is worth noting here that the framework does not have any component or subroutine dedicated to the problems of sensing, localization, or mapping. This is due to the fact that this thesis is wholly focused on planning. This is not meant to diminish the importance or difficulty of estimation, which is a fundamental challenge for robotics and control, but instead to isolate the two problems of planning and estimations. Chapter 5 discusses how the challenges of estimation are addressed for a physical robot using a Vicon motion capture system.

4.1.1 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, i.e. the configuration space is assumed to be completely free of obstacles during the offline phase. 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_s (with replacement and $N_{\text{pair}} \leq N_{\text{sample}}^2$) 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)[1], ignoring obstacles which are unknown, 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` represents a naive roadmap of the configuration space that is ignorant of obstacle locations. `Cost` may also 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 general 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 [2]. The training is accomplished with the subroutine `TrainClassifier` where $\mathcal{J}_{\text{th}}$ 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

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

[1] Also known as optimal control problems or steering problems, see Section 2.3.

Algorithm 1 Offline Computations for the Kinodynamic Motion Planning Framework

$V_s \leftarrow \texttt{Sample}(\mathcal{X}, N_{\text{sample}})$
$A \leftarrow \texttt{SampleData}(V_s, N_{\text{pair}}, \text{replace})$
$B \leftarrow \texttt{SampleData}(V_s, N_{\text{pair}}, \text{replace})$
$\texttt{Cost} \leftarrow \texttt{SolveOBVP}(A, B)$
$\texttt{Near} \leftarrow \texttt{TrainClassifier}([A, B], \texttt{Cost}(A, B), \mathcal{J}_{\text{th}})$

4.1.2 Online Computations

Upon initiation of online computation the algorithm is presented with the start state, $\mathbf{x}_{\text{init}}$, 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[2]. By applying the trained SVM, the algorithm proceeds to identify previously sampled states that are in the cost-limited, outgoing-neighborhood of $\mathbf{x}_{\text{init}}$ and the incoming-neighborhood of $\mathcal{X}_{\text{goal}}$. For each state classified in the neighborhoods of $\mathbf{x}_{\text{init}}$ 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 $\mathbf{x}_{\text{init}}$ 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.

Algorithm 2 Online Computations for the Kinodynamic Motion Planning Framework

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

[2]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.

4.1.3 Framework Subroutines

State Connections: `SolveOBVP`

The objective of the `SolveOBVP` subroutine is to produce solutions, whether exact or approximate, to the optimal boundary value problem given in Equation (2.10). The real-time kinodynamic planning framework is meant to be applicable to an general robotic system. Therefore we cannot assume an analytical, exact solution exists for our chosen robotic system and we must develop a subroutine that can give numerical approximations to a general OBVP. This can be accomplished by implementing the techniques discussed in Section 2.3.2. To avoid redundancy, please refer to the relevant section of Chapter 2 for further discussion.

Neighbor Estimation: `TrainClassifier` **and** `Near`

When the boundary states, $\mathbf{x}_{\text{init}}$ 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 as discussed in Section 2.4 and illustrated in Figure 2.10). 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.*

The estimation of neighborhoods is accomplished with the subroutine `Near`. This machine learning algorithm is trained during the offline phase using the subroutine `TrainClassifier`. The training and execution of these subroutines is the discussed in Chapter 3.

To contrast this approach with that of prior literature, Leven and Hutchinson's used a set of norms in the work space and configuration space as a rough surrogate for their desired distance metric of swept volume [10]. If our machine learning approach

were applied to the work in [10], swept volume reachable sets could be directly approximated instead of having to devise a surrogate function. This allows the flexibility in our framework to be applied to a more general set of planning problems.

Note that `Near` is trained on data in `Cost` which is generated with no knowledge of obstacle placement. Therefore, `Near` has no function in predicting obstacle collisions. Collision checking is solely within the realm of the sampling-based planner discussed in Section 4.1.4. Results on training and testing of the SVM classifier for a quadrotor system are presented in Section 3.3.

Sampling-Based Planner: *kino*-FMT

Along with the `SolveOBVP`, `TrainClassifier`, and `Near`, subroutines, the sampling-based motion planner, *kino*-`FMT`, acts as a subcomponent of the real-time planning framework presented in Figure 4.1. We dedicate a separate section to the discussion of the *kino*-FMT algorithm; refer to Section 4.1.4.

4.1.4 Kinodynamic Fast Marching Tree

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

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_s is stored in the discrete set N_z^{out}. The intersection of N_z^{out} and set W_s 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_s and the backward reachable set of x, N_x^{in}. The sample $y_{\min} \in Y_{\text{near}}$ represents the optimal connection point (assuming no obstacles) between x and the

existing tree. If the connection from $y_{\min}$ to x is free of collisions with obstacles, then the $(y_{\min}, x)$ edge is added to the tree, x is added to the frontier set H_s and removed from W_s. 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 $\mathbf{x}_{\text{init}}$ to $\mathcal{X}_{\text{goal}}$ as soon as the current pivot, z, is an element of $\mathcal{X}_{\text{goal}}$. If the frontier set H_s ever becomes empty, then *kino*-FMT reports failure. The (asymptotic) optimality properties of FMT* (and its kinodynamic variants) are discussed in [3, 39, 40].

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

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

4.2 Numerical Experiments

The real-time framework for kinodynamic planning was tested on two simulated dynamical systems: a fixed-wing UAV (Section 2.2.6) and a gravity-free spacecraft (Section 2.2.7). For both the UAV and Spacecraft problems, the real-time kinodynamic

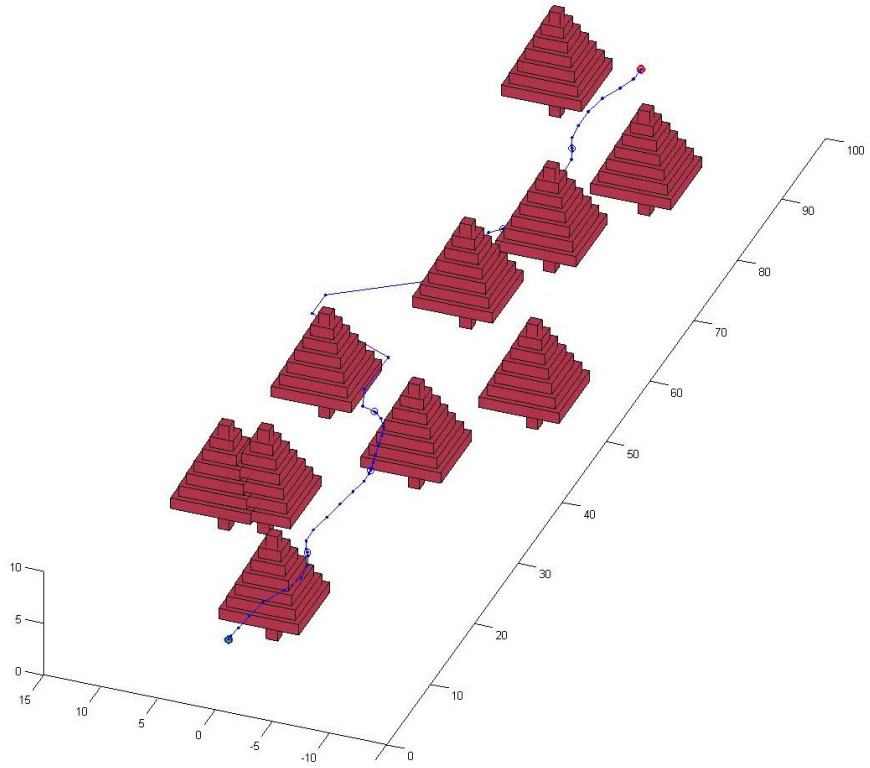


Figure 4.2: Time-optimized path for fixed-wing UAV navigating through a forest. Sharp corners are artifacts of plotting, not true representations of the trajectory.

motion planning framework demonstrates orders-of-magnitude decrease in computation time when compared to naive approaches. Table 4.1 summarizes the computation times for differing levels of complexity of the framework. Both problems where run with $N_{\text{sample}} = 5000$. 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_{sample} sampled states; involving N_{sample}^2 edge connections[3]. 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_{sample}^2 OBVPs to establish neighborhoods. The Neighborhood Learning framework does not use

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

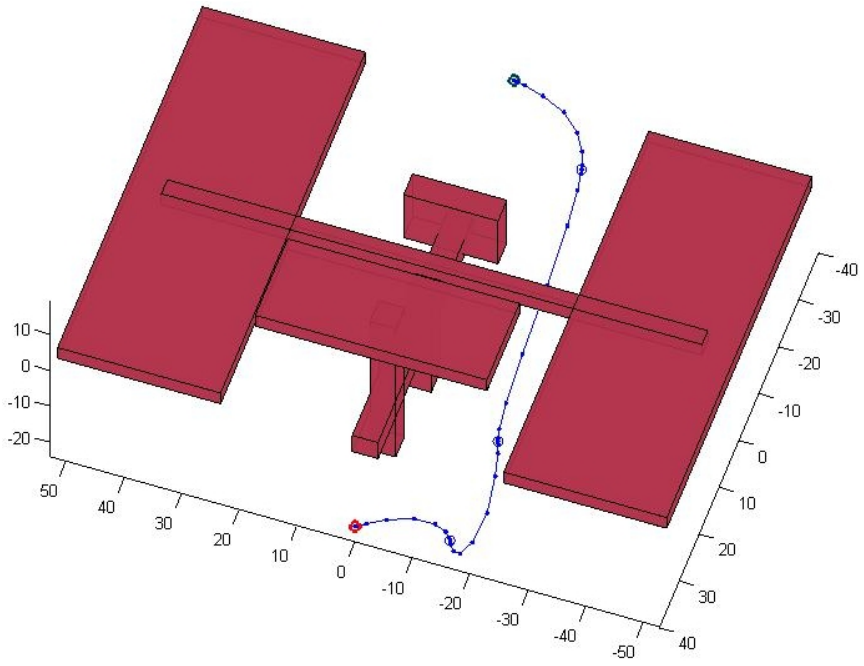


Figure 4.3: Time-optimized path for spacecraft navigating around ISS.

Table 4.1: 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

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 Section 4.1, reduces computation further by precomputing most of the OBVPs in the offline phase.

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 $\mathbf{x}_{\text{init}}$ and its out-neighborhood and $\mathcal{X}_{\text{goal}}$ its

Table 4.2: 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%

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 thesis.

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 4.2.

4.2.1 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 [41] and RRT* [42], 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 [3] and is not relevant to to evaluating the framework as a whole.

4.2.2 Notes on Intermediate Results

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 $\mathbf{x}_{\text{init}}$.

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 general 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 [43].

The implementation of these improvements and tailoring the framework to a specific system is presented in Chapter 5.

Chapter 5

Real-Time Quadrotor Planning and Control

In this chapter we build upon the results of Chapter 4 to demonstrate real-time planning for a physical, quadrotor system.

5.1 Real-Time Framework for Quadrotor Planning

As discussed in Section 2.1, sampling-based planning algorithms have become the accepted approach for planning in high-dimensional spaces where state (obstacle) constraints are only implicitly represented [1]. A critical feature of sampling-based algorithms is that they avoid the explicit construction of the configuration space (which can be prohibitive in complex planning problems as discussed in Section 2.1) 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" [1]. In this way, a complex trajectory control problem is broken down into a series of many smaller, simpler optimal boundary value problems (OBVP) [1] that are

[1]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 [44]. These techniques are prone to "wander" through the state space, lacking the optimality guarantees of

Figure 5.1: The real-time framework for kinodynamic planning and control for a quadrotor system.

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_{sample} is the number of sampled states. This is prohibitively expensive [46].

To address this we wrap a sampling-based planner in a real-time framework, given in Fig. 5.1, that minimizes the number of OBVPs that need to be solved online. This framework is an extension of that given in Chapter 4 that is designed to be implemented on a physical system. The broad structure of our framework, featuring an offline-online computation paradigm, has similarities to that presented by Leven and

algorithms such as RRT*, PRM*, and FMT* [14, 3]. 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 [45].

Hutchinson [10]. However the details of framework's construction and subcomponents, which are considered a novel contribution of this work, differ significantly from that of Leven's.

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

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

Here the framework developed in Chapter 4 (originally proposed in [23] and further expanded in [24]) is revisited in order to tailor it to a physical system. During execution of the real-time framework, computations are split into offline (Algorithm 4) and online (Algorithm 5) phases. During the offline phase – which is presented in Section 4.1 but repeated here for completeness – the subroutine `Sample` quasi-randomly draws N_{sample} 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 $N_{\text{pair}} \leq N_{\text{sample}}\,(N_{\text{sample}} - 1)$– from the discrete set of sampled states V_s, and stores them in two sets A and B. The N_{pair} samples stored in A and B are then paired and OBVPs are solved for each pair; storing the solutions for use during the online phase in a look-up table titled `Cost`. The OBVP solution subroutine, `SolveOBVP`, which is often referred to as a "steering function" in the motion planning literature, is given in Section 2.3.4. The look-up table `Cost` can equivalently be thought of as a precomputed, unobstructed roadmap (i.e. it is wholly ignorant of obstacle information which is not available until online initiation) through the state space. During the offline phase, a support vector machine (SVM) classifier, referred to as `Near`, is trained using the look-up table `Cost`. The SVM provides query-based estimates of cost-limited reachable sets (i.e. , *neighborhoods*) and is discussed in further details in Section 3.1. The cost threshold of the reachable set, often referred to a "neighborhood radius" in the motion planning literature, is the user defined value $\mathcal{J}_{\text{th}}$.

The online phase of computation is given in Algorithm 5; note the subtle differences from Algorithm 2 to account for the tailoring to a specific robotic system. At the initiation of the online phase, obstacle data is presented along with the start

Algorithm 4 Offline Phase for the Kinodynamic Motion Planning Framework

1 $V_s \leftarrow \texttt{Sample}(\mathcal{X}, N_s)$
2 $A \leftarrow \texttt{SampleData}(V_s, N_{\text{pair}}, \text{replace})$
3 $B \leftarrow \texttt{SampleData}(V_s, N_{\text{pair}}, \text{replace})$
4 $\texttt{Cost} \leftarrow \texttt{SolveOBVP}(A, B)$
5 $\texttt{Near} \leftarrow \texttt{TrainClassifier}([A, B], \texttt{Cost}(A, B), \mathcal{J}_{\text{th}})$

state, $\mathbf{x}_{\text{init}}$, and goal region, $\mathcal{X}_{\text{goal}}$[2]. 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 $\mathbf{x}_{\text{init}}$ and the incoming neighborhood of $\mathcal{X}_{\text{goal}}$ among the pre-sampled states; storing the sets in $N_{\text{init}}^{\text{out}}$ and $N_{\text{goal}}^{\text{in}}$, respectively (see Section 2.4 for discussion on outgoing and incoming neighborhoods). OBVPs are then solved from $\mathbf{x}_{\text{init}}$ 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_s, using the look-up table, or "roadmap", `Cost`. Though many candidate sampling-based planners could be used to compute a trajectory across this roadmap, we rely on the asymptotically-optimal FMT* algorithm for its efficiency (see [3] for a detailed discussion of the advantages of FMT* over state-of-the-art counterparts; see and [40] for its kinodynamic extension). The Kinodynamic Fast Marching Tree algorithm (*kino*-FMT) (adapted from [40]) leverages the roadmap to efficiently determine the optimal sequence of sampled states to connect $\mathbf{x}_{\text{init}}$ and $\mathcal{X}_{\text{goal}}$, performing collision checking in real-time (see Section 4.1.4).

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 5.1.3). Mapping the differentially flat output variables from the smooth trajectory back to the full state and control space (Section 5.1.4), we can provide feedforward terms to the flight controller (Section 5.1.5).

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

Algorithm 5 Online Phase for the Kinodynamic Motion Planning Framework

$X_{\text{goal}} \leftarrow \texttt{Sample}(\mathcal{X}_{\text{goal}}, N_{\text{goal}})$
$N_{\text{init}}^{\text{out}} \leftarrow \texttt{Near}(x_{\text{init}}, V_s \backslash \{x_{\text{init}}\}, \mathcal{J}_{\text{th}})$
$N_{\text{goal}}^{\text{in}} \leftarrow \texttt{Near}(V_s \backslash \{X_{\text{goal}}\}, X_{\text{goal}}, \mathcal{J}_{\text{th}})$
for $x \in V_s$ **do**
  if $x \in N_{\text{init}}^{\text{out}}$ **then**
    $\texttt{Cost} \leftarrow \texttt{SolveOBVP}(x_{\text{init}}, x)$
  if $x \in N_{\text{goal}}^{\text{in}}$ **then**
    $\texttt{Cost} \leftarrow \texttt{SolveOBVP}(x, X_{\text{goal}})$
$\texttt{Path} \leftarrow \textit{kino}\text{-}\texttt{FMT}\,(V_s, \texttt{Cost}, \mathbf{x}_{\text{init}}, X_{\text{goal}})$
return $\texttt{SmoothPath}(\texttt{Path})$

To handle dynamic obstacles we must develop a replanning structure that recomputes the kinodynamic motion plan as the environment evolves. We choose to implement an event-based replanner where the existing solution trajectory is continuously checked for collisions with obstacles and replanning is only initiated once the existing plan becomes obstructed. This replanning structure is represented in Figure 5.2. This event-based replanning is in contrast to a purely time-based, receding horizon replanner more typical for model predictive control. The event-based structure minimizes the number of replanning events which is desirable due to the "chaotic" nature of transitioning from one solution trajectory to another; see Section 5.4.2 for more discussion.

To further reduce the number of replanning events and provide more "graceful" behavior in proximity to dynamic obstacles we also implement a locally reactive controller. This controller is inspired by the concept of potential fields where nearby obstacles impose a virtual, repelling force on the autonomous system [47]. This reactive controller is represented in Figure 5.2 and is discussed further in Section 5.1.5. It is important to note this locally reactive controller is not necessary for the fundamental objective of real-time planning in dynamic environments which is achieved solely based on computation times of the real-time framework. In experimentation, however, it was shown to greatly improve performance, therefore it is discussed in this thesis.

We now present the mathematical details for each of the framework components

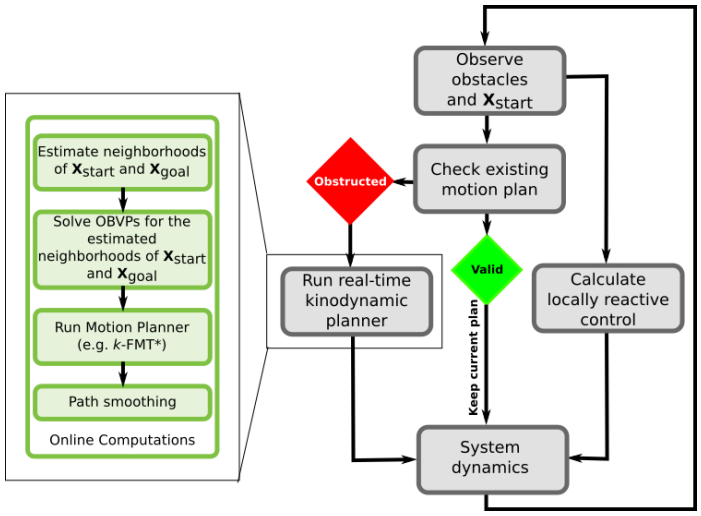


Figure 5.2: Event-based replanning structure used to account dynamic obstacles.

(to make the thesis self-contained, we also state a number of results already available in the literature).

5.1.1 Analytical Solution to OBVPs

In contrast to other works such as Leven and Hutchinson [10] and Richter, Bry, and Roy [11] that use straight line connections between states, we require the solution to an optimal boundary value problem to connect sampled states. As explained in Section 2.2.4, we minimize computations by approximating our system as the double integrator given in Eqn (2.7). 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 5.1.4. The results in this section come from the works [20, 35].

The mathematical details for solving OBVPs for double integrator systems are given in Section 2.3.4.

5.1.2 Machine Learning of Neighborhoods

As was discussed in Section 4.1.3, the boundary states, $\mathbf{x}_{\text{init}}$ and $\mathcal{X}_{\text{goal}}$, must be connected to the pre-sampled states in `Cost` before the *kino*-FMT motion planner can execute. This accomplished by training a machine learning algorithm for reachability estimation. The details of this estimation have already been discussed in Section 4.1.3 and Chapter 3, however it is again noted that the reduction in OBVPs to be solved for neighborhood connection of $\mathbf{x}_{\text{init}}$ and $\mathcal{X}_{\text{goal}}$ lies at the core of achieving real-time execution.

5.1.3 Snap Minimization for Trajectory Smoothing

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 [13], Richter et. al. [11] 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. (5.1). 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 [11, 12].

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 4.1.4. While an infinite number of splines may exist that satisfy these conditions, we seek the spline that minimizes the integral of the square of the snap. Let us begin by considering a single polynomial $P(t) = \sum_{n=0}^{N} p_n t^n$. The minimum-snap cost function for a single polynomial is defined as

$$J_{\text{snap}} = \int_0^T P^{(4)}(t)^2 \, dt = \boldsymbol{p}^{\mathrm{T}} Q(T) \boldsymbol{p}, \tag{5.1}$$

where $Q(T)$ is the Hessian matrix of J_{snap} with respect to the polynomial coefficients, $\boldsymbol{p}$ is a vector of the $N+1$ polynomial coefficients, and T is the polynomial segment time which is determined by the kinodynamic planner. The superscript (4) implies the 4th derivative of the polynomial. Without derivation, the Hessian matrix is given as[3]

$$\begin{aligned} Q_{i,j}(T) &= 2\left(\prod_{k=0}^{3}(i-k)(j-k)\right)\frac{T^{i+j-7}}{i+j-7} \quad \text{for: } i \geq 4 \wedge j \geq 4, \\ Q_{i,j}(T) &= 0 \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{otherwise.} \end{aligned} \tag{5.2}$$

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 4.1.4. 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

$$A\boldsymbol{p} = \boldsymbol{d} \tag{5.3}$$

$$A = \begin{bmatrix} A_0 \\ A_T \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} \boldsymbol{d}_0 \\ \boldsymbol{d}_T \end{bmatrix} \tag{5.4}$$

where the terms are given as

$$A_{0_{i,j}} = \begin{cases} \prod_{k=0}^{i-1}(i-k) & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{5.5}$$

$$\boldsymbol{d}_{0_i} = P^{(i)}(0) \tag{5.6}$$

[3]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 [11, 12]. In contrast, our work determines the polynomial segment times during the time-minimizing kinodynamic planning; see Section 4.1.4.

$$A_{T_{i,j}} = \begin{cases} \left(\prod_{k=0}^{i-1}(i-k)\right) T^{i-j} & \text{if } i \geq j \\ 0 & \text{if } i < j \end{cases} \tag{5.7}$$

$$\boldsymbol{d}_{T_i} = P^{(i)}(T) \tag{5.8}$$

Numerical stability can be achieved by reformulating the constrained problem represented in Eqns. (5.1) and (5.4) as an unconstrained optimization [11, 12]. This is achieved by optimizing over the polynomial derivatives at the terminal points instead of the polynomial coefficients. Under this reformulation, Eqns. (5.1) and (5.4) become

$$J_{\text{snap}} = \boldsymbol{d}^{\text{T}} A^{-\text{T}} Q(T) A^{-1} \boldsymbol{d}, \tag{5.9}$$

and the polynomial coefficients are determined, a posteriori, via inversion of Eqn. (5.3).

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

$$\boldsymbol{d}_{\text{total}} = \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}. \tag{5.10}$$

With this reordering of the derivative vector in Eqn. (5.10), 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

$$J_{\text{snap}} = \boldsymbol{d}_{\text{total}}^{\text{T}} C A_{1\ldots M}^{-\text{T}} Q_{1\ldots M} A_{1\ldots M} C^{\text{T}} \boldsymbol{d}_{\text{total}}. \tag{5.11}$$

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. (5.11) can be written

$$J_{\text{snap}} = \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_{\text{fix}} \\ \boldsymbol{d}_{\text{free}} \end{bmatrix}. \tag{5.12}$$

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

$$\boldsymbol{d}_{\text{free}}^{*} = -H_{22}^{-1} H_{12}^{\mathrm{T}} \boldsymbol{d}_{\text{fix}}. \tag{5.13}$$

Now that the derivatives at each waypoint are determined, the polynomial coefficients can be determined by inverting Eqn. (5.3). 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 5.1.4.

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 [11, 12].

5.1.4 Differentially Flat Mapping

The trajectory smoother from Section 5.1.3 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. (2.6) represents a differentially flat system with flat output variables $\vec{\xi}_N$ and ψ_N

[13]. This mathematical property proves that the smoothed trajectory from Section 5.1.3 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 [13], 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

$$u_{1_{ff}} = -\vec{z}_B \cdot \vec{F}_N, \quad \text{where: } \vec{F}_N = m\ddot{\vec{\xi}}_N - mg\vec{z}_W \tag{5.14}$$

The subscript ff indicate that this thrust value appears as a feedforward term in the flight controller (Section 5.1.5). The nominal orientation matrix is given by the nominal frame axes represented in world coordinates:

$$\vec{R}_N = \left[{}^W\vec{x}_N,\, {}^W\vec{y}_N,\, {}^W\vec{z}_N\right], \tag{5.15}$$

where

$$\begin{aligned} \vec{z}_N &= -\frac{\vec{F}_N}{\|\vec{F}_N\|} \\ \vec{y}_S &= [-\sin\psi_N, \cos\psi_N, 0]^{\mathrm{T}} \\ \vec{x}_N &= \frac{\vec{y}_S \times \vec{z}_N}{\|\vec{y}_S \times \vec{z}_N\|} \\ \vec{y}_N &= \vec{z}_N \times \vec{x}_N. \end{aligned} \tag{5.16}$$

The nominal angular velocity vector is given by

$$\vec{\Omega}_{NW} = p_N\vec{x}_N + q_N\vec{y}_N + r_N\vec{z}_N \tag{5.17}$$

where the individual components of nominal angular velocity are

$$\begin{aligned} p_N &= -\vec{h}_\Omega \cdot \vec{y}_N \\ q_N &= \vec{h}_\Omega \cdot \vec{x}_N \\ r_N &= \dot{\psi}_N \vec{z}_W \cdot \vec{z}_N \end{aligned} \tag{5.18}$$

For compactness we have defined

$$\vec{h}_\Omega = \frac{m}{u_{1_{ff}}} \left(\left(\vec{\xi}_N^{(3)} \cdot \vec{z}_N \right) \vec{z}_N - \vec{\xi}_N^{(3)} \right) \tag{5.19}$$

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

$$\dot{\vec{\Omega}}_{NW} = \alpha_{1_N} \vec{x}_N + \alpha_{2_N} \vec{y}_N + \alpha_{3_N} \vec{z}_N \tag{5.20}$$

where the individual components of nominal angular acceleration are

$$\begin{aligned} \alpha_{1_N} &= -\vec{h}_\alpha \cdot \vec{y}_N \\ \alpha_{2_N} &= \vec{h}_\alpha \cdot \vec{x}_N \\ \alpha_{3_N} &= \left(\ddot{\psi}_N \vec{z}_N - \dot{\psi}_N \vec{h}_\Omega \right) \cdot \vec{z}_W \end{aligned} \tag{5.21}$$

Again for compactness we give

$$\begin{aligned} \vec{h}_\alpha = -\frac{1}{u_{1_{ff}}} \Big(& m \vec{\xi}_N^{(4)} + \ddot{u}_{1_{ff}} \vec{z}_N + 2 \dot{u}_{1_{ff}} \vec{\Omega}_{NW} \times \vec{z}_N \\ & + \vec{\Omega}_{NW} \times \vec{\Omega}_{NW} \times \vec{z}_N \Big) \end{aligned} \tag{5.22}$$

The derivative of the net thrust, which appear in Eqn (5.22), are derived to be

$$\begin{aligned} \dot{u}_{1_{ff}} &= -m \vec{\xi}_N^{(3)} \cdot \vec{z}_N \\ \ddot{u}_{1_{ff}} &= -\left(m \vec{\xi}_N^{(4)} + \vec{\Omega}_{NW} \times \vec{\Omega}_{NW} \times \vec{z}_N \right) \cdot \vec{z}_N \end{aligned} \tag{5.23}$$

Note that the equations presented in this section are taken almost directly from Mellinger et. al. [13] but are stated here for completeness of our approach.

5.1.5 Flight Controller

The flight controller synthesizes work by Lee et. al. [26] with Ge and Cui [47] and is composed of three parts: a feedforward component, a feedback component, and a "locally reactive" component. Feedforward inputs, denoted with subscript ff, are generated from the differentially flat mapping in Section 5.1.4 and feedback terms, denoted with subscript fb, are generated via proportional-derivative (PD) tracking of position, velocity, orientation and angular velocity. The locally reactive terms, denoted with subscript lr, are loosely based on the concept of potential fields where proximity to obstacles create a virtual force to "push" the quadrotor away (i.e. the quadrotor reacts to obstacles). The reactive terms were originated from Ge and Cui's work, but were modified during experimentation until a desirable behavior was observed. Since these terms were empirically derived, they no longer represent a gradient of a potential field. As previously noted, the locally reactive terms of the controller are not necessary to achieve real-time obstacle avoidance as the planning framework is fast enough to account for dynamic obstacles on its own. During flight tests, however, the locally reactive controller terms significantly improved the performance of the quadrotor by generating more predictable, graceful manuevers. Equation (5.24) gives the net thrust control input due to the feedforward, feedback, and locally reactive components.

$$\begin{aligned} u_1 &= u_{1_{ff}} + u_{1_{fb}} u_{1_{lr}} \\ &= -\vec{z}_B \cdot \left(\vec{F}_{ff} + \vec{F}_{fb} + \vec{F}_{lr} \right) \\ &= -\vec{z}_B \cdot \left(m\ddot{\vec{\xi}}_N - mg\vec{z}_W + K_\xi \vec{e}_\xi + K_v \vec{e}_v + \vec{F}_{lr} \right) \end{aligned} \tag{5.24}$$

Equation (5.25) presents the control inputs for the moments about the body axes.

$$\begin{aligned} [u_2, u_3, u_4]^T &= [u_2, u_3, u_4]^T_{ff} + [u_2, u_3, u_4]^T_{fb} \\ &= J_B \left(R_B^T R_N \dot{\vec{\Omega}}_{BW} - \vec{\Omega}_{BW} \times \left(R_B^T R_N \vec{\Omega}_{BW} \right) \right) \\ &\quad + \vec{\Omega}_{BW} \times J_B \vec{\Omega}_{BW} + K_R \vec{e}_R + K_\Omega \vec{e}_\Omega \end{aligned} \tag{5.25}$$

The error terms for feedback control are given by Eqn. (5.26) [26]

$$\begin{aligned}\vec{e}_\xi &= \vec{\xi}_N - \vec{\xi} \\ \vec{e}_v &= \dot{\vec{\xi}}_N - \dot{\vec{\xi}} \\ \vec{e}_R &= \frac{1}{2}\left(R_B^{\mathrm{T}} R_D - R_D^{\mathrm{T}} R_B\right)^\vee \\ \vec{e}_\Omega &= R_B^{\mathrm{T}} R_D \vec{\Omega}_D - \vec{\Omega_B}\end{aligned} \tag{5.26}$$

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.

The rotation matrix R_D represents the *desired* orientation to account for feedback and locally reactive terms. This is distinct from the nominal orientation R_N that is independent of feedback and obstacle influence. During perfect trajectory tracking with no nearby obstacles we have $R_N = R_D = R_B$. The rotation matrix R_D is calculated by substituting $\vec{F}_{ff} + \vec{F}_{fb} + \vec{F}_{lr}$ into Equation 5.14 and proceeding with Equations 5.15 and 5.16.

The locally reactive force term, $\vec{F}_{lr}$, in Equation 5.24 is calculated based on obstacle proximity and velocity via

$$\vec{F}_{lr} = \sum_{i=1}^{n_{\text{obs}}} \frac{1}{||\vec{r}_i||^2}\left(-\eta_r \hat{n}_i + \eta_{va} v_{n_i} \hat{n}_i - \eta_{vp} v_{n_i}\left(v_{n_i}\hat{n}_i - \vec{v}_i\right)\right) \tag{5.27}$$

where n_{obs} is the number of obstacles, $\vec{r}_i$ is the position of the closest point on the i^{th} obstacle with respect to the quadrotor body frame, $\hat{n}_i$ is the unit vector in the $\vec{r}_i$ direction, $\vec{v}_i$ is the relative velocity of the i^{th} obstacle with respect to the quadrotor, and $v_{n_i} = \vec{v}_i \cdot \hat{n}_i$. The first two terms in Equation (5.27) represents a repulsive force due to obstacle relative position and velocity, respectively. The third term is a steering term due to obstacle relative velocity. The variables η_r, η_{va}, and η_{vp} are weighting factors for position, aligned velocity, and perpendicular velocity, respectively. For obstacles outside of a user-defined influence region, the locally reactive force in Equation (5.27) is set to zero. Furthermore, if $v_{n_i} < 0$ then the velocity terms of Equation (5.27) are set to zero. It should be again noted that this locally reactive control is non-essential for addressing the motivating themes in Section 1.1; however it improves performance

during physical demonstrations by minimizing the number of replanning events necessary by avoiding occlusion of the existing motion plan. Distinguishing the locally reactive control as non-essential is important because it does require additional obstacle information that is not required by the rest of the framework (i.e. position and velocity data for each obstacle as opposed to just collision detection). Therefore, if we want to be more strict with our assumptions of obstacle data, we could eliminate the locally reactive control without sacrificing the real-time planner as a whole.

5.2 Numerical Experiments

While physical demonstrations are the ultimate test of the framework's effectiveness, limited laboratory space constrains the number and complexity of obstacles sets that can be tested. Therefore a simulation with a maze of obstacles is devised to validate our approach in more complex environments. The simulated environment consists of a corridor with dimensions 20m $\times$ 4m $\times$ 4m with the start and goal states randomly generated from opposing ends of the corridor. Cuboid obstacles are arranged to create a 3-dimensional "maze". A fixed number of spherical obstacles with radii of one meter are placed at random to ensure that we have not inadvertently tailored our algorithm to this specific cuboid-maze obstacle set[4]. Figure 5.3 gives an instance of this obstacle configuration and associated solution. The start state is shown on the left side of the image and the goal state is obscured by the final obstacle on the right. The initial motion plan, as returned by *kino*-FMT (see Sections 2.3.4 and 4.1.4), is indicated in blue and the smoothed, dynamically feasible trajectory (see Section 5.1.3) is indicated in multicolor.

The primary performance metrics of the real-time planning framework are considered the online computation time, solution cost, failure rate, and classification accuracy of reachable sets by the machine learning algorithm. Here we discuss the performance in terms of computation time, solution cost, and failure rate; saving

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

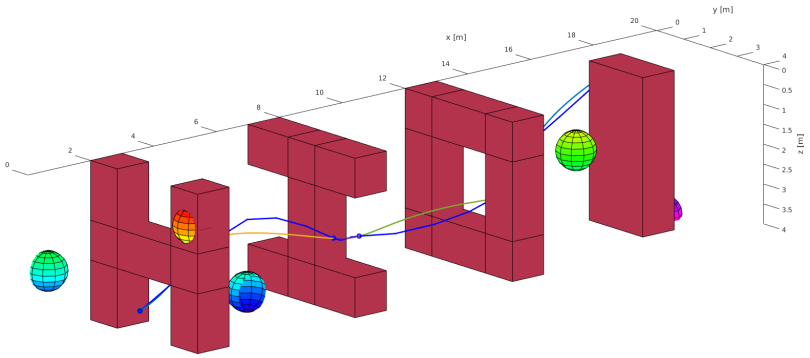


Figure 5.3: A instance of one of the simulated flight tests with a 3D maze of wall structures and randomly placed spherical obstacles.

discussion of the machine learning performance for Section 3.3.

Through the simulated test campaign it was determined that the performance metrics were most dependent upon two *design variables*: number of sampled states and number of terminal state neighbor connections; and a third, situation-dependent variable: obstacle coverage. The simulated test campaign involved selecting values for the design variables and executing 100 trials for each combination. For each trial the start state, goal state, and spherical obstacle placement were randomized. We summarize the trade-offs between design variables and performance metrics in Table 5.1.

Figures 5.4 and 5.5 illustrate the trends of performance metrics as functions of the design variables. As expected, with increasing number of samples N_{sample}, the solution cost, $\mathcal{J}$, decreases while computation time increases. Based on Figure 5.4, we can see that $N_{\text{sample}} = 500$ is an acceptable sample density for this obstacle configuration as there is marginal decrease in solution cost for higher sample numbers. As shown in Table 5.1, 500 samples corresponds to an average solution time of 0.110 seconds for a fixed number of terminal state neighbors and 0.154 seconds for a fixed ratio of terminal state neighbors. It is argued that these computation times represent 'real-time' planning; we verify this claim with physical demonstrations in Section 5.4.2 and compare to computation times in the existing literature in Section 5.4.3.

The effect of neighborhood sizes for the terminal states (i.e. the number of states in the pre-sampled set V_s for which connections are made to the start and goal states)

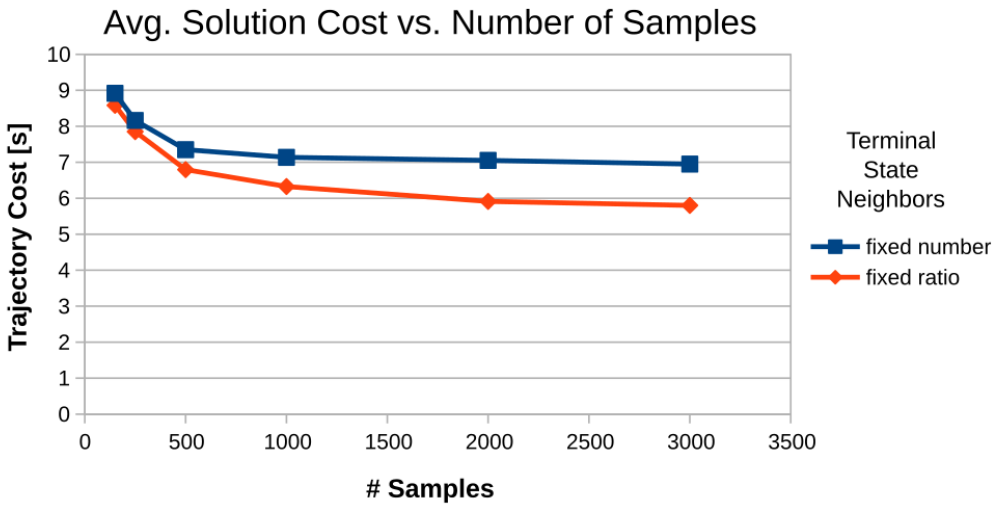


Figure 5.4: Average trajectory cost as a function of number of state samples and number of terminal state neighbors for a fixed obstacle coverage.

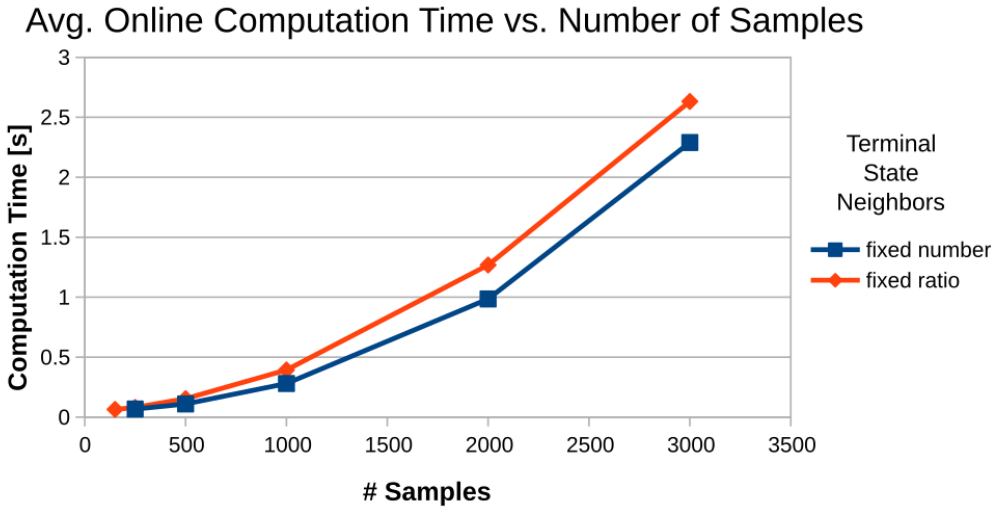


Figure 5.5: Average online computation time as a function of number of state samples and number of terminal state neighbors for a fixed obstacle coverage.

Table 5.1: Trajectory cost and computation time breakdown for the Real-Time Kinodynamic Framework for a range of design variables

	Design Variables		Performance Metrics	
	# Samples	**# Terminal State Neighbors**	**Avg. Trajectory Cost [s]**	**Avg. Computation Time [s]**
Fixed Numbers	150	10	8.91	0.060
	250	10	8.16	0.067
	500	10	7.35	0.110
	1000	10	7.14	0.280
	2000	10	7.05	0.985
	3000	10	6.95	2.289
Fixed Ratios	150	15	8.58	0.065
	250	25	7.85	0.081
	500	50	6.79	0.154
	1000	100	6.33	0.394
	2000	200	5.91	1.268
	3000	300	5.80	2.632

is also presented in Figures 5.4 and 5.5. For the fixed number of terminal state neighbors, the start and goal states are connected to the precomputed roadmap at the 10 closest states in the set of pre-sampled states, V_s; where closeness is approximated by the machine learning algorithm. For the fixed ratio of terminal state neighbors, the start and goal states are connected to the closest 10 *percent* of the set V_s. Each connection between the terminal states and the precomputed roadmap constitutes an online OBVP solution; therefore the fixed number corresponds to $\mathcal{O}(1)$ online OBVPs, where the fixed ratio corresponds to $\mathcal{O}(N_{\text{sample}})$ online OBVPs.

This comparison of number of terminal state neighbors is made to determine the effect of restricting the online OBVP solutions to constant order, which is argued to be an enabling technique for real-time kinodynamic planning. From Figure 5.5 we see that, indeed, restriction of terminal state neighbors leads to a reduction in online computation time of up to 15%. While 15% is not a staggering difference in computation time, it is important to note that this is only representative of the quadrotor system where much work has been done to minimize the computation time

for OBVPs (see Section 2.3.4). For more general systems where OBVPs may be very computationally expensive, this restriction to $\mathcal{O}(1)$ online OBVPs may reduce online planning times by several orders of magnitude [2]. The decrease in computation time, however, comes at the expense of increased solution cost as indicated by Figure 5.4.

Another insightful question is, *for a given obstacle coverage, what is the appropriate number of samples?*. As indicated by Figure 5.5, it is desirable to use the minimum number of samples, N_{sample}, while still achieving acceptable solution cost, as this requires the minimum computation. Since the data given in Table 5.1 only represents a single obstacle coverage, a second test campaign was run to determine the necessary sample count as a function of obstacle coverage. Figures 5.6, 5.7, and 5.8 summarize the data from this second test campaign. Note that we referred to *approximate* obstacle coverage which is measured as the ratio of obstacle volume to unobstructed workspace volume. This value may be greater than one because obstacles were placed at random and overlapping volumes were double counted.

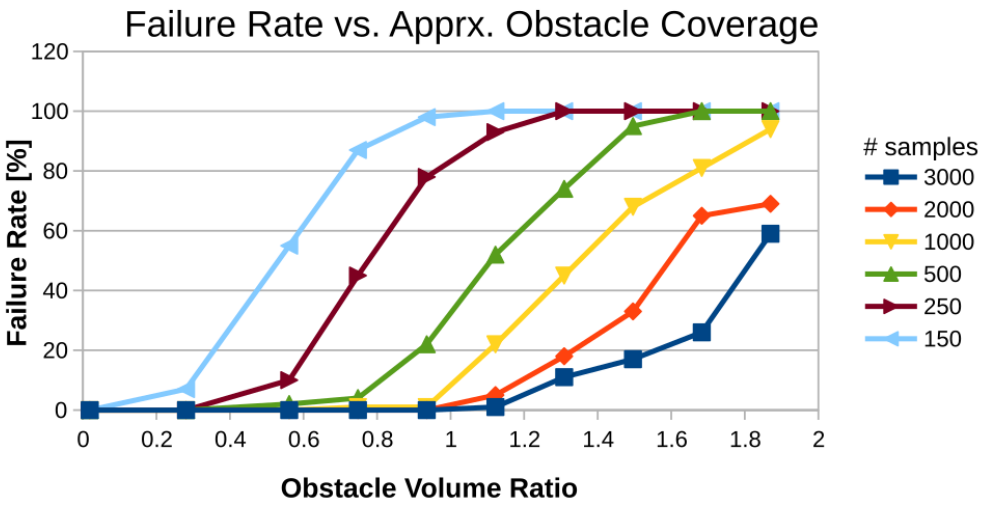


Figure 5.6: Rate of failure to find solution as a function of approximate obstacle coverage for a range of sample sizes.

We immediately see several trends in the data that we expect. First, Figure 5.6 shows that as the obstacle coverage increases, we must use larger sample numbers to prevent planner failure. Lower sample numbers, $N_{\text{sample}} < 500$, quickly rise to 100%

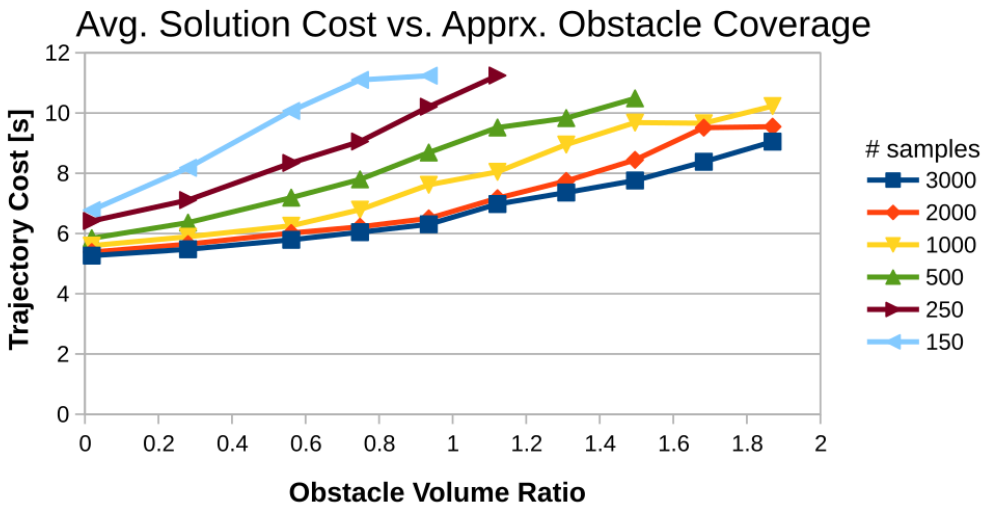


Figure 5.7: Average solution trajectory cost as a function of approximate obstacle coverage for a range of sample sizes.

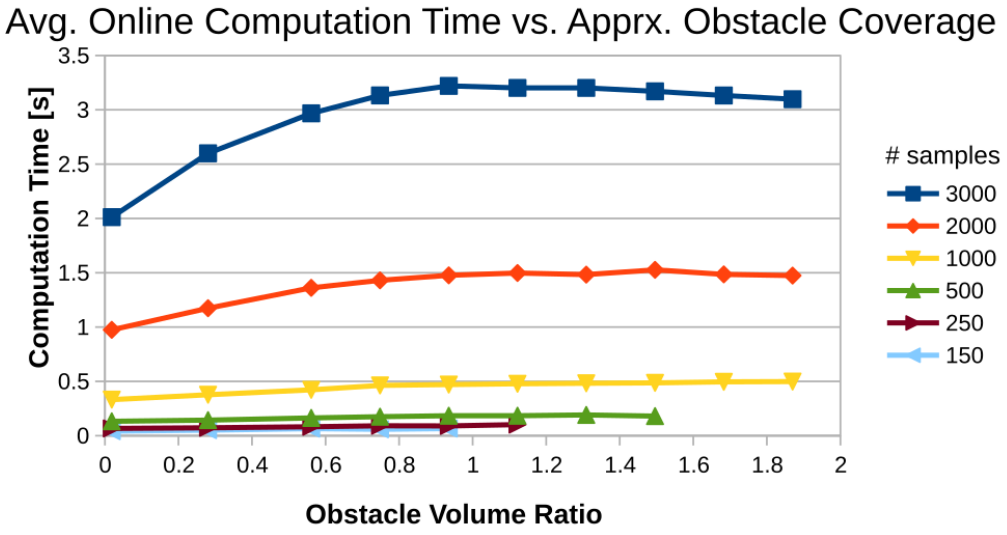


Figure 5.8: Average computation time as a function of approximate obstacle coverage for a range of sample sizes.

failure with increasing obstacle coverage. Note that for sample counts of $N_{\text{sample}} = \{1000, 2000, 3000\}$, the curves diverge from 0% failure at roughly the same obstacle volume ratio. Therefore, $N_{\text{sample}} = 1000$ is a favorable sample count because higher sample counts give no better guarantees for 0% failure at higher obstacle coverage.

Figure 5.7 gives average cost of a solution trajectory as a function of obstacle coverage and sample count. We see two expected trends: solution cost increases with increasing obstacle coverage and decreases with increasing samples count (as was indicated in Figure 5.4). We also see that there is marginal improvement in solution cost beyond a sample count of 2000.

Figure 5.8 gives the online computation time as a function of sample count and obstacle coverage. Again we see the expected trends that computation time increases with increasing obstacle count and with increasing sample count. In more detail, we see that computation time roughly doubles for each tier of sample counts. This leads to large increases in computation time for sample counts greater than 1000. Based on this observation, plus the observation that there is marginal improvement in solution cost beyond $N_{\text{sample}} = 2000$, plus the observation that $N_{\text{sample}} = \{1000, 2000, 3000\}$ all diverge from 0% failure at the same obstacle coverage, we assert that $N_{\text{sample}} = 1000$ is the best suited sample count for our physical experiments. Figure 5.8 shows that computation times for $N_{\text{sample}} = 1000$ are less than 0.5 seconds, even in the worst case.

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

Table 5.2: 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	$\lvert\Delta z\rvert$	$(\Delta z)^2$	$(\Delta z)^3$	$\sqrt{(\Delta x)^2+(\Delta y)^2+(\Delta z)^2+(\Delta \dot{x})^2+(\Delta \dot{y})^2+(\Delta \dot{z})^2}$
$\dot{x}_1$	$\dot{x}_2$	$\lvert\Delta \dot{x}\rvert$	$(\Delta \dot{x})^2$	$(\Delta \dot{x})^3$	
$\dot{y}_1$	$\dot{y}_2$	$\lvert\Delta \dot{y}\rvert$	$(\Delta \dot{y})^2$	$(\Delta \dot{x})^3$	
$\dot{z}_1$	$\dot{z}_2$	$\lvert\Delta \dot{z}\rvert$	$(\Delta \dot{z})^2$	$(\Delta \dot{x})^3$	

5.3 Machine Learning of Reachable Sets

Due to the reliance on machine-learning of neighbor sets, it is important to determine the classification accuracy of the `Near` algorithm. In this work we apply the `Near` algorithm (see Section 3.1) to the control-penalized double integrator system presented in Section 2.2.4. The state space of the double integrator system in Equation (2.7) is 6-dimensional. The two boundary values for an OBVP are concatenated into a 12-dimensional attribute vector, given as $\mathbf{p}$ in Equation (3.1). The feature vector is a 33-element vector, given in Table 5.2, composed of nonlinear mappings of elements from the attribute vector. A third order kernel function is chosen; therefore $p = 3$ in Eqn. (3.2). 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 [5]. The average testing error

[5]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.

Table 5.3: Training and testing accuracy of machine-learning-based neighborhood classification algorithm

# Training Examples	Avg. # Training Errors	# Testing Examples	Avg. # True Positives	Avg. # True Negatives	Avg. # False Positives	Avg. # False Negatives	Avg. Testing Error [%]
20000	0.6	30000	2693	26600.6	371.8	334.6	2.35

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 [2]. Table 5.3 gives the training and testing results. A 'positive' indicates that `Near` 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[6].

The information given in Table 5.3 only tells us the rate of neighborhood classification error, it does not tell us where in the state space these misclassifications occur. To form a deeper understanding of where/why misclassification of neigbhors occur, we illustrate the `Near` results with a simplified case. Figures 5.9 and 5.10 present a set of trials for the neighborhood classifier when compressed to 2 spacial dimensions and one velocity dimensions. Each trial, represented by it's own image in Figures 5.9 or 5.10, attempts to classify the reachable neighborhood of an initial state. For each trial, the initial state is the origin with a y-velocity ranging from 0 m/s to 14 m/s. The final states for each classification are spread across the xy-axes and all have a final velocity of zero. The neighborhood cost threshold is 2.178 seconds which corresponds to the 10^{th} quantile of costs for this set of training data.

The reachability of each final state is assessed by solving an OBVP, as discussed in

[6]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.

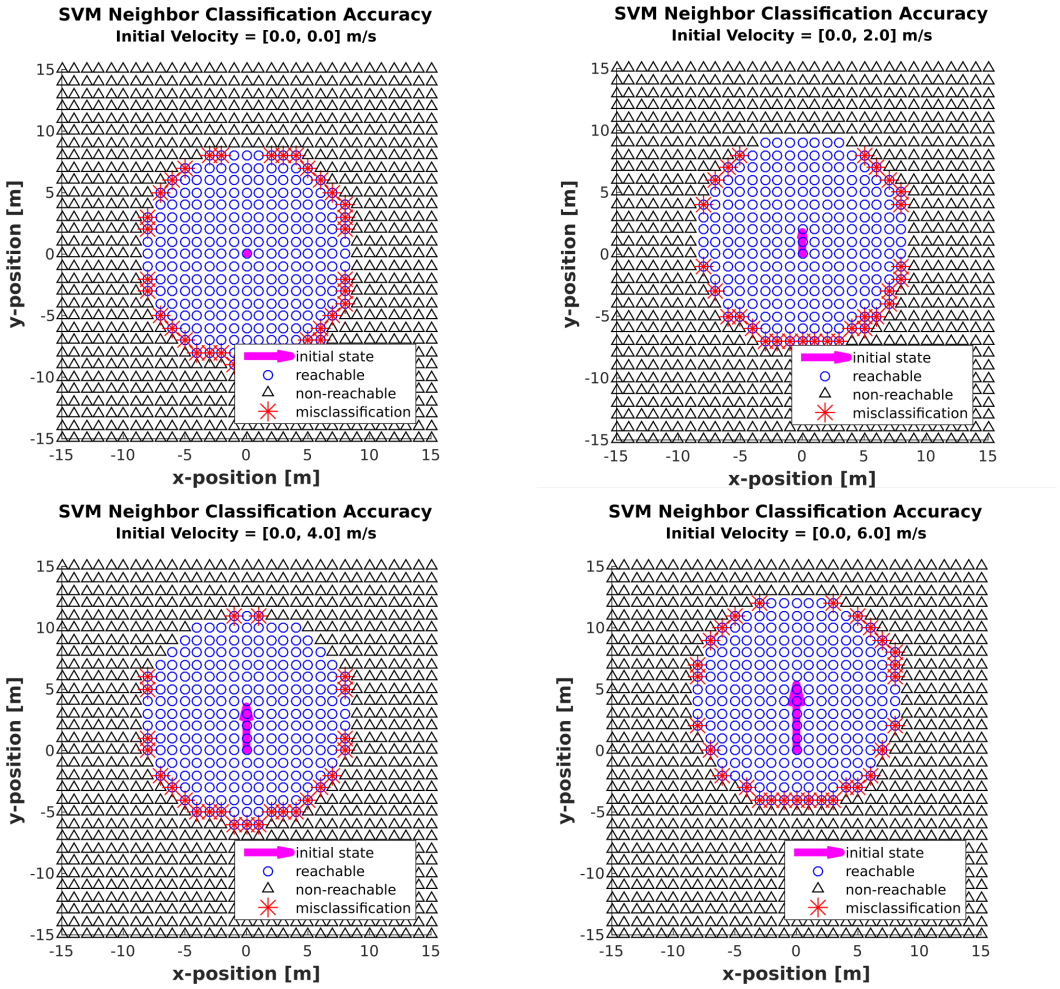


Figure 5.9: *Part 1*: Simplified, 2-dimensional reachable set classification. For all cases, the start state is at the origin with an initial velocity in the y-direction. The final states all have a velocity of zero.

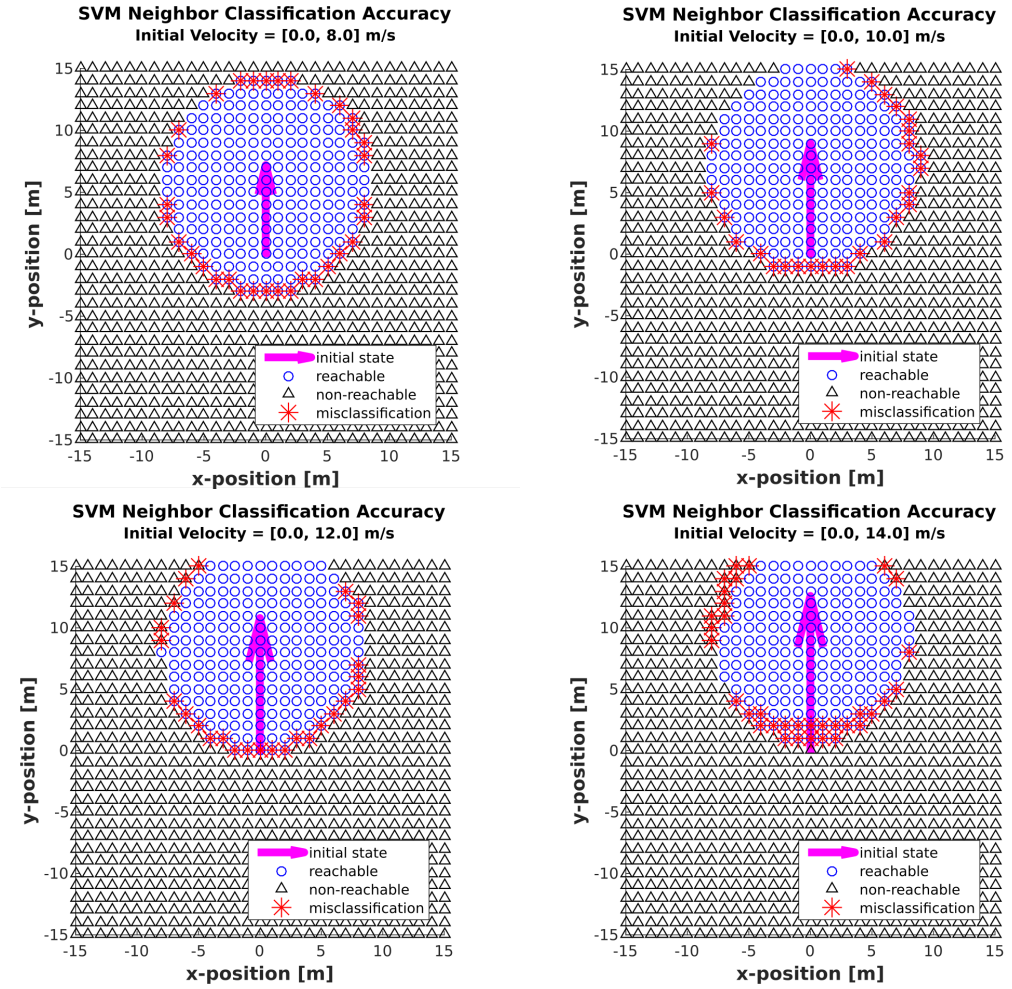


Figure 5.10: *Part 2*: Simplified, 2-dimensional reachable set classification. For all cases, the start state is at the origin with an initial velocity in the y-direction. The final states all have a velocity of zero.

Section 2.3.4, between the initial state and each final state. Reachable states are indicated in blue circles and non-reachable are indicated in black triangles. The machine learning algorithm, `Near`, is then applied to estimate reachability. Misclassification of reachability, whether it be a false-positive or false-negative, is indicated by a red star.

As expected, Figures 5.9 and 5.10 show that the true reachable set shifts further along the y-axis as the initial velocity in the y-direction increases. We also see that misclassification of reachability always occurs on the boundary between the reachable and non-reachable sets. This is a desirable result for a well-trained algorithm. A poorly-trained algorithm would make classification errors well within the reachable or non-reachable sets. If a poorly trained neighborhood classifier were used in the real-time planning framework, we would end up solving OBVPs for states that our well outside of neighborhood - likely leading to collisions with obstacles, increasing computation time - or fail to recognize nearby states, thus increasing solution cost of our trajectory.

It is important to note that an analytical solution exists for the reachable set of the control penalized double integrator [20]. The question is then, why would you use the machine learning approximation for reachable sets when an analytical solution exists? This question lies at the balance point between the two motivating themes presented in Section 1.1. While we want to demonstrate real-time planning for a quadrotor, we also want to validate a planning framework that is more applicable to a more general set of dynamical systems. Since an general dynamical system cannot be expected to have an analytical solution for reachable sets, we maintain the use of the machine learning approach in effort to validate it in physical experiments.

5.4 Flight Demonstrations

5.4.1 Experimental Flight Setup

The real-time framework is demonstrated on a Pixhawk autopilot flown on a DJI F-450 and F-330 frame. Positioning information is provided by a Vicon motion tracker

Table 5.4: Computational platform and programming language for the major components of the real-time framework.

Process	Reference	Processor	Language
localization	NA	workstation	C++
precomputations	Sec. 4.1.1	workstation	MATLAB
neighborhood estimation	Chp. 3	workstation	MATLAB
OBVP solutions	Sec. 2.3.4	workstation	MATLAB
sampling-based planning	Sec. 4.1.4	workstation	C++
min-snap smoothing	Sec. 5.1.3	workstation	MATLAB
flat-to-nonlinear mapping	Sec. 5.1.4	Pixhawk	C/C++
flight control	Sec. 5.1.5	Pixhawk	C/C++

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. 5.11. Table 5.4 gives detailed information on the computational platform and programming language for each of the major components of the framework discussed in Section 5.1. Future work will convert all portions of the online phase (see Alg. 5) to C++ to be run on an embedded processor on the quadrotor.

The quadrotor is navigating an indoor environment with dimensions of approximately 3m $\times$ 4m $\times$ 3m. The framework was tested on a range of obstacle sets, two of which are discussed in detail in Section 5.4.2.

5.4.2 Experimental Flight Results

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

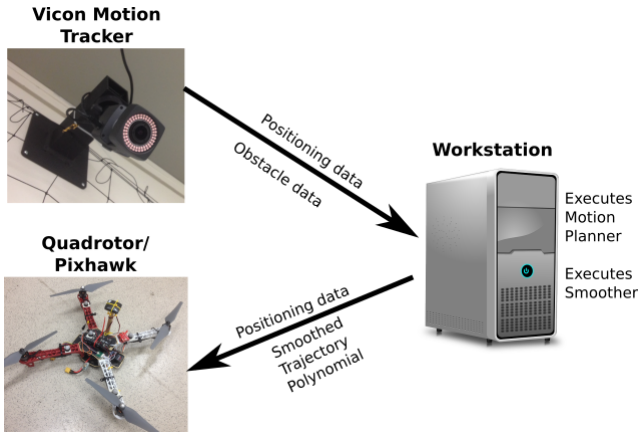


Figure 5.11: Communication/computation structure for flight tests.

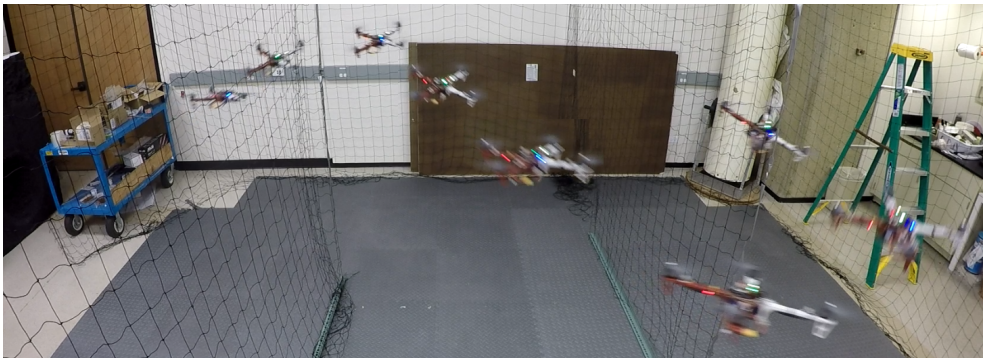


Figure 5.12: Timelapse of quadrotor navigating static obstacles.

Figure 5.12 gives a timelapse of the most basic flight test performed: The quadrotor navigating a set of parallel walls with no dynamic obstacles present. The walls are arranged to create a z-shaped corridor 1.5m in width. This test acted as the first validating experiment for the real-time kinodynamic planner. The solution used 500 sampled states and required 0.313 seconds of online computation time. This test also demonstrated the agility of the quadrotor platform as indicated by the banged-turns as it rounded the corners. Building from this initial experiment, originally presented in Allen and Pavone [23], the planning code was optimized to further reduce computation times and dynamic obstacles were introduced.

In the first physical experiments with dynamic obstacles, shown in a sequence of images in Figure 5.13, the autonomous quadrotor is presented with a single obstacle

in between its start state and its goal region. This creates two doors, each roughly one meter in width, from which the quadrotor can "choose" to navigate. When the real-time planner completes, the quadrotor begins executing the trajectory through the nearest of the two doors. Upon nearing the door, a human subject enters and presents a dynamic obstacle; in this demonstration the obstacle is the point of a fencing blade which is numerically expanded to a sphere with radius equal to the arm length of the quadrotor. The human subject continues to approach the quadrotor, causing it to be "pushed back" due to the reactive controller (see Section 5.1.5), until the existing trajectory is completely obstructed and replanning is initiated. Figure 5.14 shows the moment of replanning along with the solution provided by the real-time planning framework. Due to the proximity to the initially chosen door, the quadrotor executes three planning cycles that attempt to navigate the initial door and the dynamic obstacle. Since the dynamic obstacle is acting advicarially, always obstructing the chosen trajectory, this continues until the quadrotor is forced to a point where the second door becomes the optimal solution and the quadrotor navigates to the goal region without further obstruction from the human subject.

Figure 5.15 gives a second scenario where the autonomous quadrotor is tasked with navigating a parallel-walled maze with a corridor of roughly 1.5m in width. Again, a human subject introduces a set of dynamic obstacles to force online recomputation of the motion plan. During this demonstration the dynamic obstacles are presented immediately before the quadrotor rounds a corner, causing an abrupt replanning cycle that navigates over- and between the dynamic obstacles and the wall. Because the human subject does not continue to act advicarially in this case, only a single recomputation of the motion plan is necessary.

Figures 5.14 and 5.15 show that the online computation times for planning are on the order of 1/4 of a second.

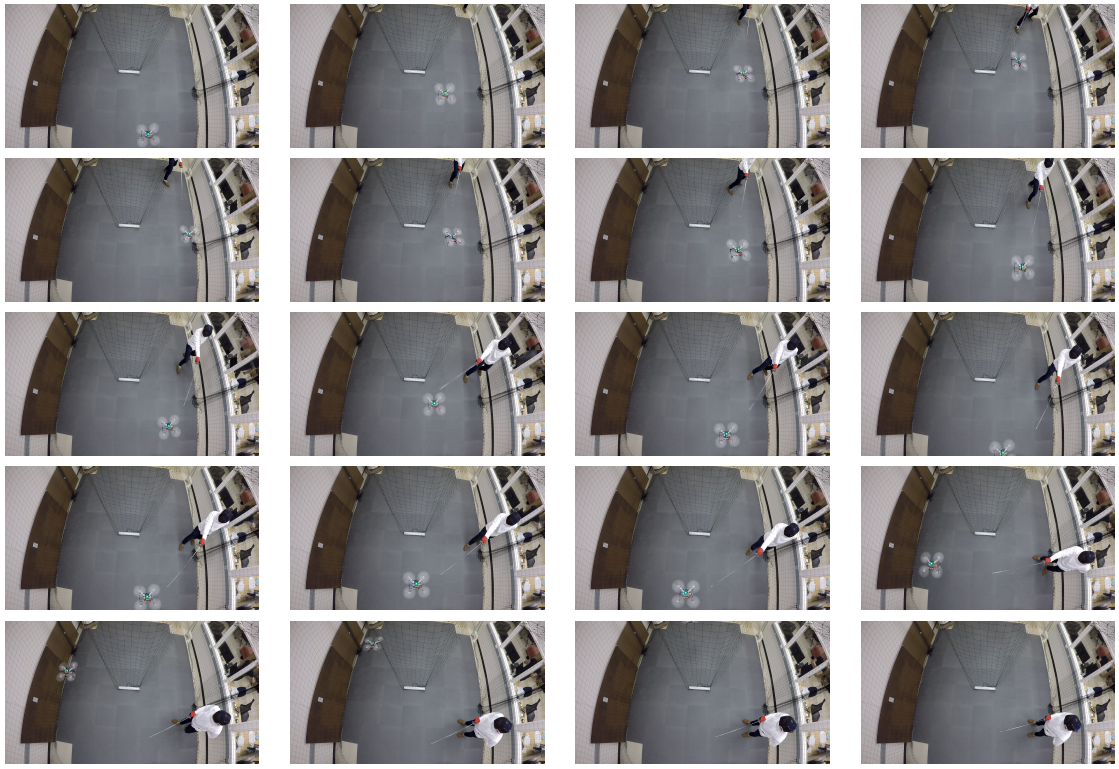


Figure 5.13: Sequence of images, in order from left to right and down, showing the quadrotor navigating the two-door environment with a human adversary obstructing one door using a fencing blade. The quadrotor initially attempts to navigate the door on the right. A human subject can be seen entering and obstructing the trajectory, causing the quadrotor to be "pushed back" due to the reactive controller. After several replanning events that attempt to navigate the door on the right, all of which are obstructed by the human subject, the left door eventually becomes the optimal solution which is determined by the real-time framework and subsequently executed by the quadrotor.

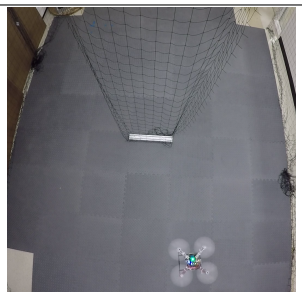
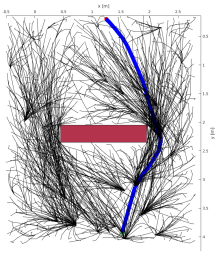
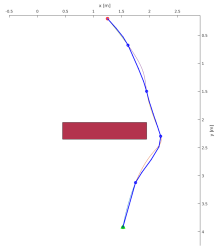
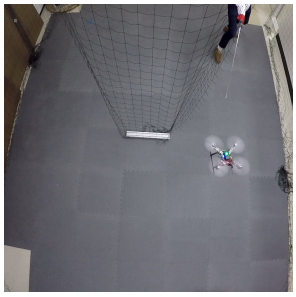
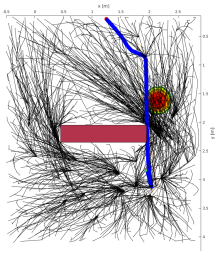
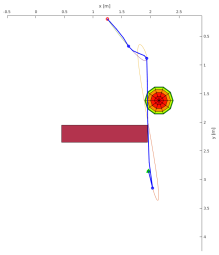
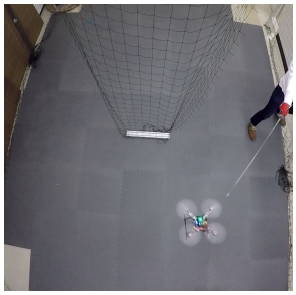
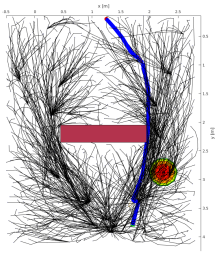
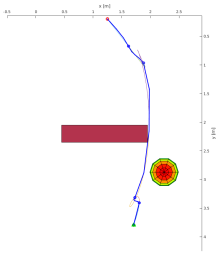
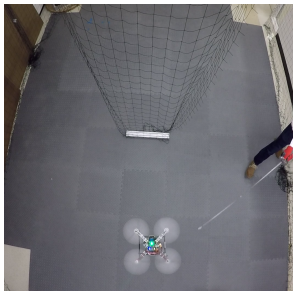
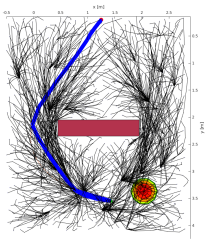
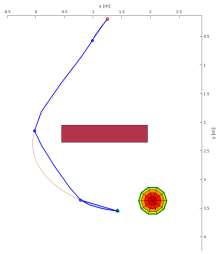
Computation Time [s]	Screen Capture	Exploration Tree	Solution Trajectory
$\Delta t = 0.2518$			
$\Delta t = 0.2474$			
$\Delta t = 0.2700$			
$\Delta t = 0.1997$			

Figure 5.14: Time sequence of real-time planning in "two door" environment with static and dynamic obstacles. Column 1 gives the online computation time for the planning event. Column 2 gives screen capture of the moment of replanning. Column 3 gives the tree explored during replanning with the preliminary solution in blue. Column 4 gives the preliminary planning solution and the smoothed trajectory. Obstacles are represented by red rectangles or spheres

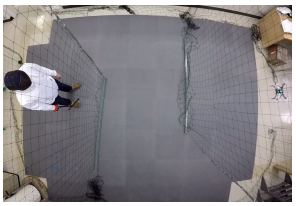
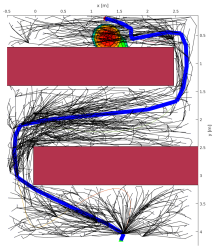
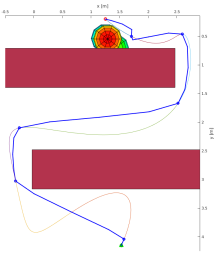
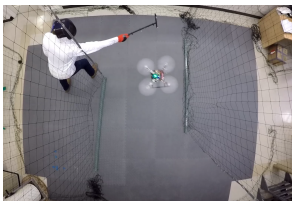
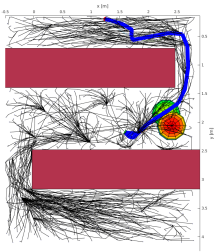
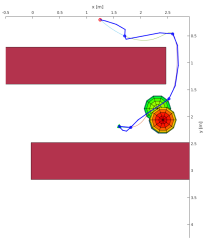
Computation Time [s]	Screen Capture	Exploration Tree	Solution Trajectory
$\Delta t =$ 0.3260			
$\Delta t =$ 0.2968			

Figure 5.15: Time sequence of real-time planning in "maze" environment with static and dynamic obstacles. Column 1 gives the online computation time for the planning event. Column 2 gives screen capture of the moment of replanning. Column 3 gives the tree explored during replanning with the preliminary solution in blue. Column 4 gives the preliminary planning solution and the smoothed trajectory. Obstacles are represented by red rectangles or spheres

Table 5.5: Computation time breakdown for the Real-Time Kinodynamic Framework for differing numbers of sampled states

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

5.4.3 Discussion

A primary goal of this work, the goal that is implied by the second motivating theme in Section 1.1, was to prove that the entire planning framework can be executed in a real-time environment. As shown in Figures 5.14 and 5.15, we achieve online computation times between 0.20 sec and 0.33 sec for 1000 sampled nodes. These computation times are in good agreement with those presented in Section 5.2 which ranged from 0.11 sec to 0.5 sec for comparable sample counts. The difference in computation times between simulated and physical experiments are due to differing obstacle sets (simulated tests being more densely obstructed) and additional computational tasks for physical experiments (e.g. communication of solution trajectories). To better understand the breakdown of computation time during physical experiments, Table 5.5 gives the percentage of computation time for separate tasks in the planner for a range of sample counts.

Referring to Table 5.5, the computation time is broken down into percentages for the major components of the framework: neighborhood classification for the terminal states (see Section 3.1); neighborhood OBVP solutions for the terminal states (see Section 2.3.4); sampling-based motion planning (see Section 4.1.4); path smoothing to generate a minimum-snap, dynamically feasible trajectory (see Sec. 5.1.3); and communication (see Sec. 5.4.1). We see that the majority of the computation time is consumed by the solution of optimal boundary value problems between the terminal states, $\mathbf{x}_{\text{init}}$ 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 `Near` classification time is 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.

To compare the computation times we achieved to those presented in the existing literature, Webb and van den Berg simulate an almost identical problem; however they do not perform any path smoothing or communication to a physical quadrotor [20]. With 1000 sampled states Webb and van den Berg's solution takes 51.603 seconds; i.e. $\sim$150x, or 2 orders of magnitude, slower than the technique presented here. Richter et. al. do not state the computation time for motion planning demonstrated in their work [11]. 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. $\sim$9.1x, or 1 order of magnitude, slower than the slowest physical experiment computation time presented here. Therefore the real-time kinodynamic framework demonstrates a significant reduction in computation time when compared to existing techniques.

Frazzoli et. al. boasts the most impressive computation times with sub-second execution for the similar, but not identical, helicopter system navigating static spherical objects [9]. Computation times for dynamic obstacles rise to 10s of seconds for a parallel wall obstacle set. Therefore we again see our method produce roughly 2 orders of magnitude improvement in computation time when compared to existing techniques for dynamic obstacles. 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.

We note here that our physical demonstrations were not infallible; roughly 50% of

experiments ended in some form of a crash. These failures were found to be due to two factors: poor system identification of the quadrotor leading to inaccurate dynamic parameters in the flight controller (see Section 5.1.5), and loss of positioning data due to exiting Vicon coverage. Both of these failure modes were outside the scope of this work which was solely focused on developing the real-time planning framework. These failure modes, do however, motivate future work: advanced system identification for improved controller performance and robust, onboard estimation/localization to eliminate reliance on a motion capture system.

As another note on future work, all experiments were conducted in an indoor, restricted environment. To make this work applicable to a non-laboratory environment, an environment scaling technique must be developed. As an example, if this quadrotor was tasked with delivering a package across town, the scale of the town may be too large to sample states to a sufficient density throughout. The quadrotor must be able plan on a local scale, for example to navigate power lines or perhaps moving cars, while also following a global scale plan across the town. A *stitching* scheme can be designed that allows for solving problems at such various scales. This scheme may employ a dense set of local samples that are "dragged" along with the robot as it moves through a global environment that is too large for adequate resolution of sampling.

Chapter 6

Conclusions

6.1 Summary

This thesis has produced 3 major contributions, each of which being the respective focal point of Chapters 3 - 5.

In Chapter 3 we show how machine learning can be used to approximate the reachable sets for dynamical systems in a query-based fashion. For the range of systems for which this technique was tested, we showed that a testing error of less than 10% is achievable, with some system achieving much lower errors. This was also accompanied by a significant reduction in computation time of up to 3 orders of magnitude when compared to the numerical solutions for reachable sets. The machine learning of reachable sets was then used as an pivotal technique for real-time motion planning for kinodynamic systems.

In Chapter 4 we developed a novel framework for real-time motion planning for differentially-constrained systems. The framework relied on the Fast Marching Tree algorithm, an asymptotically optimal sampling-based planner, to solve planning problems while utilizing a precomputed look-up table of boundary value solutions. The look-up table was generated by implementing an offline-online computation paradigm that reduced the number of online OBVP solutions from $O(N^2)$ to $O(N)$. Machine learning of reachable sets, as discussed in Chapter 3, was then used to further reduce the number of online OBVP solutions to $O(1)$. The framework was shown to reduce

the only computation time for kinodynamic planning problems by up to six orders of magnitude while remaining general enough to be applied to a wide range of problems.

Chapter 5 tailored the general framework from Chapter 4 to the quadrotor system. Implementing the tailored framework on a physical system produced, arguably, the first demonstration of real-time kinodynamic planning on a quadrotor robot. The property of differential flatness was leveraged in the planning framework to minimize computation time and aid in flight control. By first approximating the quadrotor as a double-integrator system, solving the planning problem to generate a trajectory through the flat output space, and then using differentially flat mapping back to the full state and control space, real-time computation of a kinodynamic motion plan was realized. Computation times of 0.25 seconds allowed a physical quadrotor robot to navigate an obstructed indoor environment, even in the presence of dynamic and adversarial obstacles.

6.2 Future Work

The contributions of this work open the door to many future directions, of which a few are discussed. We separate the future work into categories based on extensions in depth, breadth, or theory of the current work.

6.2.1 Extensions in Depth

Chapter 5 presented experimental results on a quadrotor system. There are several aspects of this experimental work that could be extended to generate a more in depth, self-contained, fully embedded implementation of the framework.

Currently real-time computations are distributed over a range of processors onboard and offboard the quadrotor. In future works, all processing will be moved onboard the quadrotor. Reduction in processing power of the embedded processors will be compensated by code optimization. Implementation on an embedded GPU will further accelerate computation times.

Sensing, localization, and mapping are currently achieved with a Vicon motion

capture system. While this has been an important tool for isolation of the planning problem during labratory-based development, it is an untenable solution for real-world applications. Therefore the planning framework developed in this thesis must be fused with a sensing-capable platform, performing localization and obstacle detection onboard. Since the sensing and planning process are parallel, each can have a dedicated processor to avoid computational slow-down. This, of course, is at the cost of added weight and power consumption due to the additional hardware needed.

The planning framework implemented in this work assumed perfect knowledge of state and obstacles. For real-world applications, this assumption must be relaxed so that sensing and dynamic system uncertainties can be addressed. The additional computational complexity introduced by uncertainties can be addressed by using a parallel sampling-based planning algorithm, such as that developed by Janson, Schmerling, and Pavone [48].

6.2.2 Extensions in Breadth

While Chapter 5 demonstrated real-time planning for a quadrotor system, the framework developed in Chapeter 4 was designed to be applicable to general robotic systems. Therefore it is natural to extend the breadth of this work by applying and demonstrating it on other robots.

Autonomous Cars: While there exists a rich literature in the field of planning and control for autonomous cars [49], the framework proposed in this thesis has the potential to contribute further. Due to its focus on *kinodynamic* planning, as opposed to geometric planning with smoothing, adapting the framework presented in Chapter 4 to the structured environment of road driving may allow agile maneuvers that push the dynamic limits of a car's abilities. This agility could prove to be essential for incorporating emergency evasive maneuvers into autonomous car planning and control.

Robotic Spacecraft: Spacecraft represent an important application for robotic motion planning, especially when considering operations far from Earth that suffer from

long communication delays (e.g. a robotic rendezvous at Mars during a Mars sample return mission). Spacecraft introduce unique control constraints such as thruster plume impingement. Furthermore, due to the immense cost of any spacecraft, greater safety margins and guarantees on non-failure of the planning algorithm must be addressed.

Humanoid Robotics and Robotic Arms: Humanoid robots and robotic arms often represent a different, challenging paradigm for motion planning. Instead of pure obstacle avoidance, often humanoid robotics actually want to physically contact their environments, albeit in a controlled, dexterous fashion. To address this, the framework would need to be tailored around a new notion of collision checking that allow for environment interaction.

Autonomous Marine Vehicles: Due to their slow maneuvers and large scale, large marine vehicles – such as cargo ships – may seem to have little in common in fast, agile robots like quadrotors. In fact, the relatively slow dynamics of a marine craft do not diminish the complexity of the planning problems they must solve; on the contrary, it may make them more difficult. Due to their large momentum vectors and relatively low control authority, marine vehicles would require tailoring the framework to focus heavily on planning over large time horizons with strong safety margins.

Sling-Load Robots: Robotic cranes could have a dramatic impact on the shipping industry if, for example, even a modest speed up in unload time for a large cargo ship could be realized. By casting a robotic crane as a kinodynamic motion planning problem, highly-dynamic maneuvers could be executed for sling-loaded payloads that may be multiple times faster than a human operator. The unique challenge of crane robots comes from the fact that their dynamics typically do not appear as ordinary differential equations, as given in Equation (2.3), but instead must be represented in the more general differential algebraic equation (2.2). To implement on a crane robot, the state connection module of the planning framework would need to be tailored to address this more complex set of dynamics.

Multi-Class Robotic Infrastructure: the flexibility of this planning framework could enable robotic infrastructures that are comprised of many different classes of robots. Each robot would have its own distinct planning problem to be solved, involving

their own set of dynamics and control constraints, but they could all be addressed with variations of the framework presented in Chapter 4. As an example, one could imagine a robotic shipping and distribution infrastructure that involved robotic cargo ships navigating a busy sea port; which are then unloaded by robotic cranes that must navigate heavy sling-loaded containers to a cluttered dock; which are then transported to distribution centers via autonomous trucks; where products are finally delivered using aerial drones. Each of these robots would require the solution to unique, real-time kinodynamic planning problems which could be achieved by similar implementations of the same framework.

6.2.3 Extensions in Theory

The results on machine learning presented in Chapter 3 are, in some senses, empirical: we proposed a new application of machine learning algorithms for reachability analysis and then showed their effectiveness. In future work, more theoretical detail will be developed for this component of the planning framework. In particular, bounds on machine learning accuracy as a function of training sample size will be developed.

Thus far all of the applications considered have involved single agent problems. Future work could take the framework proposed in Chapters 4 and 5 and apply them to a multi-agent system. Since the planning framework adjusts to dynamic obstacles (i.e. other robots) in real-time, it would be interesting to see if any emergent behavior would appear from many agents selfishly executing their own motion planner, without any form coordination.

This work relies on a simplistic model of obstacles where only the current state of the obstacle is accounted for. A more complex approach would be to apply a game model to the framework so that obstacles could be treated as other players in a differential game. This would allow for much "smarter" avoidance maneuvers, particularly in multi-agent systems.

Bibliography

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] Peter Leven and Seth Hutchinson. A framework for real-time path planning in changing environments. *The International Journal of Robotics Research*, 21(12):999–1030, 2002.

[11] 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.

[12] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial Trajectory Planning for Quadrotor Flight. In *International Conference on Robotics and Automation*, 2013.

[13] 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.

[14] Sertac Karaman and Emilio Frazzoli. Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. A Fully Autonomous Indoor Quadrotor. *Robotics, IEEE Transactions on*, 28(1):90–100, 2012.

[23] 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*, pages 928–934, Seattle, WA, May 2015.

[24] Ross Allen and Marco Pavone. A Real-Time Framework for Kinodynamic Planning with Application to Quadrotor Obstacle Avoidance. In *AIAA Conf. on Guidance, Navigation and Control*, pages 1–18, San Diego, CA, January 2016.

[25] I Michael Ross and Fariba Fahroo. A perspective on methods for trajectory optimization. In *Proceedings of the AIAA/AAS Astrodynamics Conference, Monterey, CA*, 2002.

[26] 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.

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

[28] M. Hwangbo, J. Kuffner, and T. Kanade. Efficient Two-phase 3D Motion Planning for Small Fixed-Wing UAVs. In *Proc. IEEE Conf. on Robotics and Automation*, pages 1035–1041. IEEE, 2007.

[29] 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.

[30] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2012.

[31] 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.

[32] J. T. Betts. Survey of Numerical Methods for Trajectory Optimization. *AIAA Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

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

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

[35] 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.

[36] 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.

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

[38] E. Cockayne. Plane pursuit with curvature constraints. *SIAM Journal on Control and Optimization*, 15(6):1511–1516, 1967.

[39] 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, Seattle, WA, May 2015.

[40] 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.

[41] 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.

[42] S. Karaman and E. Frazzoli. Sampling-based Optimal Motion Planning for Non-holonomic Dynamical Systems. In *Proc. IEEE Conf. on Robotics and Automation*, pages 5041–5047, 2013.

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

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

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

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

[47] Shuzhi S. Ge and Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, 2002.

[48] L. Janson, E. Schmerling, and M. Pavone. Monte Carlo Motion Planning for Robot Trajectory Optimization Under Uncertainty. In *International Symposium on Robotics Research*, Sestri Levante, Italy, September 2015.

[49] Brian Paden, Michal Cap, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles. *arXiv preprint arXiv:1604.07446*, 2016.