

High Performance and Safe Flight of Full-Scale Helicopters from Takeoff to Landing with an Ensemble of Planners

Sanjiban Choudhury

Vishal Dugar

Silvio Maeta

Brian MacAllister

Sankalp Arora

Daniel Althoff

Sebastian Scherer

*

Abstract

Autonomous flight of unmanned full-size rotorcraft has the potential to enable many new applications. However, the dynamics of these aircraft, prevailing wind conditions, the need to operate over a variety of speeds and stringent safety requirements make it difficult to generate safe plans for these systems. Prior work has shown results for only parts of the problem. Here we present the first comprehensive approach to planning safe trajectories for autonomous helicopters from takeoff to landing. Our approach is based on two key insights. Firstly, we compose an approximate solution by cascading various modules that can efficiently solve different relaxations of the planning problem. Our framework invokes a long-term route optimizer, which feeds a receding-horizon planner which in turn feeds a high fidelity safety executive. Secondly, to deal with the diverse planning scenarios that may arise, we hedge our bets with an ensemble of planners. We use a data-driven approach that maps a planning context to a diverse list of planning algorithms that maximize the likelihood of success. Our approach was extensively evaluated in simulation and in real-world flight-tests on three different helicopter systems for a duration of more than 109 autonomous hours and 590 pilot-in-the-loop hours. We provide an in-depth analysis and discuss the various tradeoffs of decoupling the problem, using approximations and leveraging statistical techniques. We summarize the insights with the hope that it generalizes to other platforms and applications.

1 Introduction

Unmanned rotor-craft have a high demand in applications such as cargo delivery (Paduano et al., 2015), emergency rescue operations (Scherer et al., 2012) and surveillance (Nigam et al., 2012) due to their dexterity in operating in close vicinity to ground (Fig. 1). Such applications typically require the UAV to visit a sequence of waypoints that may stretch across large distances and different environments. They often also involve flying at varying speed regimes, varying proximity to obstacles and repeatedly landing / taking off from unprepared sites as shown in Fig. 2. Hence, from a motion planning perspective, this problem has several challenging aspects.

1.1 Challenges

Restrictive dynamic constraints. The dynamics of such systems must satisfy a variety of constraints imposed by the control system, flight performance charts (Prouty, 1995) and wind (Seleck et al., 2013; McGee et al., 2005; Techy, 2011). These constraints restrict the reachability of the system, i.e. limit the set of feasible trajectories the vehicle can execute. The reachability plays a critical role when the UAV has to avoid obstacles or plan maneuvers to fly through waypoints.

*S. Choudhury, V.Dugar, S.Maeta, B.MacAllister, S. Arora and S. Scherer are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15206 USA e-mail: {sanjibac,vdugar,smaeta,bmaca,asankalp,basti}@andrew.cmu.edu.



Figure 1: Boeing Unmanned Little Bird (equipped with TALOS system) executes an autonomous landing on snow.

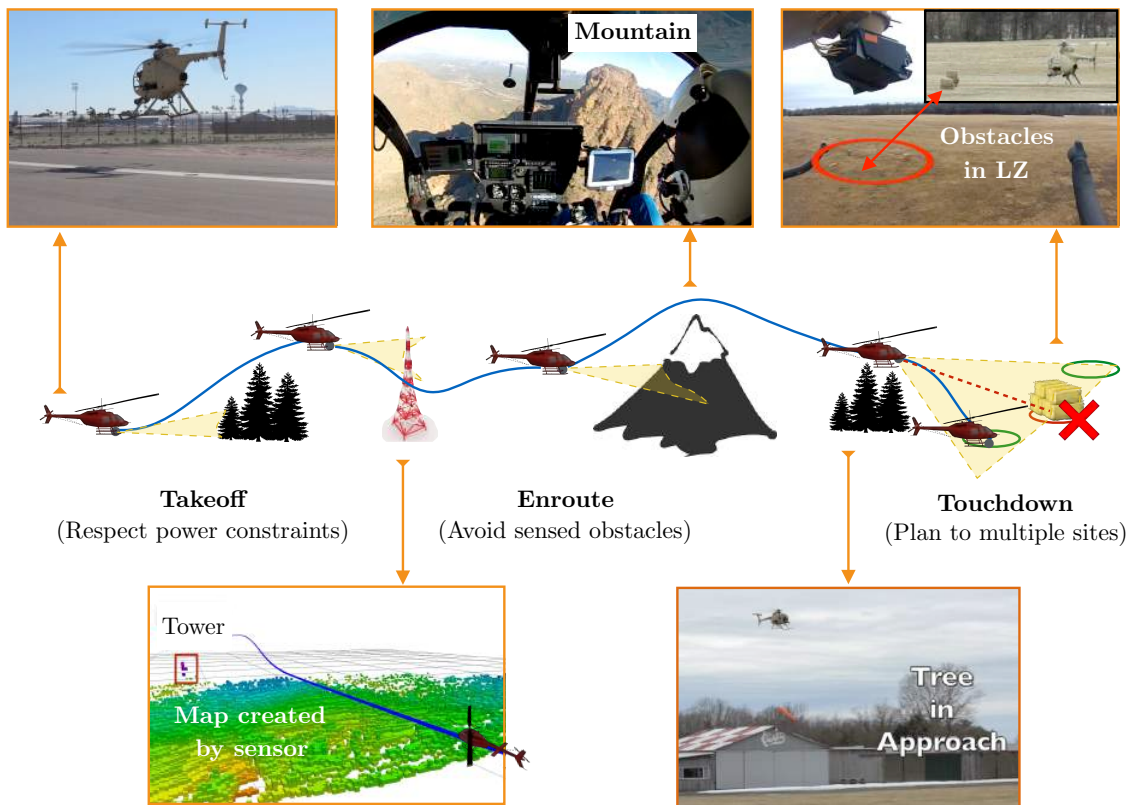


Figure 2: A typical mission for a helicopter can be divided into three phases. During takeoff, the helicopter must use an ascent profile that does not exceed available power while not colliding with nearby obstacles. During the en route phase, the helicopter must avoid collision with sensed obstacles (and no-fly zones) as it flies towards its objective. These scenarios can be quite diverse such as avoiding towers or mountains. During the touchdown phase, the helicopter must land on a clear spot within a designated Landing Zone (LZ) while avoiding obstacles along the descent.

Guarantee safety in partially known environments. Guaranteeing safety implies planning trajectories that satisfy safety constraints for an infinite time horizon which is challenging in partially known environments. Traditional approaches recommend limiting the speed such that the system can come to a full stop within the known volume. Unlike unmanned ground vehicles, coming to a stop is often not a safe option since rotorcraft with a high payload

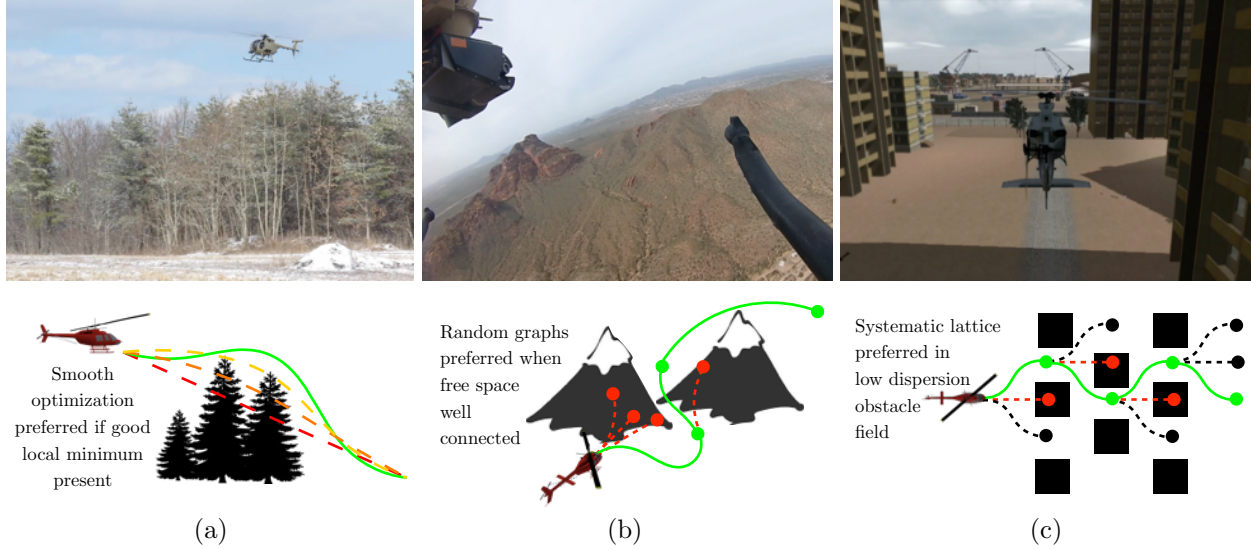


Figure 3: A spectrum of environments, each corresponding to a planning problem that favours particular planning strategies. (a) Smooth trajectory optimization can bend over tree-lines. (b) Random sampling methods are effective when planning around mountains (c) In Manhattan environments, search on a lattice with small edges is effective.

have to move to stay airborne.

Navigation in varying scenarios. Fig. 3 shows a spectrum of different scenarios that it encounters. Not all scenarios are always equally likely - the distribution over scenarios is dictated by the specifics of the mission that the robot has to execute. As the nature of the planning problems for these scenarios differ significantly, it is challenging for a planning algorithm to have consistent real-time performance during the mission. It is also impractical for a human designer to continue tuning algorithm parameters at the onset of every mission.

1.2 Key idea

To deal with the challenges, we present a general trajectory planning architecture for UAVs based on two key ideas:

Efficiently decoupling route planning, trajectory planning and safety. Since the overall problem is intractable to solve in real-time, we propose a way to decouple the problem to compute an approximate solution. Our approach is based on a novel 3 stage decoupling scheme - a global nominal trajectory planner, a receding horizon trajectory planner for avoiding obstacles and a safety checking executive module to guarantee safety. Each of these modules operate at different resolutions, different time scales and examine complementary aspects of the problem.

Hedge our bets with a diverse ensemble of planners. To deal with varying planning scenarios the robot encounters, we propose an adaptive motion planning strategy. We assume we have a library of planning strategies - these could be different algorithms, or different parameters of the same algorithm. Given an environment, we learn to predict which planner to use. Specifically, we learn to select a diverse ensemble of planning strategies to be executed independently in parallel, and then pick the best solution among them. Since there is a fair deal of uncertainty about the efficacy of a planning algorithm, allowing for such redundancies at the expense of potentially additional planning effort has significant payoff.

1.3 Contribution

Our contributions in this paper are as follows:

1. We propose a novel trajectory planning architecture based on a 3 stage decoupling scheme to compute high quality solutions in real-time (Section 4).

2. We describe how to generate a library of expert planners for a helicopter. We describe how to train a meta-planner to choose from this library (Section 7).
3. We present extensive evaluation our approach in both simulation and over 700.4 hours of flight tests (autonomous 109.8 hours and pilot-in-the-loop 590.6 hours) that took place over 4 years. We provide detailed statistics of flight tests and discuss reasons for failures. We show how capabilities were developed and tested to remedy these failures. We also present analysis for each module in both simulation and real flights (Section 9).

This paper builds upon our previous works on most of the individual components of our framework: using an ensemble of planners (Choudhury et al., 2014; Choudhury et al., 2015), dynamics projection filter (Choudhury and Scherer, 2015), guaranteeing safety (Arora et al., 2014; Arora et al., 2015), training a dynamic list of planners (Tallavajhula and Choudhury, 2015) and smooth route optimization (Dugar et al., 2017a; Dugar et al., 2017b). In this paper, for the first time we present a description of the unified architecture, details of the trajectory planning framework and extensive flight test evaluation of the whole system.

2 Related Work

We present related work on the unmanned rotorcrafts and on the individual components of the architecture - route optimization, trajectory planning and safety.

2.1 Unmanned rotorcrafts

Over the past decade, a significant amount of progress has been made towards full autonomy of both full-scale (large enough to transport people), category I, and medium-scale (can carry at least 10 kg), category II unmanned rotorcraft (Kendoul, 2012). Efforts are underway to make such systems for full-scale versions available for both military and commercial applications (Staff, 2017; Whittle, 2016; Johnson, 2017). Since such systems are capable of moving at significantly faster speeds and over longer distances than their MAV (micro aerial vehicle) cousins, they present unique challenges otherwise not encountered in smaller craft. In addition, flying outside poses the additional challenge of ensuring vehicle safety in adverse weather conditions, such as strong winds.

The level of autonomy for any given system ranges from providing visual guidance to the pilot (Zimmermann, 2016; Lantzch et al., 2012), which is typically performed in full sized rotorcraft, to full control of the platform. Our approach much like (Whalley et al., 2016) is configurable for both modes of autonomy.

While some systems are focused on above ground flight planning (Whalley et al., 2016; Johnson and Mooney, 2014; Scherer et al., 2008; Wzorek et al., 2006; Pettersson and Doherty, 2004; Vachtsevanos et al., 2005), others are solely focused on the approach to land phase of flight (Taamallah et al., 2017; Zimmermann, 2016; Yomchinda et al., 2011). Our approach is similar to (Fabiani et al., 2007; Lantzch et al., 2012; Scherer et al., 2012) in that it plans through all phases of flight (from takeoff to land) for a full sized rotorcraft.

Some approaches rely on a single planner to navigate the aerial vehicle through its entire route (Johnson and Mooney, 2014; Scherer et al., 2012; Hrabar, 2008). To address computational intractability encountered using a single planner/configuration over larger distances, the planning problem is usually divided between a coarse (waypoint path) global planner and a finer (smooth trajectory) local planner. Typically the global plan is either computed offline or is provided by an end user, while the local planner handles re-planning of fine trajectories when newly observed obstacles are encountered (Whalley et al., 2016; Fabiani et al., 2007; Wzorek et al., 2006). In the event that a large change in feasible volume to fly in is encountered, or the goal has changed mid-flight, other approaches perform both levels of planning online (Lantzch et al., 2012; Scherer et al., 2008; Vachtsevanos et al., 2005; Pettersson and Doherty, 2004). Our approach is a mix between the last two categories as we compute a smooth, global trajectory (through the route guide optimizer) over a provided waypoint path (route) online while planning locally with similar fidelity. Similarly, rather than fit a single planning algorithm/configuration to address the variety of planning problems encountered by a UAS (Unmanned Aerial System) in flight, some approaches query multiple planners/configurations (Wzorek et al., 2006) to increase system robustness. To achieve the same ends, our approach also makes use of multiple planners through our planner ensemble.

Some systems have been developed specifically for the case in which the rotorcraft must make an emergency landing if it were ever in a state of autorotation (Taamallah et al., 2017; Yomchinda et al., 2011), which can be encountered either from engine failure or loss of tail rotor effectiveness. Others have been developed to consider nearby winds in order to ensure rotorcraft safety during nominal flight in the presence of strong winds (Lantzch et al., 2012; Scherer et al., 2012). In the event that the system fails to navigate the vehicle away from impending collision with terrain or similar obstacles, some methods use a backup system to slow the helicopter down to a stop (Scherer et al., 2008; Wzorek et al., 2006; Pettersson and Doherty, 2004). While such methods are appropriate for category II rotorcraft and below, when flying at altitudes appropriate for hover, they are not an option for full sized craft engaging in flight modes similar to fixed wing craft. Our approach makes use of a trajectory executive to ensure the commanded trajectory is safe for the vehicle to follow (within previously observed space) without requiring the vehicle to stop.

Our system focuses solely on the generation of safe trajectories made feasible for full sized rotorcraft, as part of the fully autonomous TALOS system (Paduano et al., 2015) with perception handled in (Stambler et al., 2016).

2.2 Route optimization

The problem of generating feasible trajectories for UAVs has previously been explored in the literature. (Anderson et al., 2005) builds on the Dubins solution and uses fixed-radius arcs to link straight segments with constant speed. (Jung and Tsiotras, 2008b) proposes an online, corridor-constrained smoothing algorithm that uses B-spline templates to generate paths, but not time profiles. Sampling-based techniques like (Frazzoli et al., 2002; Karaman and Frazzoli, 2010) are quite popular, but do not scale well with problem size. There has also been some work that deals with trajectory optimization in the presence of wind. The classic Zermelo–Markov–Dubins problem has been studied in (Bakolas and Tsiotras, 2010; Techy, 2011; Bakolas and Tsiotras, 2013) to characterize optimal solutions, but they use sharp turns and constant speed, neither of which are practical. (McGee et al., 2005) also uses a bounded turning radius assumption to yield minimum-time trajectories with constant speed. (Techy et al., 2010) uses a bounded roll-rate to construct smooth, continuous-curvature paths between two states in the presence of wind. However, it again assumes constant speed and does not provide a mechanism to extend the method to variable-speed trajectories. Since practical trajectory planning problems are extremely hard, it is often necessary to decouple the problem into an initial path-finding stage and a subsequent velocity-optimization process (Choset, 2005; Bobrow et al., 1985; Hwan Jeon et al., 2013)). We use concepts from previous work done on optimizing velocity profiles given a fixed path and a finite set of velocity bottlenecks ((Lipp and Boyd, 2014; Verscheure et al., 2009)) to compute time-optimal velocity profiles.

2.3 Trajectory planning algorithms

Motion planning has a rich and varied history (LaValle, 2006). A substantial body of work has looked at tractable approximate algorithms to solving nonholonomic motion planning problems encountered by mobile robots (Laumond, 1986). Curvature bounded path planning in presence of obstacles was proved to be NP-Hard (Reif and Wang, 1997). Approximation algorithms for finding such paths were proposed by (Jacobs and Canny, 1989) with exact results obtained only for obstacles with bounded curvature (Boissonnat and Lazard, 1996). These early works provide a glimpse into the inherent hardness of the problem and the need for good approximations to get high quality real-time solutions.

Sampling based planners (LaValle, 1998) have enjoyed a lot of attention due to their simplicity and asymptotic guarantees (Karaman and Frazzoli, 2013). They are also amenable to a lot of modifications to improve real-time performance such as biased sampling (Urmson and Simmons, 2003; Hsu et al., 2005), lazy evaluation (Hauser, 2015; Gammell et al., 2015; Bohlin and Kavraki, 2000) or hybrid local global optimization (Choudhury et al., 2016; Raveh et al., 2011; Luna et al., 2013). However, a large number of such sampling based methods require solving the boundary value problem online, i.e. finding a steering input to drive the system from one configuration to another. For curvature bounded systems, analytical solutions were obtained by Dubins (Dubins, 1957) and (Reeds and Shepp, 1990). In general however, the boundary value problem can be expensive to solve.

While the original RRT (LaValle, 1998) algorithm required a local planner to connect states, the EST method introduced shortly after by (Hsu et al., 1999) planned by directly forward integrating random control inputs. However, this method does require careful parameter selection and quality cannot be assured. Recently, the SST method (Li et al., 2015) was developed providing almost sure asymptotic convergence to an approximately optimal solution without requiring a local planner. Another method, Generalized Label Correcting (Paden and Frazzoli, 2016) was developed

recently with further improvements. While this remains an active area of research, these methods are still quite a bit away from solving nonholonomic problems in real-time.

Another class of approach has been the use of a state lattice as proposed by (Pivtoraiko et al., 2009). This is a lattice where edges between vertices are solved for offline using BVP solvers. These edges are called motion primitives. This acts as a wall of separation between the vehicle dynamics that create the graph and algorithms that can search the graph. This has been used to great success by (Likhachev and Ferguson, 2009; Lindemann and LaValle, 2006; MacAllister et al., 2013; Dolgov et al., 2010; Hwangbo et al., 2007; Heng et al., 2011). However, this process does require a fair bit of engineering in terms of lattice design, selecting a resolution, selecting a search method. These methods do not naturally adapt to changing dynamics or changing environment although progress has been made in this area by (Howard, 2009).

Local trajectory optimization methods focus on improving an initial suboptimal path towards a local optimum (Zucker et al., 2013; Schulman et al., 2013). A class of these methods, such as differential dynamic programming (Mayne, 1966) can satisfy dynamic constraints exactly while minimizing a second order approximation of the objective. To handle non-differentiable objectives, one can also employ evolutionary strategies effectively (Kalakrishnan et al., 2011; Kobilarov, 2012). While optimization can occasionally switch between topologically close classes, these methods are generally limited to the homotopy class of the initial path. Hence we conclude that nonholonomic problems can only be approximately solved, the quality of the solution heavily dependent on the type of approximation applied. Moreover, the approximation is influenced by the dynamics of the robot and the environment in which its operating in.

2.4 Guaranteed safe planning

The idea of computing invariant sets (Blachini, 1999) (i.e sets in which a system stays forever) to ensure safety or stability of finite-horizon plans is not new. In control theory, invariant sets are used to prove stability of model predictive control (also called receding horizon control), as shown in (Michalska and Mayne, 1993). In (Kerrigan and Maciejowski, 2000) a *nonlinear model predictive control* framework is presented that guarantees nominal feasibility using invariant sets by providing necessary and sufficient conditions on the control horizon, the prediction horizon and the constraint set. Receding horizon control for UAVs based on mixed integer linear programming is presented in (Schouwenaars, 2006). Loiter circles are added to the mixed integer linear programming formulation to ensure safety for an infinite time horizon. A library of precomputed emergency loiter circles for a full-scale autonomous helicopter is suggested in (Arora et al., 2014). The emergency maneuvers are optimized regarding their path diversity, i.e the difference of paths according to some measure.

An alternative is to directly find regions in which a collision is inevitable. The goal of the controller is to avoid those *regions of inevitable collision* (LaValle and Kuffner Jr, 2001), which are defined as the union of *inevitable collision states* (Fraichard and Asama, 2004), within its finite planning horizon. Most of the work on inevitable collision has focused on static environments, however, the concept has already been applied in dynamic environments for car-like vehicles (Parthasarathi and Fraichard, 2007; Althoff et al., 2012). Additionally, in (Althoff et al., 2011; Bautin et al., 2010) an extension for stochastic environments is presented taking into account the uncertain motion prediction of obstacles in the workspace, which calculates the probability that a state is an inevitable collision state. Other related work deals with the problem of the existence of an infinite long collision-free trajectory. This problem is discussed in (Karaman and Frazzoli, 2012) by the means of the ergodic forest scenario by finding criteria such as the size of obstacles, the distribution of obstacles and the maximum velocity of the robot under which almost surely an infinite time trajectory exists.

3 Problem Definition

3.1 Notation

The planner uses a *simplified dynamics model* that can be tracked by the closed-loop controller (which in turn uses a high-fidelity model). This simplified model was provided to us by the control system designers. This simple model imposes a set of constraints on the trajectory. We define a trajectory, $\sigma(t)$ as a smooth mapping from time to a position (x, y, z) and heading (ψ) (represented as a 5th order spline)

$$\sigma(t) : [0, t_f] \rightarrow \mathbb{R}^3 \times SO(2) \quad (1)$$

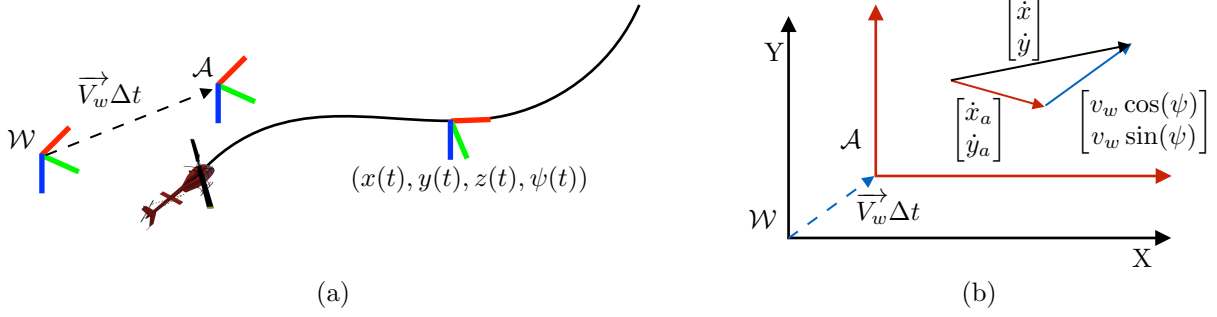


Figure 4: (a) \mathcal{W} is the fixed reference frame. \mathcal{A} is the moving airframe which has a time varying offset $\vec{V}_w \Delta t$ from \mathcal{W} , where \vec{V}_w is the wind speed vector. (b) The relation between velocity vectors in the two frames. A velocity vector in the airframe \mathcal{A} is $[\dot{x}_a \ \dot{y}_a]^T$. To express the velocity in ground frame \mathcal{W} , a wind vector is added to get $[\dot{x} \ \dot{y}]^T$.

We follow the North-East-Down coordinate convention as shown in Fig. 4(a). The position and heading is defined with respect to a fixed (ground) reference frame \mathcal{W} . However, the dynamics model is specified in the airframe \mathcal{A} . The airframe is a *moving reference frame* and is linked to the ground frame by a wind displacement vector which varies with time. This is illustrated in Fig. 4(b). When specifying configurations in the airframe, we will use the following notation (x_a, y_a, z_a, ψ_a) .

3.2 Input and output for the trajectory planner

The input to the trajectory planner is a *mission* that consists of 6 main components:

1. *Route*: Sequence of 3D waypoints. Each pair of waypoints has a desired speed and margins for a *flight corridor*, a valid region of operation. The route also contains a set of no-fly-zones where the system cannot enter.
2. *Start state*: This is either the current pose of the vehicle or the reference point on the trajectory that is currently being tracked by the controller. It serves as the initial boundary value for the trajectory.
3. *Dynamics constraints*: Set of equality and inequality constraints due to actuator limits of the control system and flight performance charts.
4. *Obstacle map*: Online map from sensor data.
5. *Touchdown sites*: A set of potential touchdown locations (x, y, z, ψ) provided by the perception system.
6. *Wind Conditions*: The wind speed and direction that is a parameter in the constraints.

The output of the planner is a time parameterized trajectory. We delve further into the dynamics constraints, the safety constraints and the objective.

3.3 Dynamics constraints

Let $\mathcal{F}_{\text{dyn}}(\sigma) = 0$ and $\mathcal{H}_{\text{dyn}}(\sigma) \leq 0$ be the constraints. The system can be modelled differently based on the speed regime - this is to do with the control authority of the tail rotor.

When the speed is below a critical threshold, $\|[\dot{x}(t) \ \dot{y}(t) \ \dot{z}(t)]\|_2 < v_{\text{crit}}$, the system is modelled as a composition of single and double integrator systems. These are $|\dot{x}(t)|, |\dot{y}(t)| \leq v_{\text{max}}, |\dot{z}(t)| \leq v_{z,\text{max}}, |\ddot{x}(t)|, |\ddot{y}(t)| \leq a_{\text{max}}, |\ddot{z}(t)| \leq a_{z,\text{max}}, |\ddot{x}(t)|, |\ddot{y}(t)| \leq a_{\text{max}}, |\ddot{x}(t)|, |\ddot{y}(t)| \leq j_{\text{max}}, |\ddot{z}(t)| \leq j_{z,\text{max}}$ and $|\dot{\psi}(t)| \leq \dot{\psi}_{\text{max}}$

For our application, we used the following values: $v_{\text{crit}} = 10$ m/s, $v_{\text{max}} = 51.44$ m/s, $v_{z,\text{max}} = 5.07$ m/s, $a_{\text{max}} = 0.49\text{m/s}^2$, $a_{z,\text{max}} = 0.49\text{m/s}^2$, $j_{\text{max}} = 0.98\text{m/s}^3$, $j_{z,\text{max}} = 0.98\text{m/s}^3$ and $\dot{\psi}_{\text{max}} = 0.26$ rad/s.

When $\|[\dot{x}(t) \ \dot{y}(t) \ \dot{z}(t)]\|_2 \geq v_{\text{crit}}$, we can model the system as a fixed wing model moving in airframe that changes its heading by rolling.

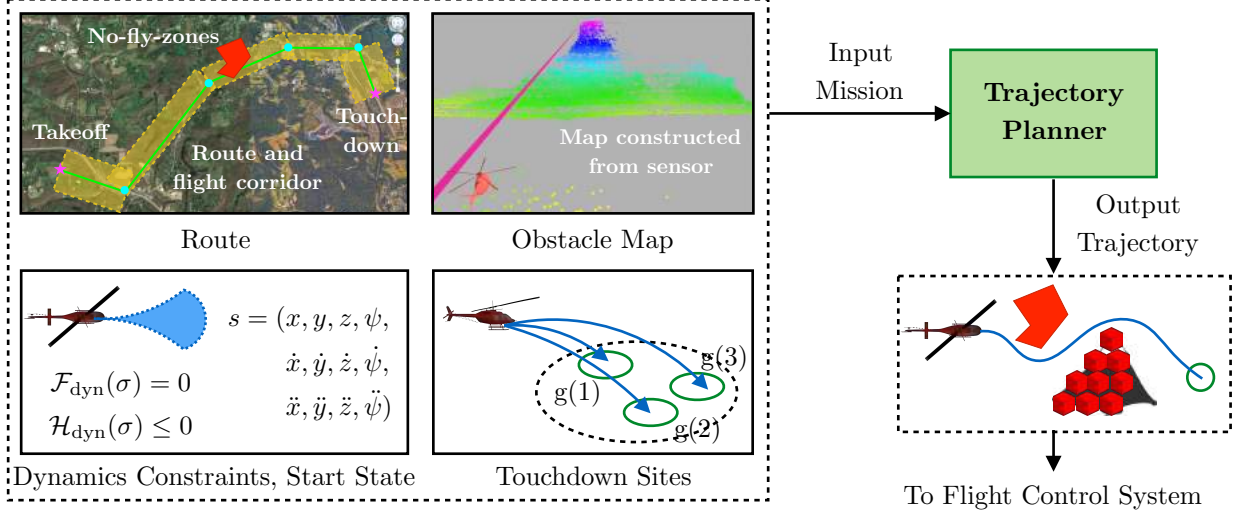


Figure 5: Overview of the input and output to the trajectory planner. The input is a mission. The first component of a mission is a route which comprises of waypoints, flight corridors and no-fly-zones. The second component is a map of the world constructed based on sensor measurements. The third component is a set of dynamic constraints that the trajectory must satisfy and the start state which serves as a boundary value condition. The final component is a set of touchdown sites that is updated dynamically as the sensor scans the landing zone. The output of the module is a trajectory. This trajectory is sent to the flight control system which attempts to track it.

$$\left. \begin{aligned} \dot{\psi}_a(t) &= \frac{g \tan \phi_a(t)}{v_a(t)} \\ \dot{x}_a(t) &= v_a(t) \cos \psi_a(t) \\ \dot{y}_a(t) &= v_a(t) \sin \psi_a(t) \end{aligned} \right\} \begin{array}{l} \text{Dynamics in} \\ \text{Airframe} \end{array} \quad \left. \begin{aligned} \dot{x}(t) &= \dot{x}_a(t) + v_w \cos(\psi_w - \psi_a) \\ \dot{y}(t) &= \dot{y}_a(t) + v_w \sin(\psi_w - \psi_a) \\ \psi(t) &= \tan^{-1} \left(\frac{\dot{y}(t)}{\dot{x}(t)} \right) \end{aligned} \right\} \begin{array}{l} \text{Airframe to} \\ \text{Ground Frame} \end{array} \quad (2)$$

We have inequality constraints on the derivatives: $|v_a(t)| \leq v_{\max}$, $|\dot{v}_a(t)| \leq a_{\max}$, $|\ddot{v}_a(t)| \leq j_{\max}$, $|\phi_a(t)| \leq \phi_{\max}$, $|\dot{\phi}_a(t)| \leq \dot{\phi}_{\max}$, $|\ddot{\phi}_a(t)| \leq \ddot{\phi}_{\max}$, $|z(t)| \leq v_{z,\max}$, $|\dot{z}(t)| \leq a_{z,\max}$ and $|\ddot{z}(t)| \leq j_{z,\max}$. For our application, we used the following values: $v_{\max} = 51.44$ m/s, $v_{z,\max} = 5.07$ m/s, $a_{\max} = 0.49\text{m/s}^2$, $a_{z,\max} = 0.49\text{m/s}^2$, $j_{\max} = 0.98\text{m/s}^3$, $j_{z,\max} = 0.98\text{m/s}^3$, $\phi_{\max} = 0.44\text{rad}$, $\dot{\phi}_{\max} = 0.17\text{rad/s}$, $\ddot{\phi}_{\max} = 0.44\text{rad/s}^2$.

There are some additional constraints arising from flight performance charts. Fig. 6(a) illustrates the height-velocity chart which corresponds to a set of regions that the height above ground (derived from $z(t)$) and airspeed $v_a(t)$ cannot enter. Fig. 6(b) shows the torque limit constraint. The torque drawn from the motor depends on the airspeed $v_a(t)$ and the climb rate $-\dot{z}(t)$. The total torque cannot exceed a specified limit (we use 90%). Fig. 6(c) illustrates the autorotation limit constraint. It shows the descent rate $\dot{z}(t)$ and airspeed $v_a(t)$ combination that is invalid. Fig. 6(d) shows the loss of tail rotor effectiveness (LTE) chart which is a constraint on $v_a(t)$, and the relative wind heading ψ_w .

3.4 Route constraints

Given a route, let $\text{corr}_i(\sigma(t)) \rightarrow \{0, 1\}$ describe an indicator function to specify whether a point in the trajectory lies inside a flight corridor. Each flight corridor is also associated with a velocity limit $v_{\text{seg},i}$. The constraints are as follows:

$$\sum_{i=0}^{n-1} \text{corr}_i(\sigma(t)) > 0 \quad \text{s.t.} \quad \text{corr}_i(\sigma(t))v_a(t) \leq v_{\text{seg},i} \quad \forall i \in \{1, \dots, n-1\} \quad (3)$$

Fig. 7 illustrates these route constraints.

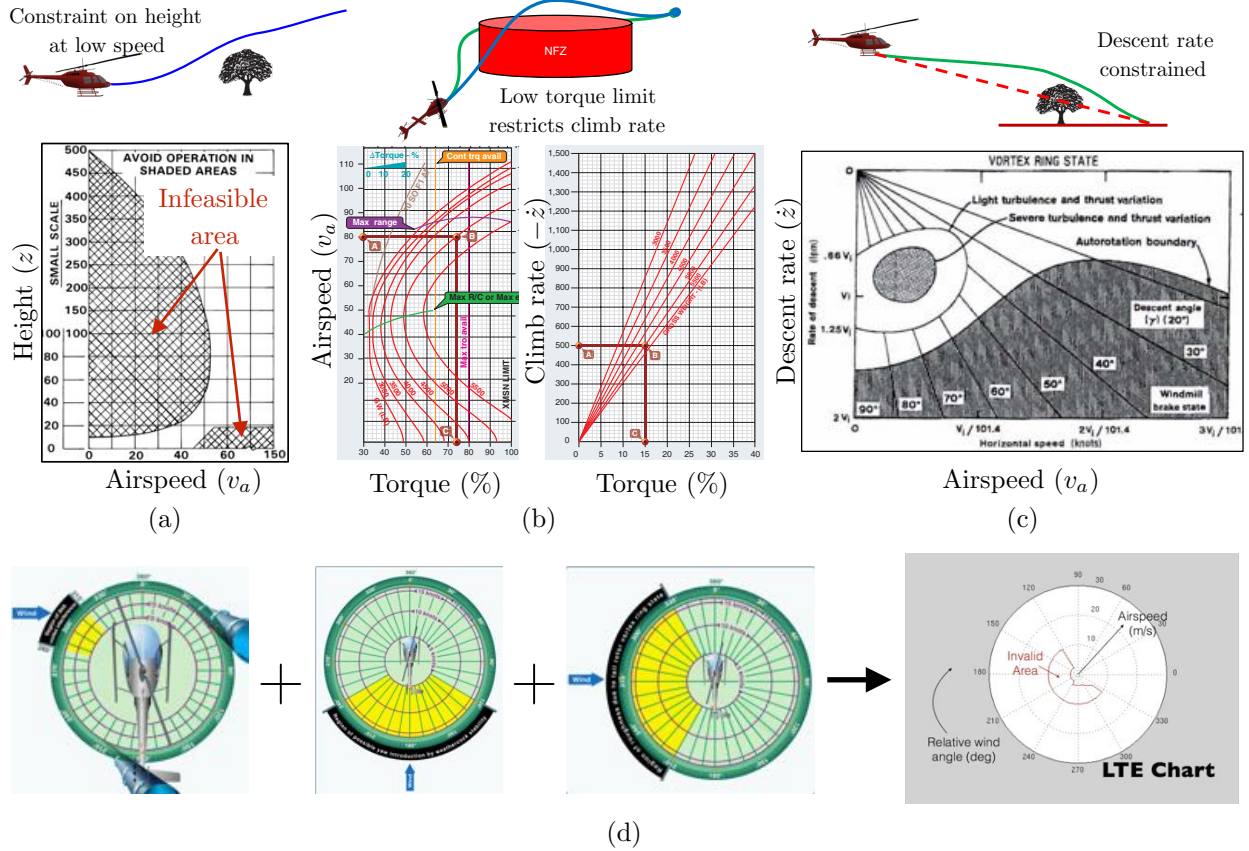


Figure 6: Dynamics constraints from performance charts of a helicopter (FAA, 2012). (a) The height-velocity chart (b) The torque limits chart (c) The autorotation limits chart (d) The loss of tail rotor effectiveness (LTE) chart

3.5 Safety constraints

We now define a set of safety constraints $\sigma(t) \in \Sigma_{\text{valid}}$ for obstacle and NFZ avoidance (Fig. 8).

We use *time to collision* to measure how close the system is to obstacles. This objective scales with speed and velocity direction. Let \mathbf{p} be the position and \mathbf{v} be the velocity vector. Let $d_{\text{obs}}(\mathbf{p})$ be the distance to an obstacle. The time to collision is:

$$\text{ttc}(\mathbf{p}, \mathbf{v}) = \frac{\|d_{\text{obs}}(\mathbf{p})\|_2}{\|\mathbf{v}\|_2} \times \text{Inflation}_{XY} \times \text{Inflation}_Z \quad (4)$$

where $\text{Inflation}_{XY} \geq 1$, $\text{Inflation}_Z \geq 1$ is an inflation applied due to difference between the velocity vector and obstacle direction in XY and Z respectively. The exact expression and algorithm for computing time to collision is specified in Appendix A. This is used to define the constraint $\text{ttc}(\mathbf{p}, \mathbf{v}) \geq t_{\text{coll}, \text{min}}$.

A no-fly-zone (NFZ) is an unsafe volume Fig. 8(b)). It is defined as a 2D polygon with a lower and upper height.

3.6 Trajectory planning problem

The objective is to minimize the travel time $J(\sigma) = t_f$ which leads to the problem:

Problem 1 (Trajectory Planning Problem). *The trajectory planning problem is then formally defined as the search for the trajectory, σ^* , that minimizes the objective function, while satisfying boundary and trajectory wide constraints*

$$\begin{aligned} \min_{\sigma} \quad & J(\sigma) \\ \text{s.t.} \quad & \sigma(0) = s, \sigma(t_f) = g \\ & \mathcal{F}(\sigma) = 0, \mathcal{H}(\sigma) \leq 0, \sigma(t) \in \Sigma_{\text{valid}} \forall t \in [0, t_f] \end{aligned} \quad (5)$$

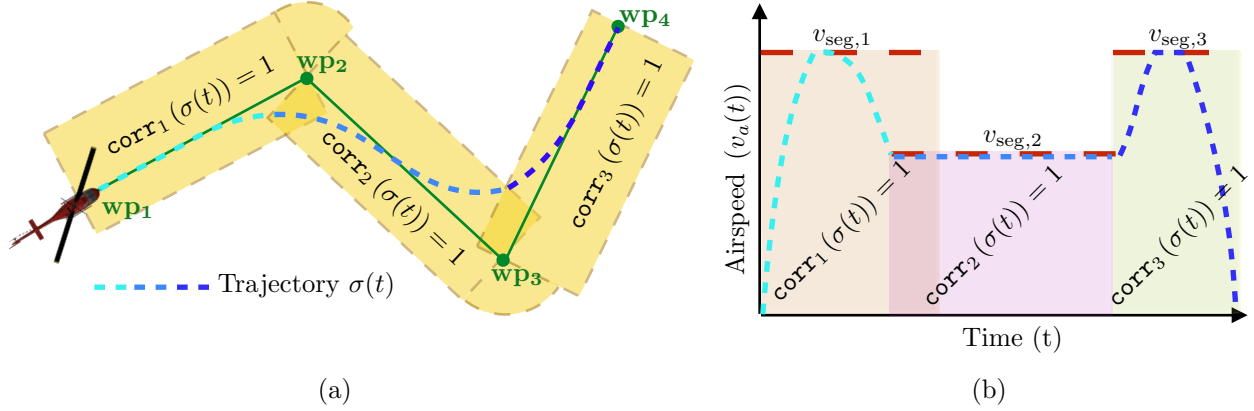


Figure 7: Illustrations of route constraints. (a) A route which defines a set of 3 safe corridors (yellow rectangles) $\text{corr}_1, \text{corr}_2, \text{corr}_3$. A feasible trajectory $\sigma(t)$ must lie inside all corridors. (b) A route also defines velocity constraints along the segments. The air speed of the trajectory $v_a(t)$ must not exceed constraint velocities $v_{\text{seg},1}, v_{\text{seg},2}, v_{\text{seg},3}$.

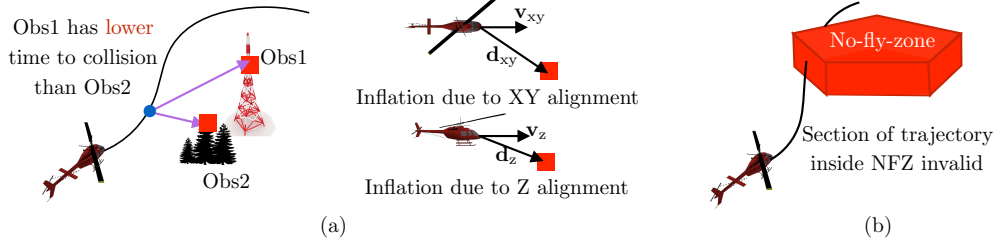


Figure 8: Illustration of safety constraints (a) The time to collision constraint penalizes proximity to obstacles scaled by the speed and direction of flight. It is an approximation of the minimum time it takes for a state to collide with an obstacle. Obs1, the tower, has a lower time to collision as the velocity vector is pointed at it. Obs2, although closer has a higher time to collision as it is unlikely that the system is likely to hit it. This can be captured by inflating time to collision in both XY and Z depending on the dot product between the velocity vector and the direction to the obstacle. Appendix A provides a mathematical formula. (b) Constraints due to no-fly-zones which are specified as polygons with a minimum and maximum height.

$\sigma(0) = s, \sigma(t_f) = g$ is the initial and final boundary value constraint, $\mathcal{F}(\sigma) = 0$ and $\mathcal{H}(\sigma) \leq 0$, are the combination of both dynamics and the route constraints. The final constraint is the safety constraint $\sigma(t) \in \Sigma_{\text{valid}}$.

Problem 5 is a complex non-convex optimization problem that scales across a large distance. In the next section, we will talk about how we approximately decompose the problem in order to solve it in real time.

4 Planning Architecture

The block-diagram of our trajectory planning architecture is shown in Fig. 9. The key idea is to decompose the overall problem into 3 cascaded sub-problems as shown in Fig. 10. These sub-problems operate at different resolutions, different time-periods and focus on different aspects of the optimization problem.

1. The first stage is a global planner which is responsible for producing a route guide trajectory $\sigma_{\text{rg}}(t)$ that has guarantees with respect to the overall mission such as ensuring the vehicle can stay in a designated safe flight corridor, respect speed limits along segments and can feasibly transition between corridors in presence of wind. It is computationally expensive and is invoked a few times per mission.
2. The second stage is a receding horizon trajectory planner that is responsible for locally repairing the trajectory

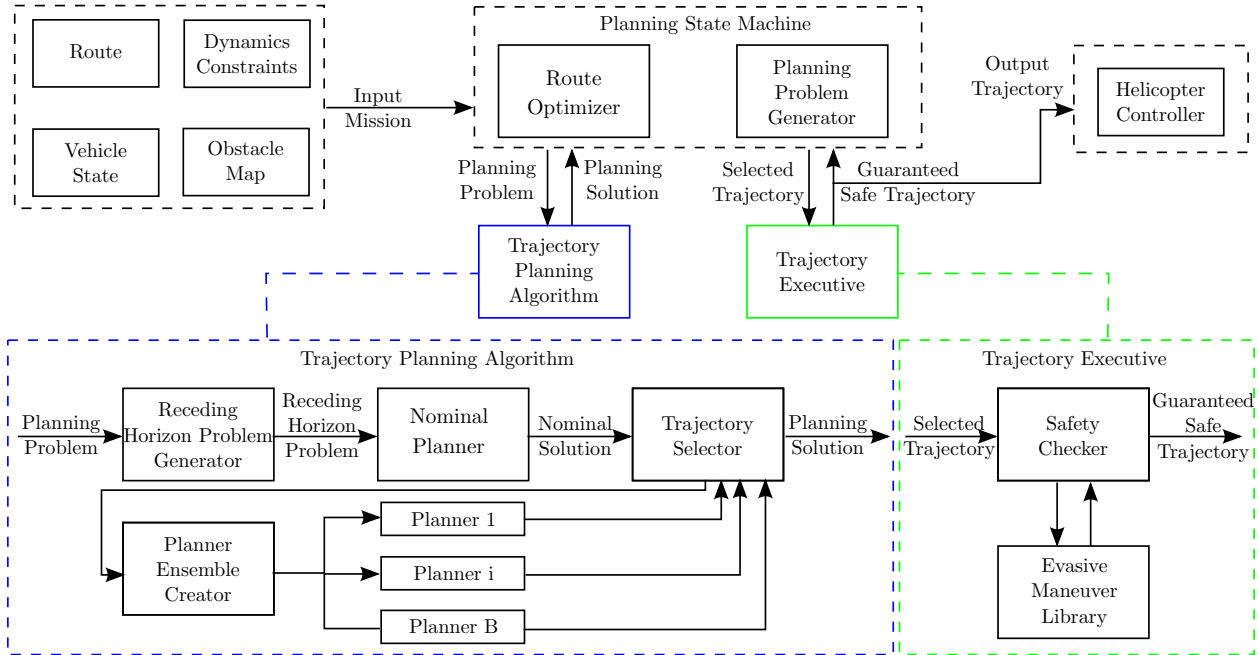


Figure 9: The trajectory planner block diagram. It consists of 3 main modules - the planning state machine, the trajectory planning algorithm and the trajectory executive. The planning state machine takes as input the mission and processes it to create a planning problem that is sent to the trajectory planning algorithm. The planning algorithm returns a planning solution to the state machine. The state machine then sends this solution to the trajectory executive which checks for safety and returns a guaranteed safe trajectory which is then sent to the helicopter controller.

to avoid sensed obstacles. It takes as input the route guide trajectory and obstacle constraints and repairs the trajectory locally up to a finite horizon (few km) to produce $\sigma_{\text{hor}}(t)$. This runs periodically (~ 1 Hz) to ensure obstacles are avoided shortly after they are sensed.

3. The final stage is an executive module that takes as input the computed trajectory and ensures that it is guaranteed to be safe. It does this by ensuring that there exists an evasive maneuver that lies in known free space. The executive module is tasked with the job of checking if such maneuvers exist, and if not either slowing down the trajectory or in the worst case executing the evasive maneuver. The executive module reasons within the sensor horizon range and runs at the highest frequency (~ 10 Hz).

Whenever a decoupling based approach is used, a pertinent concern is deadlocks:

1. *Deadlock between route guide and finite horizon trajectory*: The scenario when a deadlock can potentially occur is when an obstacle appears such that no finite horizon trajectory exists. We overcome this scenario by ensuring that the goal point on the route guide is always in a feasible location sufficiently far away from obstacles. Hence as long as the flight corridors are wide enough, the trajectory planners should be able to find a path.
2. *Deadlock between finite horizon trajectory and guaranteed safe trajectory*: The scenario where a deadlock occurs is where the trajectory planner produces a solution to which no evasive maneuvers can be attached. The executive alleviates the situation by slowing down the trajectory. If that does not resolve the deadlock, the system enters the evasive maneuver and the sensor keeps on scanning and increasing the known free space. This increases the chance of the planned trajectory being verified to be safe.

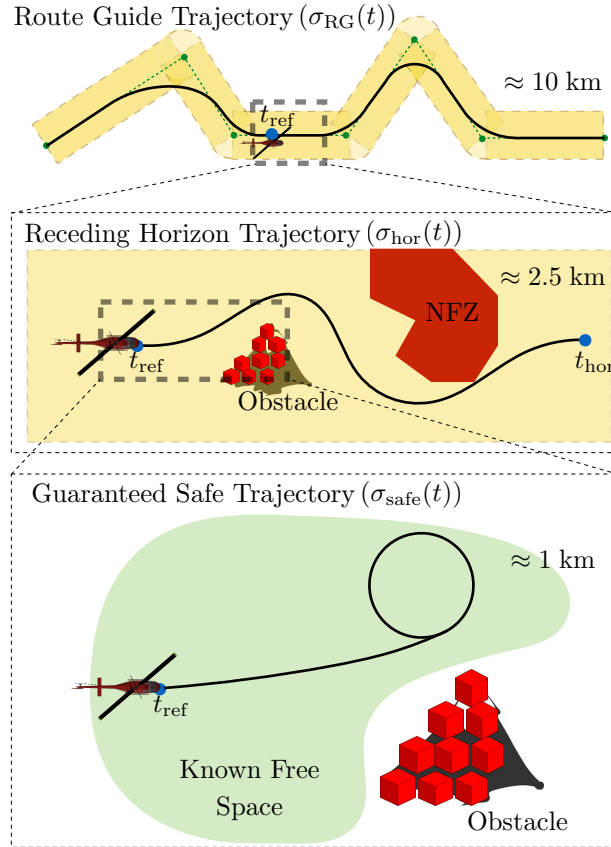


Figure 10: Three stage decomposition of the problem.

5 Planning State Machine

The main role of the planning state machine is to process the input mission and create a planning problem for the trajectory planning module to solve. In order to do so, it must capture the notion of *state*, i.e. the progress made along the mission so far. This implies that the other system components can be terminated and restarted, and the state machine will be able to provide all the information required to resume current operation. To gather such information, the state machine plays the crucial role of communicating with the flight control system and the perception system.

It invokes two main sub-modules as shown in Fig. 9 - the *route optimizer* and the *planning problem creator*.

1. *Route optimizer*: This module computes the route guide and is described in detail in Section 6.
2. *Planning Problem Generator*: A *planning problem* is a complete list of specifications sent to the trajectory planning module. It is defined by the following tuple $\Gamma_{pp} = \langle s, g, \mathcal{F}, \mathcal{H}, \Sigma_{\text{valid}}, \sigma_{\text{rg}} \rangle$.
 - (a) s - The start pose and velocity that the trajectory must satisfy
 - (b) g - The terminal touchdown point (set of touchdown points) coming from the mission specification
 - (c) \mathcal{F} and \mathcal{H} - The set of equality and inequality constraints as defined in Section 3.
 - (d) Σ_{valid} - The set of valid states.
 - (e) σ_{rg} - The route guide trajectory

The planning problem generator prepares this tuple and sends it along to the trajectory planning algorithm.

We now describe the states maintained by the state machine illustrated in Fig. 11. These states dictate when and what

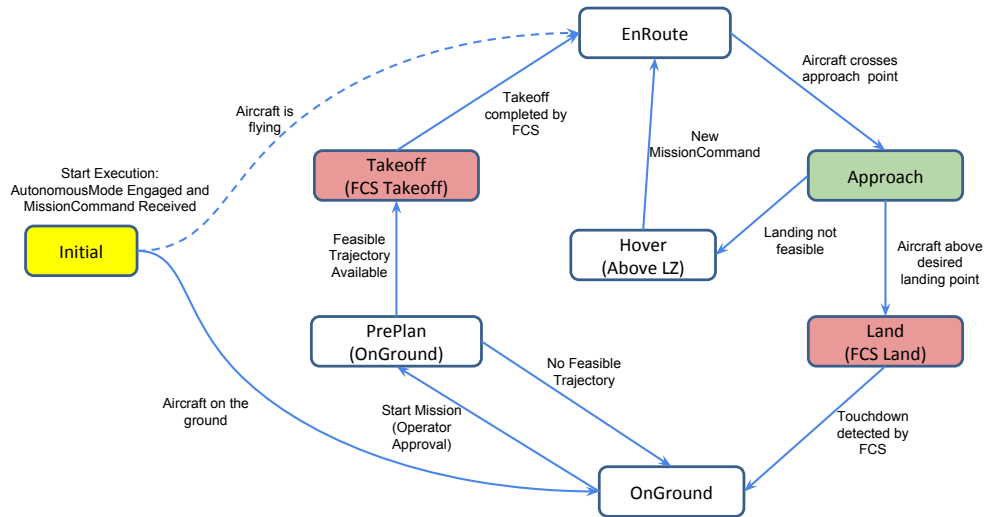


Figure 11: The different states of the planning state machine.

planning problem is sent to the trajectory planning algorithm. They also dictate when communications should be made to the control and perception system.

1. (Initial) When the system is up and running, the state machine waits for a valid mission as well as an engaged flag from the control system in order to start mission execution. The initialization process also waits for all the other systems to be ready.
2. (OnGround) The aircraft is sitting on the ground, with all systems ready, just waiting for human operator approval to start mission execution.
3. (PrePlan) This happens before the takeoff command is issued to the control system. The route guide optimizer is invoked to compute a route guide. Only when the mission is deemed feasible, does the state machine tell the control system to takeoff (go to hover).
4. (Takeoff) The state machine issues a takeoff command to the control system and waits until it is completed (aircraft gets off the ground). After this, the control system starts following the planned trajectories.
5. (EnRoute) This is the main execution state. While in this state, the trajectory planner algorithm will be constantly updating the generated trajectories, which are affected by dynamic wind estimates, detected obstacles, and popup NFZs.
6. (Approach) As the aircraft gets closer to the landing area, the system enters the Approach state. This state is composed by several sub-states that will control actions in several sub-systems to perform landing zone evaluation, planning to multiple touchdown point candidates, and touchdown point selection processes. The whole sequence is explained with more details next.
7. (Land) When the aircraft is above the desired touchdown point, a land command is issued to the control system which performs the touchdown process. After the land is completed, the aircraft returns to OnGround state and it is ready to execute another run.

One of the most complex execution sequences during a regular run is the approach to land sequence (Approach Phase). This complexity is due to the fact that it is necessary to monitor the approach to the landing zone and generate events to trigger changes in the sensor's scanning pattern, keep track of all available landing sites, provide information to the trajectory planning algorithm to generate paths to all valid landing sites, and interact with the mission executive to select the most appropriate landing site, or abort a landing, if no valid landing site is available. The sequence of events are essentially triggered by the distance to the landing site as illustrated in Fig. 12.

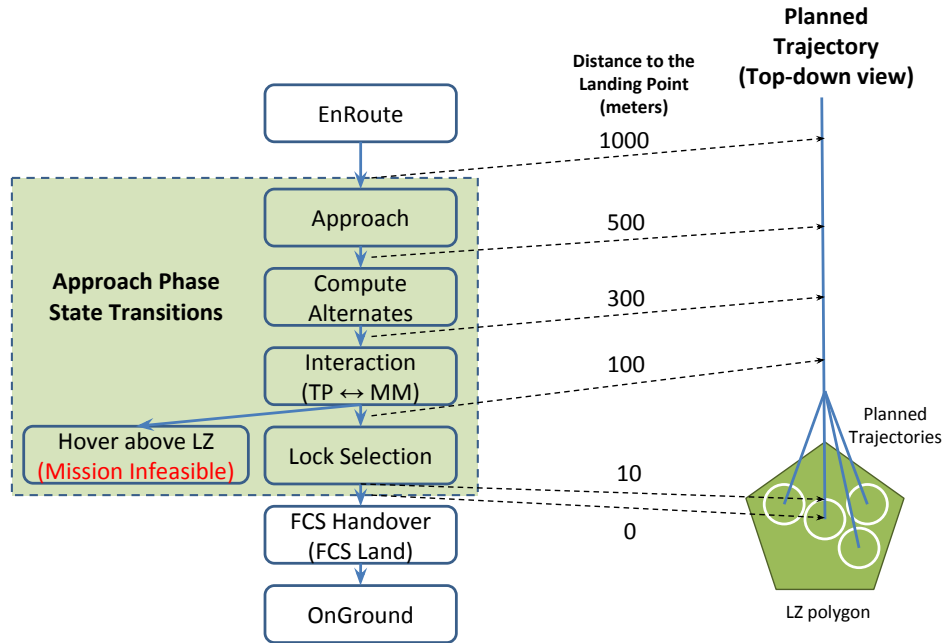


Figure 12: The sub states of the Approach state in the state machine. State transition events are mostly triggered by distance to the landing site.

1. (Approach) The state machine polls the perception system for a list of candidate touchdown points.
2. (Compute Alternates) As the aircraft gets closer to the landing zone, the perception system begins to return alternate touchdown points. These points are then used to update the planning problem. The list is updated constantly as obstacles are detected (small obstacles are only detected late in the approach process). The trajectory planning algorithm searches for feasible plans for each landing point.
3. (Interaction) The state machine interacts with the mission executive to select the touchdown point. It is important to note that the selected point can be changed until a selected point lock happens. If no valid candidate is available, the state machine issues a "MissionInfeasible" status to the mission executive commands the helicopter to hover above the LZ.
4. (Lock Selection) The selected touchdown point cannot be changed anymore, but the perception system can still detect an obstacle and remove the selected touchdown point from the valid candidates list. In this case too, the aircraft will be commanded to hover above the LZ and a "MissionInfeasible" is issued to the mission executive.
5. (FCS Handover) When the aircraft is above the selected touchdown point, the state machine sends a command to the control system to perform the touchdown process. The control system sends a control flag informing the state machine when the aircraft has completed the maneuver and is safe on the ground.

6 Route Optimizer - the KITE algorithm

In this section, we present κ ITE (Curvature (κ) parameterization Is very Time Efficient), our approach for solving the optimization problem to generate the route guide trajectory (Dugar et al., 2017a; Dugar et al., 2017b). As we discussed in Section 4, the route guide trajectory is the solution to the optimization problem 1 with the additional relaxation of the safety constraint $\sigma(t) \in \Sigma_{\text{valid}}$. The motivation is that such a solution will be a good heuristic for

the trajectory planning module which is primarily concerned with this safety constraint. By attempting to compute a trajectory that is dynamically (i.e. satisfying dynamic constraints like speed, acceleration and roll-rate) and spatially (i.e. satisfying flight-corridor and boundary value constraints) feasible along the entire length of the mission, the optimizer also ensures that the input mission is actually valid – this prevents the aircraft from attempting an unsafe mission.

The problem is still intractable as it involves non-convex dynamics constraints and flight-corridor constraints for an aircraft flying in wind. Due to wind, the aircraft operates in a moving frame of reference (the air frame) and experiences a time-dependent shift. Simultaneously satisfying dynamics constraints (such as limits on acceleration, jerk and roll-rate) that exist in the air frame and spatial constraints (such as the flight-corridor) that exist in the ground frame is challenging. Additionally, the problem is intrinsically multi-scale in nature - the mission could be several kilometres with long straight line stretches interleaved with sharp turns. Hence an approximation approach is required that exploits the structure of the problem to produce good solutions within a reasonable time-frame.

6.1 Approach

To make the solution-search tractable, we decouple the optimization problem into a path optimization and a velocity profile optimization (Choset, 2005; Bobrow et al., 1985; Hwan Jeon et al., 2013). The optimization approach, summarized in Fig 13, proceeds in 4 stages:

1. Phase A: *Path Optimization*. This phase solves for a path that is *guaranteed to be feasible* (in terms of dynamics and route constraints) for a range of constrained velocity profiles. The path is parameterized as a sequence of *sections* which are either straight lines or arcs. The optimizer solves for each section i independently along with a corresponding $V_{\text{lim},i}$, such that the section is feasible for any velocity profile limited by $V_{\text{lim},i}$. In the interest of time-optimality, the objective of this optimizer is to maximize the velocity limit $V_{\text{lim},i}$ while keeping the total arclength of the section small.
2. Phase B: *Velocity Optimization*. This phase optimizes velocity at specific *control points* at the end of the sections to minimize time. By ignoring jerk constraints at this stage and assuming a trapezoidal velocity profile, we are able to solve this optimization very efficiently.
3. Phase C: *Velocity Spline Fitting*. This phase solves for smooth velocity splines that introduce jerk limits.
4. Phase D: *Ground Frame Trajectory Repair*. This phase combines the path from Phase A with the velocity profile from Phase C to yield the final ground frame trajectory.

For simplicity of exposition, we drop the z coordinate from our formulation and note that the Z-profile is solved independent of the X-Y profile while accounting for wind in a similar manner as described in later sections.

6.2 Phase A: Path Optimization

Let $\xi(\tau) = \{x(\tau), y(\tau), \psi(\tau)\}$ be a *path* defined over $\tau \in [0, 1]$. The first stage of the algorithm solves for a ground-frame path $\xi_W(\tau) = \{x_W(\tau), y_W(\tau), \psi_W(\tau)\}$, $\tau \in [0, 1]$ that respects corridor constraints. Path optimization is a challenging problem because of a number of reasons - it must guarantee that the path will be dynamically feasible when the velocity profile is determined subsequently and reason about time optimality. Moreover, the presence of wind as a forcing function breaks the necessary decoupling between path and time, and must be dealt with in a principled manner. Our solution structure addresses these concerns.

6.2.1 Parameterization

Solving for arbitrary path shapes is intractable, especially with lots of waypoints and large segment lengths. We restrict our solution to the space of ground-frame straight lines ξ_{st} and arcs ξ_{arc} with smooth curvature profiles, where the path is a sequence $\{\xi_{\text{st}}^1, \xi_{\text{arc}}^1, \xi_{\text{st}}^2, \xi_{\text{arc}}^2, \dots\}$. Arc end-points of ξ_{arc}^i lie on the lines defined by waypoints (wp_i, wp_{i+1}) and (wp_{i+1}, wp_{i+2}) respectively. This parameterization also scales well with problem size. Instead of searching for individual points along the path, only the arcs need to be explicitly determined, and the straight segments simply connect their endpoints.

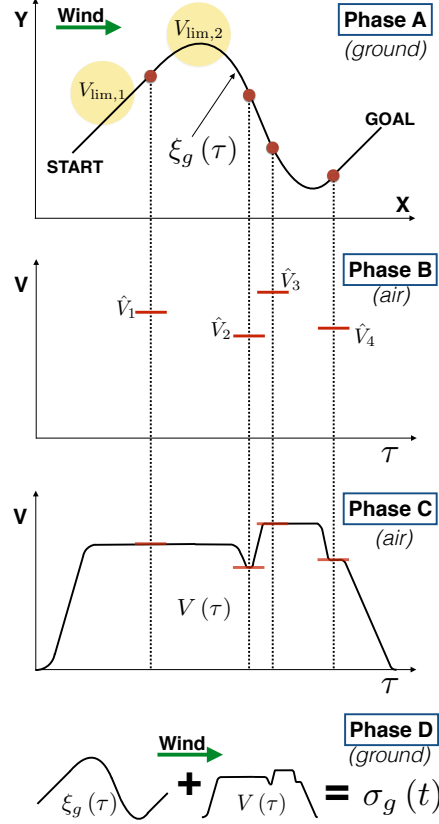


Figure 13: Overview of the four phases in the route optimizer. Phase A solves for a feasible path for any velocity profile limited by desired segment velocities. Phase B backs out speed constraints at certain control points. Phase C solves for a smooth velocity profile. Phase D combines the path from Phase A and the velocity from Phase C.

6.2.2 Arc Optimization

Arcs are determined for every ordered triple of waypoints $(wp_{i+1}, wp_{i+2}, wp_{i+3})$. Limits on ϕ (roll), $\dot{\phi}$ and $\ddot{\phi}$ become active along the arc, and must be satisfied. These constraints directly limit the arc's curvature κ and its derivatives with respect to arclength (κ', κ'') . It is essential to note that these curvature limits are *not invariant* - it can be shown that they also depend on airspeed and acceleration according to the function $\text{CurvLimit}(v_{\max}, \phi_{\max}, \dot{\phi}_{\max}, \ddot{\phi}_{\max}, a_{\max})$:

$$\kappa_{\max} = \frac{g \tan \phi_{\max}}{v_{\max}^2}, \quad \kappa'_{\max} = \frac{\alpha}{\gamma^3} - \frac{\beta}{\gamma^2}, \quad \kappa''_{\max} = \frac{g \ddot{\phi}_{\max}}{v_{\max}^4} \quad (6)$$

$$\left(\alpha = g \dot{\phi}_{\max}, \quad \beta = 2\kappa_{\max} a_{\max}, \quad \gamma = \frac{3\alpha}{2\beta} \right)$$

Solving for a curvature profile that respects these limits guarantees that the path will be dynamically feasible when the velocity profile is subsequently optimized with the same limits on airspeed and acceleration, which allows us to effectively *decouple solving for the path and velocity in a principled way*. These dynamic limits are defined in the air-frame; therefore, our approach solves for curvature profiles in the air-frame and projects them into the ground-frame.

Since curvature is constrained by airspeed, arcs function as velocity bottlenecks. To enforce time optimality, we determine the maximum airspeed at which we can perform a turn with the least curvature, while respecting corridor constraints. Finally, arcs are meant to carry out heading changes in the ground frame. To convert between ψ_W and ψ ,

we use the function `HeadingInAir` (ψ_W, v, v_w):

$$\psi = \text{acos}\left(-\frac{v_w \sin(\psi_W)}{v}\right) - \frac{\pi}{2} + \psi_W \quad (7)$$

Algorithm 1 captures this process, while the following subsections explain how we determine arcs that effect a required heading change.

6.2.3 Arc Parameterization

Let s be the arc-length along a path. Let $\xi(s)$ be an arclength parameterized path. Building on (Kelly and Nagy, 2003), each arc ξ_{arc}^i is represented as a C1 curvature spline $\kappa(s)$ comprising a degree-4 polynomial $\kappa_1(s)$, a constant-curvature section $\kappa_2(s) = \kappa_{\text{trans}}$ and another degree-4 polynomial $\kappa_3(s)$. S_f^1, S_f^2, S_f^3 denote the arclengths of the three sections, and S_f denotes the total arclength. At this stage of the algorithm, we assume a constant velocity $V_{\text{arc},i}$ along the arc to determine its ground-frame shape. This velocity serves as the upper limit for this segment in a subsequent velocity optimization. We can recover a ground-frame path $\xi_W(s)$ from $\kappa(s)$ using the function `CurvPolyGnd` ($\kappa(s)$):

$$\begin{aligned} \psi(s) &= \int_0^s \kappa(s) \\ x(s) &= \int_0^s \cos(\psi(s)) + \frac{v_w}{V_{\text{lim},i}} \int_0^s ds \\ y(s) &= \int_0^s \sin(\psi(s)) \\ \psi_W(s) &= \tan^{-1}\left(\frac{\dot{y}(s)}{\dot{x}(s)}\right) \end{aligned} \quad (8)$$

It is now trivial to carry out a change of index from $s \in [0, S_f]$ to $\tau \in [0, 1]$ by setting $\tau(s) = s/S_f$ and obtain $\xi_W(\tau)$.

To solve for one of the segments of $\kappa(s)$, we use `CurvPoly` ($\kappa_0, \kappa_f, S_f, \kappa_{\text{max}}, \kappa'_{\text{max}}, \kappa''_{\text{max}}$):

$$\begin{aligned} \text{find } & \kappa(s) \\ & \kappa(0) = \kappa_0, \kappa(S_f) = \kappa_f \\ & \kappa'(0) = 0, \kappa'(S_f) = 0 \\ & \kappa(s) \leq \kappa_{\text{max}}, \kappa'(s) \leq \kappa'_{\text{max}}, \kappa''(s) \leq \kappa''_{\text{max}} \end{aligned} \quad (9)$$

9 represents an optimization problem that solves for the coefficients of a polynomial $\kappa(s) = a + bs + cs^2 + ds^3 + es^4$ given boundary value and inequality constraints that are active in $[0, S_f]$. A finite sampling of the interval $[0, S_f]$ yields a system of linear inequalities $Ap \leq b$, where $p = [a, b, c, d, e]^T$. 9 can thus be represented and solved as a Linear Least Squares problem. For efficiency, we solve this optimization problem offline with a range of boundary values and constraints, and use the resulting curvature library for fast online lookup.

Algorithm 1, lines 4 to 15 highlight how the spline is constructed to satisfy the required $\Delta\psi$.

6.2.4 Final path and velocity limits

Once we have obtained all the arcs, we concatenate straight segments and arcs to yield the final ground-frame path $\xi_W(\tau)$. Each segment also has an airspeed bound imposed by Phase A for ξ_{arc}^i , resulting in a set of airspeed limits $V_{\text{lim}} = [V_{\text{st},1} \quad V_{\text{arc},1} \quad \dots \quad V_{\text{arc},N-2} \quad V_{\text{st},N-1}]$.

6.3 Phase B: Time Optimization

6.3.1 Time Optimization Problem

This phase determines an optimal scheduling of speeds along a finite set of control points belonging to $\xi_W(\tau)$. The $2N - 4$ control points are the start and end points of each turn segment, which divide $\xi_W(\tau)$ into a sequence of

Algorithm 1: ArcOpt ($wp_i, wp_{i+1}, wp_{i+2}, v_w$)

```

1 for  $v \leftarrow [v_{\max}, v_{\min}]$  do
2    $(\kappa_{\max}, \kappa'_{\max}, \kappa''_{\max}) \leftarrow \text{CurvLimit} \left( v, \phi_{\max}, \dot{\phi}_{\max}, \ddot{\phi}_{\max}, a_{\max} \right)$ 
3    $\Delta\psi \leftarrow \text{HeadingInAir} (\angle(wp_{i+1}, wp_{i+2})) - \text{HeadingInAir} (\angle(wp_i, wp_{i+1}))$ 
4   for  $\kappa \leftarrow [\kappa_{\min}, \kappa_{\max}]$  do
5      $\kappa_2(s) = \kappa$ 
6     for  $S_f^1 \leftarrow [S_f^{\min}, S_f^{\max}]$  do
7        $\kappa_1(s) \leftarrow \text{CurvPoly} \left( 0, \kappa, S_f^1, \kappa_{\max}, \kappa'_{\max}, \kappa''_{\max} \right)$ 
8       if  $\kappa_1(s) \in \emptyset$  then
9         break
10       $S_f^3 \leftarrow S_f^1$ 
11       $\kappa_3(s) \leftarrow \text{CurvPoly} \left( \kappa, 0, S_f^3, \kappa_{\max}, \kappa'_{\max}, \kappa''_{\max} \right)$ 
12       $S_f^2 = \frac{\Delta\psi - \int_0^{S_f^1} \kappa_1(s) - \int_0^{S_f^3} \kappa_3(s)}{\kappa}$ 
13      if  $S_f^2 \geq 0$  then
14        break
15       $\kappa(s) \leftarrow [\kappa_1(s) \ \kappa_2(s) \ \kappa_3(s)]$ 
16       $\xi_{\text{arc,gnd}}^i(s) \leftarrow \text{CurvPolyGnd} (\kappa(s))$ 
17      if  $\sum_{j=i}^{i+1} \mathbb{I}_j \left( \xi_{\text{arc,gnd}}^i(s) \right) > 0$  then
18        break
19 return  $v, \xi_{\text{arc}}^i(s), \xi_{\text{arc,gnd}}^i(s)$ 

```

straight segments ξ_{st}^i and turns ξ_{arc}^i . We obtain the segment velocity limits V_{lim} from Phase A, along with the airframe path lengths S_i for each segment. We further assume an acceleration $\hat{a}_{\max} = a_{\max} - \varepsilon_{\text{tol}}$ which is lower than the acceleration limit of the system. While the current phase ignores jerk, using a lower acceleration at this stage allows us to fit a jerk-limited velocity spline at a later stage. The optimization problem now is to determine the control-point velocities $\{\hat{V}_i\}$ which minimize time (where $\hat{V}_0 = V_{\text{start}}, \hat{V}_{N+1} = V_{\text{goal}}$):

$$\begin{aligned}
& \underset{\{\hat{V}_i\}}{\text{minimize}} && t_f \left(\{\hat{V}_i\}, \{S_i\}, \hat{a}_{\max} \right) \\
& \text{subject to} && \hat{V}_i \leq V_{\text{lim},i} \\
& && \frac{|\hat{V}_{i+1}^2 - \hat{V}_i^2|}{2\hat{a}_{\max}} \leq S_i
\end{aligned} \tag{10}$$

The total time t_f is defined as follows-

$$t_f = \sum_i t_i \tag{11}$$

Given a pair of consecutive point velocities \hat{V}_i, \hat{V}_{i+1} , t_i can be computed according to the following-

$$\begin{aligned}
V_{\text{mid}} &= \min \left(\sqrt{\frac{2\hat{a}_{\max}S_i + \hat{V}_i^2 + \hat{V}_{i+1}^2}{2}}, V_{\text{lim},i} \right) \\
t_i &= \frac{2V_{\text{mid}} - \hat{V}_{i+1} - \hat{V}_i}{\hat{a}_{\max}} + \left(S_i - \frac{2V_{\text{mid}}^2 - \hat{V}_{i+1}^2 - \hat{V}_i^2}{2\hat{a}_{\max}} \right) \frac{1}{V_{\text{mid}}}
\end{aligned} \tag{12}$$

This is solved online using *NLopt* (Johnson, 2014), an open-source nonlinear optimization library.

6.3.2 Algorithm for Initialization

Algorithm 2: VelOpt($V_{goal}, V_{start}, \{V_{lim,i}\}, \{S_i\}, a$)

```

1  $\{\hat{V}_i\} \leftarrow \{V_{start}, \{V_{pt,i}\}, V_{goal}\}; \text{Visited} \leftarrow \{0\}_{N+2};$ 
2  $\hat{V}_1 \leftarrow \text{MakeFeasible}(\hat{V}_0, \hat{V}_1, a, S_1); \hat{V}_N \leftarrow \text{MakeFeasible}(\hat{V}_{N+1}, \hat{V}_N, a, S_N);$ 
3  $\text{Visited}[0] \leftarrow 1; \text{Visited}[N+1] \leftarrow 1;$ 
4 repeat
5    $i \leftarrow \text{Minimum}(\{\hat{V}_i\}) \text{ s.t. } \text{Visited}[i] = 0;$ 
6   if  $\text{Visited}[i-1] = 0$  then
7      $\hat{V}_{i-1} \leftarrow \text{MakeFeasible}(\hat{V}_i, \hat{V}_{i-1}, a, S_i)$ 
8   if  $\text{Visited}[i+1] = 0$  then
9      $\hat{V}_{i+1} \leftarrow \text{MakeFeasible}(\hat{V}_i, \hat{V}_{i+1}, a, S_{i+1})$ 
10   $\text{Visited}[i] \leftarrow 1;$ 
11 until  $\text{Visited}[0..N+1] = 1;$ 

```

Algorithm 2 is used to feasibly initialize the nonlinear optimization problem above, which results in significant improvements in convergence rates. It uses the function $\text{MakeFeasible}(\hat{V}_1, \hat{V}_2, a, S)$, defined as:

$$\hat{V}_2 = \sqrt{\hat{V}_1^2 + 2aS} \quad (13)$$

6.4 Phase C

This stage operates on $\{\hat{V}_i\}$ and fits a smooth, jerk-limited spline $V_i(t)$ between each \hat{V}_i and \hat{V}_{i+1} . $V_i(t)$ is derived by integrating a C1 acceleration spline $a(t)$ comprising a degree-3 polynomial $a_1(t)$, a constant-acceleration section $a_2(t) = a_{trans}$ and another degree-3 polynomial $a_3(t)$. t_f^1, t_f^2, t_f^3 denote the time spanned by the three sections, and t_f denotes the total time of the spline. Each acceleration spline segment effects a velocity change from some \hat{V}_i to \hat{V}_{i+1} , and is exactly analogous in structure to the curvature spline described earlier. We omit the details of computing these splines, since they are the same as for the curvature splines.

Once all the spline segments have been computed, they are combined to yield the final airspeed spline $V(t)$, $t \in [0, t_f]$. It is important to note that time has been used here merely as a suitable parameter to compute these splines, and that it *does not* represent the actual time profile of the trajectory. $V(t)$ is thus converted to $V(\tau)$ by setting $\tau = \frac{t}{t_f}$, and consistency of τ with Phase A is maintained by construction. The next stage computes the final time-parameterized trajectory.

6.5 Phase D

At this stage, we have a ground-frame path $\xi_W(\tau)$ (Phase A), a ground-referenced heading profile $\psi_W(\tau)$ (Phase A) and an airspeed profile $V(\tau)$ (Phase C). We obtain a ground-referenced, time-parameterized trajectory $\sigma_W(t)$ according to the following-

1. Obtain a ground speed profile:

$$v_W(\tau) = \sqrt{V(\tau)^2 - v_w^2 \sin^2(\psi_W(\tau))} + v_w \cos(\psi_W(\tau)) \quad (14)$$

2. Obtain the *ground-frame* distance profile:

$$S_W(\tau) = S_W(x(\tau)) = \int_0^{x(\tau)} \sqrt{1 + y'(x)^2} dx, \quad y' = \frac{dy}{dx} \quad (15)$$

3. Compute the time profile:

$$t(\tau) = \int_0^{S_W(\tau)} \frac{dS_W}{v_W(\tau)} \quad (16)$$

At this point, it is trivial to replace τ with $t(\tau)$, and obtain a time-parameterization.

4. Compute the *air-referenced* yaw profile:

$$\psi(t) = \text{atan} \left(\frac{v_W(t) \sin(\psi_W(t))}{v_W(t) \cos(\psi_W(t)) - v_w} \right) \quad (17)$$

5. Compute the roll profile:

$$\phi_W(t) = \phi(t) = \text{atan} \left(\frac{V(t) \dot{\psi}(t)}{g} \right) \quad (18)$$

The final ground-frame trajectory $\sigma_W(t) = \{x_W(t), y_W(t), \psi_W(t), \phi_W(t)\}$ is now complete.

7 Trajectory Planning Algorithm

The trajectory planning algorithm module is tasked with producing feasible trajectories upto a finite horizon as shown in Fig. 9. It takes as input the original planning problem and produces a planning solution. Since the planning problem contains a global route guide trajectory, this module does not have to solve the full problem. If there are no obstacles or no-fly-zones, then this module simply feeds forward the route guide trajectory. If obstacles or no-fly-zones are present, the module computes a finite horizon trajectory that is valid, that connects to the route guide trajectory at the horizon and that copies over the velocity profile of the route guide trajectory.

7.1 Receding horizon problem generator

This sub-module takes the original planning problem Γ_{pp} as input (Fig. 9). It then decides a horizon t_{hor} to plan to and produces a corresponding receding horizon planning problem Γ_{rhpp} . This receding horizon problem is then passed along to the other submodules. The horizon restricts the scope of the planning so that the planning algorithms can produce solutions within the real-time constraints. This also allows the planning to scale to arbitrarily large distances.

Algorithm 3: CreateRecedingHorizonPlanningProblem(Γ_{pp}, t_{hor})

- 1 $(s, g, \mathcal{F}, \mathcal{H}, \Sigma_{valid}, \sigma_{rg}) \leftarrow \text{Unpack}(\Gamma_{pp});$
 - 2 $t_{proj} \leftarrow \text{Project}(s, \sigma_{rg});$
 - 3 $t_{hor}^+ \leftarrow \text{Propagate}(\sigma_{rg}, t_{proj}, \delta_{hor});$
 - 4 **if** $t_{hor}^+ \geq \min(t_{hor} + \varepsilon_{hor}, t_f(\sigma_{rg}))$ **then**
 - 5 $t_{hor} \leftarrow t_{hor}^+;$
 - 6 $\sigma_{rg}^+ \leftarrow \text{Chop}(\sigma_{rg}, t_{proj}, t_{hor});$
 - 7 $\Sigma_{valid}^+ \leftarrow \text{FilterRelevant}(\Sigma_{valid}, t_{hor});$
 - 8 $\Gamma_{rhpp} \leftarrow (s, g, \mathcal{F}, \mathcal{H}, \Sigma_{valid}^+, \sigma_{rg}^+);$
 - 9 **return** $(\Gamma_{rhpp}, t_{hor});$
-

Algorithm 3 describes this process for creating a receding horizon planning problem. Line 2 projects the start state s on the route guide trajectory σ_{rg} . This is done by iterating through states in σ_{rg} and returning the time index t_{proj} of the closest state according to the 3D Euclidean distance. Line 3 computes the time index of a potential new horizon t_{hor}^+ by propagating the state $\sigma_{rg}(t_{proj})$ forward by a distance of δ_{hor} along σ_{rg} . Line 4 checks whether the new horizon t_{hor}^+ is at-least ε_{hor} greater than the current horizon t_{hor} , and updates t_{hor} if that is the case. This condition is enforced to make sure the horizon does not move every time step invalidating old plans. Line 6 creates a new route guide σ_{rg}^+

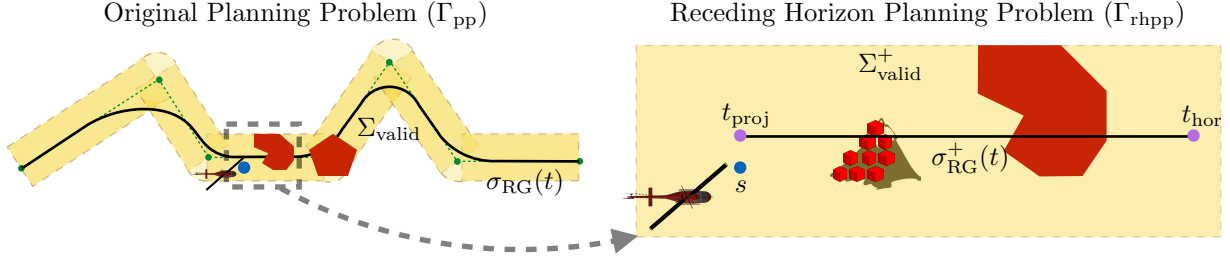


Figure 14: Generating a receding horizon problem. A time horizon t_{hor} is computed. The original route guide trajectory σ_{rg} is segmented to create σ_{rg}^+ . Only obstacles within the relevant region are considered to create Σ_{valid}^+ .

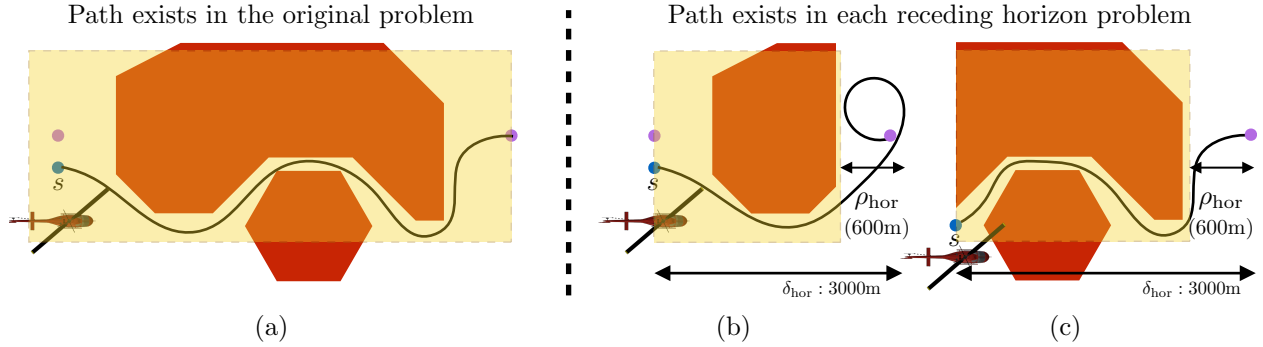


Figure 15: The receding horizon decomposition does not violate completeness. (a) The original problem where a path exists (b) The first receding horizon problem. Note that to ensure a path exists, filtering is done up to a distance ρ_{hor} from the horizon point. Since ρ_{hor} is more than the minimum turn radius, it is reachable. (c) As the robot moves, the horizon gets updated. Hence the robot does not end up entering the invalid area.

by copying σ_{rg} from start time t_{proj} to end time t_{hor} . Line 7 filters the obstacle and no-fly-zone constraints to remove obstacles that are not within the horizon t_{hor} . We will discuss in Section 7.1.2 how this filtering is performed to retain completeness. The updated components are packed to create the receding horizon planning problem Γ_{rhpp} . An illustration of this algorithm is shown in Fig. 14.

7.1.1 Dealing with multiple touchdown sites

The goal state g in Γ_{rhpp} represents a set of potential touchdown sites. This set is copied directly from the original planning problem Γ_{pp} . As long as the horizon is smaller than the terminal time, i.e. $t_{\text{hor}} < t_f(\sigma_{\text{rg}})$, the goal state g is considered inactive. In such situations, the goal of the trajectory planning module is to simply plan till $t_f(\sigma_{\text{rg}}^+)$.

Once $t_{\text{hor}} = t_f(\sigma_{\text{rg}})$, the planner is expected to plan to the goal state g . If the goal state has multiple touchdown sites (which are provided by the perception module and change with time), the planner has to produce a plan to each of these states. This can potentially increase the computational load on the planner. We describe in Section 7.6.1 how we deal with this issue.

7.1.2 Dealing with potential violation of completeness

A potential danger of receding horizon planning is that, although it helps in real-time performance by reducing the search volume, it can potentially violate completeness. Completeness in this context implies that given a solution exists for the original problem Γ_{pp} , will there be a situation where Γ_{rhpp} does not have a solution? A simple scenario where completeness is violated is if the horizon point $\sigma_{\text{rg}}^+(t_f)$ is invalid, i.e. $\sigma_{\text{rg}}^+(t_f) \notin \Sigma_{\text{valid}}^+$.

To fix this, we *filter* the constraint set Σ_{valid}^+ . This filtering not only ensures $\sigma_{\text{rg}}^+(t_f) \in \Sigma_{\text{valid}}^+$ is always true, but that $\sigma_{\text{rg}}^+(t_f)$ is always *reachable*. Hence, a section of the route guide of distance ρ_{hor} preceding $\sigma_{\text{rg}}^+(t_f)$ must be forced to

be valid as illustrated in Fig. 15. As the robot progresses, the horizon is updated, Σ_{valid}^+ is also updated and the robot never ends up actually violating Σ_{valid} . Fig. 15 illustrates this phenomenon.

We summarize how we select the parameters:

1. *The horizon distance* (δ_{hor}): We set this as large as possible such that planning within a real-time budget is possible given the computation constraints. Since we plan at 1 Hz, and we fly at speeds of up-to 50 m/s, we set this value to 3000 m.
2. *The scrolling threshold* (ε_{hor}): We set this to ensure old plans are valid for a few planning cycles before being invalidated by the horizon being updated. We set this to a value of 500 m.
3. *The filtering radius* (ρ_{hor}): We set this to be the minimum turning radius at the highest speed, i.e. 600 m.

7.2 Nominal planner

The role of the nominal planner submodule (Fig. 9) is to produce a planning solution in nominal situations. These are situations where it is not necessary to invoke the planning module - it suffices to re-use the route guide trajectory which was computed in absence of obstacles. The input to the module is the receding horizon planning problem $\Gamma_{\text{rhpp}} = (s, g, \mathcal{F}, \mathcal{H}, \Sigma_{\text{valid}}, \sigma_{\text{rg}})$. The output of the module is a nominal trajectory $\sigma_{\text{nom}}(t)$.

We need to define a subproblem for computing the nominal trajectory. At a given planning cycle, if the start state of the robot is on the route guide trajectory, then nominal solution is simply the route guide trajectory segment up to the horizon. However, if the start state is not on the route guide trajectory (e.g. because the system had to avoid an obstacle), this solution wont suffice. We assume that even in such situations the nominal planner should try to minimize deviations from the route guide trajectory. This is because the route guide trajectory was computed taking into account all global constraints - always trying to stay close to this trajectory results in predictable behavior. We also want the nominal planner to solve the a relaxed version of the original problem where we ignore the obstacle and route constraints. As we will see, this relaxation allows us to solve such problems efficiently.

We define the nominal planning problem as follows:

Problem 2 (Nominal planning problem). *The nominal planning problem is formally defined as solving for a nominal trajectory σ_{nom} that minimizes the projection error from the route guide trajectory while satisfying boundary and dynamics constraints.*

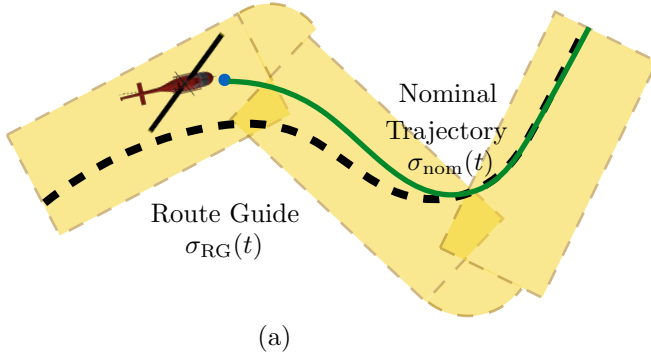
$$\begin{aligned}
 \min_{\sigma_{\text{nom}}, t_f} \quad & \int_0^{t_f} \inf_{\tau} \|\sigma_{\text{nom}}(t) - \sigma_{\text{rg}}(\tau)\| dt \\
 \text{s.t.} \quad & \sigma_{\text{nom}}(0) = s \\
 & \sigma_{\text{nom}}(t_f) = \sigma_{\text{rg}}(t_f(\sigma_{\text{rg}})) \\
 & \mathcal{F}_{\text{dyn}}(\sigma_{\text{nom}}) = 0 \\
 & \mathcal{H}_{\text{dyn}}(\sigma_{\text{nom}}) \leq 0
 \end{aligned} \tag{19}$$

The constraints are relaxed as the route inequality constraints and the safety constraints Σ_{valid} are removed. Because of the boundary and the dynamics constraints, this problem is still a non-linear optimization problem. However, we can approximately solve such problems using the Dynamics Projection Filter (DPF) (Choudhury and Scherer, 2015). This is a non-linear projection operator that can project an infeasible trajectory to the feasible set using a feedback controller. If we invoke such a filter with s as the initial value and σ_{rg} as the trajectory to follow, we get a solution σ_{nom} that not only satisfies the dynamics constraints by construction but also reduces the projection error¹ Details about DPF is described in Appendix C.

The employment of DPF for computing nominal solutions results in very small computation times - we can easily solve for dynamically feasible trajectories that are 3000 m long within the 100ms time budget that we allocate. The success of the method can be attributed to the heavy lifting of the route guide trajectory, the fact that the s does not stray significantly from the route guide. As illustrated in Fig. 16(a), the nominal solution closely approximates the

¹However, it is not directly optimizing the stated objective. Rather, the feedback controller inside the filter is defining a tracking error which leads to a reduction of the projection error.

Output of DPF is feasible as it satisfies all constraints, hence this is returned as the planning solution



DPF violates route constraints - results in avoidance problem that requires planning

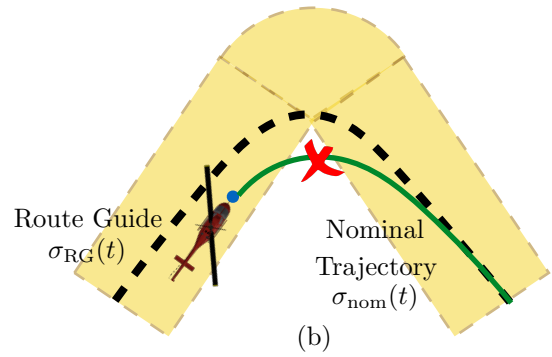
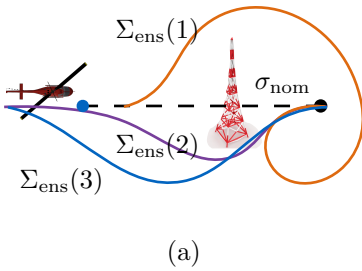
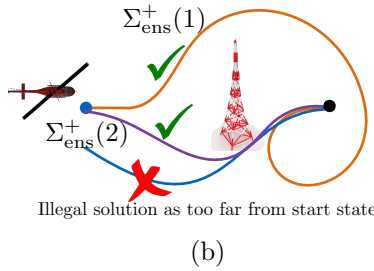


Figure 16: Success and failure scenarios for the nominal planner which solves a relaxed version of the original problem. (a) The DPF produces a solution that closely approximates the route guide trajectory and happens to satisfy the route constraints. In such situations, this is a desirable solution to follow. (b) The DPF produces a solution that violates route constraints. In such situations, the nominal planner does not suffice and a planning algorithm needs to be invoked to solve it.

Nominal solution infeasible. Ensemble of planners invoked.



Trajectory selector filters illegal solutions to get a valid set



Best solution selected and returned as planning solution

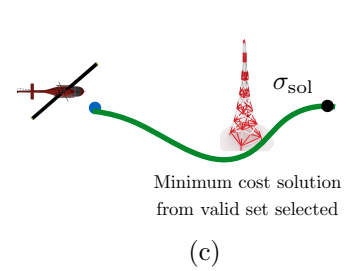


Figure 17: Illustration of the trajectory selector (a) Since the nominal solution is infeasible, an ensemble of planners is invoked. These planners solve for receding horizon trajectories that are feasible. Solutions from previous time steps are also retained in a buffer. Here Σ_{ens} has 3 trajectories: $\Sigma_{ens}(1)$ is a newly planned trajectory, $\Sigma_{ens}(2)$ and $\Sigma_{ens}(3)$ are old trajectories from a previous time step (b) The selector filters illegal solutions (e.g. solutions that originated from before the start state and cannot be repaired) to get a valid set Σ_{ens}^+ . Here $\Sigma_{ens}(3)$ is filtered out but $\Sigma_{ens}(2)$ is repaired. (c) The best solution is selected to be $\sigma_{sol} = \Sigma_{ens}(2)$.

desired solution that we would have obtained had we invoked the route guide to solve the entire problem at every time step (the route guide takes several seconds to compute a solution).

However, the nominal solution does fail when there are obstacles or no-fly-zones in the environment. For such situations, a planning algorithm needs to be invoked. There are other situations as well where even in the absence of any obstacles, the nominal solution fails as shown in Fig. 16(b). These are situations where the route constraints are violated, thus requiring a planner to compute a solution.

7.3 Trajectory selector

The role of this submodule (Fig. 9) is to select the best solution to send as output back to the planning state machine module. It takes as input the nominal solution σ_{nom} . It also takes as input solutions from an ensemble of planners in the event the nominal solution is infeasible. The output is the selected solution σ_{sol} .

Algorithm 4: TrajectorySelector($\Gamma_{\text{rhpp}}, \sigma_{\text{nom}}, \Sigma_{\text{ens}}, \sigma_{\text{sol}}$)

```
1 ( $s, g, \mathcal{F}, \mathcal{H}, \Sigma_{\text{valid}}, \sigma_{\text{rg}}$ )  $\leftarrow$  Unpack( $\Gamma_{\text{rhpp}}$ );
2 if  $\mathcal{F}(\sigma_{\text{nom}}) = 0$  and  $\mathcal{H}(\sigma_{\text{nom}}) \leq 0$  and  $\sigma_{\text{nom}} \in \Sigma_{\text{valid}}$  then
3    $\sigma_{\text{sol}} \leftarrow \sigma_{\text{nom}}$ ;
4 else
5    $\Sigma_{\text{ens}} \leftarrow \Sigma_{\text{ens}} \cup \sigma_{\text{sol}}$ ;
6    $\Sigma_{\text{ens}}^+ \leftarrow \text{FilterIllegal}(\Sigma_{\text{ens}}, \Gamma_{\text{rhpp}})$ ;
7    $\sigma_{\text{sol}} \leftarrow \arg \min_{\sigma \in \Sigma_{\text{ens}}^+} J(\sigma)$  s.t.  $\mathcal{F}(\sigma) = 0, \mathcal{H}(\sigma) \leq 0$ ;
8 return  $\sigma_{\text{sol}}$ ;
```

Algorithm 4 describes the selection algorithm. Line 2 checks if the nominal solution σ_{nom} is valid. If so, the solution is selected and returned. If σ_{nom} is infeasible, an ensemble of planners is invoked. The trajectory selector collects the set of solutions returned by the ensemble. It retains a buffer of such solutions, including previously selected solutions, as described in Line 5. It then filters out illegal solutions as described in Line 6. Illegal solutions can belong to a couple categories:

1. A solution that cannot be repaired to start from the current start state.
2. A solution that plans to an older horizon.

From the filtered valid set, the solution with the minimum cost is selected and returned as shown in Line 7. An illustration of this selection is also shown in Fig. 17.

7.4 Solving avoidance problem with an expert planner

7.4.1 Avoidance planning problem

We describe the avoidance planning problem as follows:

Problem 3 (Avoidance planning problem). *The avoidance planning problem is formally defined as solving for a trajectory σ up-to the receding horizon that minimizes the original cost function while satisfying original dynamics and safety constraints.*

$$\begin{aligned} & \min_{\sigma, t_f} && J(\sigma) \\ \text{s.t.} &&& \sigma(0) = s \\ &&& \sigma(t_f) = \sigma_{\text{rg}}(t_f(\sigma_{\text{rg}})) \\ &&& \mathcal{F}(\sigma) = 0 \\ &&& \mathcal{H}(\sigma) \leq 0 \\ &&& \sigma \in \Sigma_{\text{valid}} \end{aligned} \tag{20}$$

This is essentially the original planning problem with the one modification - the goal point is now the end of the route guide segmented at the horizon. Hence this problem is computationally simpler than solving the original problem. Also note that the route guide itself is a solution to the global problem in absent of obstacles - hence a planning algorithm can use the spatial and velocity profiles in the route guide to aid in solving this problem. However, unlike the nominal problem in Problem 2, we do not penalize deviation from the route guide as it may lead to undesirable behavior.

7.4.2 Expert planner

The planning problem in Problem 3 is a high dimensional kinodynamic planning problem which is required to be solved in real-time (≈ 1 Hz). Although this is an active area of research with methods such as Kinodynamic RRT* (Webb and van den Berg, 2013) and SST (Li et al., 2015) pushing the state of the art, it is impractical to apply such methods to the full problem.

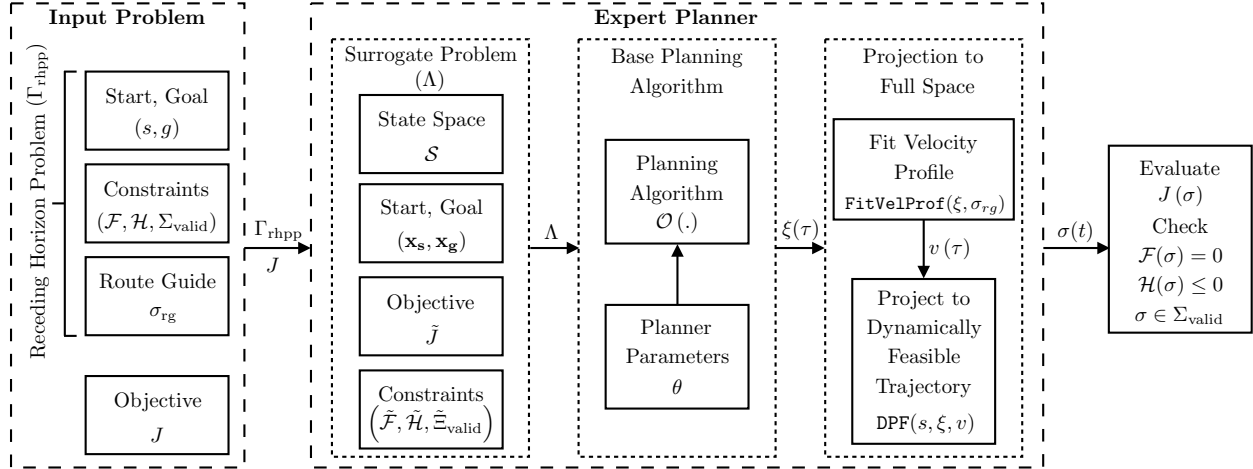


Figure 18: Overview of an expert planner.

We follow guidelines presented by (Laumond et al., 1998) and (LaValle, 2006) and decouple the approach into path planning and trajectory planning. However, we observe that there are *multiple* ways to achieve this decoupling - the success and failures of which depends on not just the dynamics of the system, but *varies* with planning problems. For example, (Laumond et al., 1998) describe a curvature rate constrained system being treated as a Dubin’s car (Dubins, 1957) to plan a path, following by a “smoothing” method such that a feasible time indexing can be applied. However, planning with the Dubin’s car model might not always be a suitable relaxation. In some cases, planning with a smoothness constraint, or even unconstrained planning - followed by the “smoothing” method and time profile assignment can provide acceptable performance. We first present a framework to allow this flexibility and give examples of different relaxations being suitable in different context.

In this paper, we propose the framework of an *expert planner* to solve the planning problem in Problem 3. An expert planner defines a mapping from the original problem to a surrogate low dimension path planning problem, solves this efficiently using a base planning algorithm, and projects the solution back into the full space to form a trajectory. The term “expert” is borrowed from (Blum, 1998) to denote that the different modules in the expert planner framework are not independently chosen at random but instead carefully assembled by an expert.

Fig. 18 shows an overview of an expert planner. It consists of 3 main modules.

1. *Surrogate Problem* Λ : The expert planner first maps the high dimensional problem Γ_{rhpp} to a low dimensional surrogate path planning problem Λ with the expectation that a solution to the surrogate problem can be processed to produce a candidate solution to the original problem. The surrogate problem is a tuple $\Lambda = (\mathcal{S}, \mathbf{x}_s, \mathbf{x}_g, \tilde{J}, \tilde{\mathcal{F}}, \tilde{\mathcal{H}}, \tilde{\Xi}_{\text{valid}})$ where \mathcal{S} is the search space, $(\mathbf{x}_s, \mathbf{x}_g)$ is the pair of start and goal states, \tilde{J} is the objective and $(\tilde{\mathcal{F}}, \tilde{\mathcal{H}}, \tilde{\Xi}_{\text{valid}})$ are the constraints.
2. *Base Planning Algorithm* $\mathcal{O}(\cdot)$: The expert planner invokes a base planning algorithm to solve the surrogate problem. This base planner is an operator $\mathcal{O}(\Lambda, \theta)$ that maps the surrogate problem Λ and base planner parameters θ to a path $\xi(\tau)$. A path is defined as a functional mapping $\xi : [0, 1] \rightarrow \mathcal{S}$.
3. *Projection to Original Space*: The expert planner finally projects the path $\xi(\tau)$ to a trajectory $\sigma(t)$ in the original space. This is done in a two step process. Firstly, a velocity profile is assigned to the path from the route guide σ_{rg} . The operator $\text{FitVelProf}(\xi, \sigma_{\text{rg}})$ maps the path $\xi(\tau)$ and the route guide σ_{rg} to a velocity profile $v(\tau)$. We do this by solving for a linear mapping between τ and t . Since the length of the solution ξ is more than σ_{rg} , this mapping does not result in a violation of velocity (and derivative) limits. Secondly, a dynamics projection filter operator $\text{DPF}(s, \xi, v)$ is used to map an infeasible path to a dynamically feasible trajectory. Description of the filter is provided in Appendix C.

Fig. 19 illustrates the expert planner procedure. An expert planner satisfies the following contract - it takes as input the

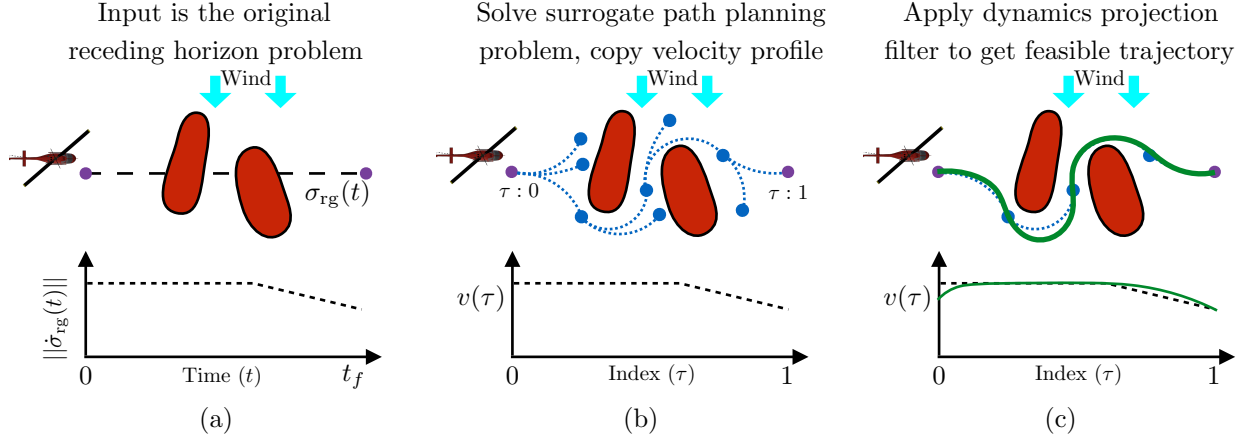


Figure 19: Mapping the original problem to a surrogate path planning problem. (a) The input problem where the route guide is infeasible. It is still used for its velocity profile (b) A surrogate path planning problem is created and solved. The velocity profile from the route guide is carried over. (c) The dynamics projection filter is applied to obtain a feasible trajectory. Because the surrogate path constraints were chosen appropriately (also taking wind into account), the filtered trajectory does not deviate significantly.

planning problem Γ_{rhpp} , and produces as output either a feasible trajectory σ (that satisfies $\mathcal{F}(\sigma) = 0, \mathcal{H}(\sigma) \leq 0, \sigma \in \Sigma_{valid}$) or an empty solution \emptyset . Hence the output can directly be used without the need for any kind of post-processing.

Example 1 (*Mapping high speed flight to surrogate problem on curvature+glide-slope constrained state space*). We now discuss an example class of expert planners that are applicable when the helicopter is moving at high speeds, i.e. $\|\dot{x}(t) \quad \dot{y}(t) \quad \dot{z}(t)\|_2 \geq v_{crit}$ where high speed dynamics constraints are active.

Table 1: Surrogate problem on curvature+glide slope constrained state space

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathbb{R}^3 \circ SO(2)$ $\xi[0, 1] \mapsto (x, y, z, \psi)$	$\mathbf{x}_s = (s_x, s_y, s_z, s_\psi)$ $q = \sigma_{rg}(t_f)$ $\mathbf{x}_g = (q_x, q_y, q_z, q_\psi)$	$\tilde{J}(\xi) = \int_0^1 \ \dot{\xi}(\tau)\ d\tau$	$\tilde{\mathcal{F}}: \psi = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$ $\tilde{\mathcal{H}}: \frac{\dot{x}\dot{y} - \dot{y}\dot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \leq \kappa_{max}$ $\frac{\dot{z}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \leq \gamma_{max}$ $\tilde{\Sigma}_{valid}: \mathbf{corr}(x, y, z) \leq 0$ $\mathbf{nfz}(x, y, z) = 0$ $\mathbf{ttc}(x, y, z) \geq t_{coll, min}$

Table 1 defines the mapping of the original problem Γ_{rhpp} to the surrogate path planning problem Λ . The search space $\mathcal{S} = \mathbb{R}^3 \circ SO(2)$ is a composition of the \mathbb{R}^3 position space and $\circ SO(2)$ heading space. The start and goal states are obtained by copying the (x, y, z, ψ) values from the original start state s and the route guide terminal point $\sigma_{rg}(t_f)$ respectively. The objective is the arc length of the path (only position space). This is chosen as the original objective $J(\sigma)$ minimizes traversal time.

We now have to constrain the path such that we are able to project it to a feasible trajectory. Based on the high speed dynamic constraints specified in Section 3.3, a useful approximation is to use the Dubins' car model (Dubins, 1957) for x, y, ψ . In this model, the heading is chosen to lie along the arc tangent, i.e. $\psi = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$. The curvature of the path is also bounded. We choose a curvature bound such that for the maximum speed required, the path will result in a feasible trajectory, i.e. the DPF(.) will not diverge.

We set the curvature limit $\kappa_{max} = \frac{1}{R_{min}}$, where R_{min} is a conservative minimum radius given the maximum airspeed v_a in the route guide, the wind speed v_w , the maximum roll angle ϕ_{max} and the maximum roll rate $\dot{\phi}_{max}$ defined as

follows:

$$R_{\min} = \frac{v_a^2}{g \tan \phi_{\max}} \max_{\theta \in [0, 2\pi]} \frac{\left(1 + \left(\frac{v_w}{v_a}\right)^2 + 2 \left(\frac{v_w}{v_a}\right) \cos \theta\right)^{\frac{3}{2}}}{1 + \left(\frac{v_w}{v_a}\right) \cos \theta} + \frac{\pi}{v_a} \frac{\dot{\phi}_{\max}}{\phi_{\max}} \quad (21)$$

The first term in the expression is the minimum radius in air frame multiplied by the maximum inflation due to the presence of wind. While this is not analytic, a lookup table of this factor can be computed and referenced. The second term is to account for the roll rate constraints and the lag induced.

We also set a limit on the glide-slope, i.e. $\frac{\dot{z}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \leq \gamma_{\max}$. This can be bounded as follows:

$$\gamma_{\max} = \frac{v_{z, \max}}{v_a} \quad (22)$$

The helicopter performance constraints can further constrain the glide-slope as described in Section 7.6.2.

The remaining constraints ensure that the path is inside the route (i.e. $\text{corr}(x, y, z) \leq 0$) and the path does not intersect with any no-fly-zones (i.e. $\text{nfz}(x, y, z) = 0$). By assuming a constant speed v_a , we can also transfer the time to collision constraints $\text{ttc}(x, y, z) \geq t_{\text{coll}, \min}$. For efficiency purposes, when computing $\text{ttc}(x, y, z)$, we use only about the nearest obstacle as the relevant set $\mathcal{P}_{\text{rel}} = p_{\text{obs}}([x \ y \ z])$.

A scenario that may occur when applying $\text{DPF}(\cdot)$ is that the output trajectory will deviate from the planned path. Hence even if the planned path avoids invalid regions, the output trajectory might violate Σ_{valid} . To overcome this, we do two things:

1. We increase the value of $t_{\text{coll}, \min}$ by ϵ_1 for the surrogate problem.
2. We expand each of the no-fly-zone polygons by ϵ_2 for the surrogate problem.

Values of ϵ_1 and ϵ_2 must be chosen based on maximum deviation expected from $\text{DPF}(\cdot)$.

We now have a fully defined surrogate path planning problem Λ . We can choose from a wide range of base planning algorithms $\mathcal{O}()$ to solve the problem. One such class of algorithms is geometric sampling based planning such as RRT* (Karaman and Frazzoli, 2011), Informed RRT* (Gammell et al., 2014), BIT* (Gammell et al., 2015), FMT* (Janson et al., 2015), etc. Another such class is discrete search on a lattice of motion primitives (Pivtoraiko et al., 2009). Appendix B describes the list of planners that we have experimented with.

We now describe one such planning algorithm RRT*Tunnel. This algorithm forms the base of expert planners RRT*Tunnel¹, RRT*Tunnel² that is commonly referred to in this paper. This algorithm modifies a sampling based planner RRT* to sample in the *workspace* - specifically in a tunnel around a nominal path. While workspace sampling does in fact violate the completeness and asymptotic optimality of the algorithm, for the range of problems we were solving we found it to have good real time performance. We describe the algorithm in Algorithm 5. For details on the parameters used in the planner, we refer the reader to Appendix B.

Example 2 (*Mapping both high and low speed flight to surrogate smooth optimization*). This example class of expert planners is applicable at both high and low speed since the surrogate problem simply enforces smoothness. Table 2 defines the mapping of the original problem Γ_{rhpp} to the surrogate path planning problem Λ . Here, the search space is $\mathcal{S} = \mathbb{R}^3$. The start and goal states are obtained by copying the (x, y, z) values from the original start state s and the route guide terminal point $\sigma_{\text{rg}}(t_f)$ respectively. The objective is the arc length integral of smoothness and a time to collision objective defined on a path. The constraint is the nonlinear glide slope constraint.

The class of expert planners we used to solve such problems is covariant gradient descent (CHOMP) as described in (Ratliff et al., 2009). Note that the vanilla version of the CHOMP algorithm is unconstrained. However, we can extend it to handle linear inequality constraints while still retaining its efficiency.

We also have the velocity profile $v(\tau)$ for the path. This allows us to transfer the time to collision function $\text{ttc}(\mathbf{p}, \mathbf{v})$ to a function $\text{ttc}(x, y, z)$. For efficiency purposes, when computing $\text{ttc}(x, y, z)$, we use only about the nearest obstacle as the relevant set $\mathcal{P}_{\text{rel}} = p_{\text{obs}}([x \ y \ z])$.

Algorithm 5: RRT* Tunnel

```
1  $V \leftarrow \mathbf{x}_s$ ;  $E \leftarrow \emptyset$ ;
2 while planning time budget left do
3    $\mathbf{p}_{\text{rand}} \leftarrow \text{Tunnel}(\delta_{\text{tunn}})$ ;  $\triangleright$  Sample a  $\mathbb{R}^3$  position from a tunnel  $\delta_{\text{tunn}}$  around the
   nominal path
4    $\mathbf{x}_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), \mathbf{p}_{\text{rand}})$ ;  $\triangleright$  Find nearest vertex using distance in  $\mathbb{R}^3$ 
5    $(\mathbf{x}_{\text{new}}, \mathbf{p}_{\text{new}}) \leftarrow \text{Steer}(\mathbf{x}_{\text{nearest}}, \mathbf{p}_{\text{rand}})$ ;  $\triangleright$  Steer vertex towards position to get new
   state and position
6   if  $\text{Valid}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$  then
7      $X_{\text{near}} = \text{Near}(G, \mathbf{p}_{\text{new}})$ ;  $\triangleright$  Find set of near vertices in  $G$  to  $\mathbf{p}_{\text{new}}$  in  $\mathbb{R}^3$ 
8      $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{nearest}}$ ;  $c_{\text{min}} \leftarrow \text{Cost}(\mathbf{x}_{\text{nearest}}) + c(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{new}})$ ;  $\triangleright$  Initialize best parent  $\mathbf{x}_{\text{min}}$ 
     and cost  $c_{\text{min}}$ 
9     foreach  $\mathbf{x}_{\text{near}} \in X_{\text{near}}$  do
10       $\mathbf{x}_{\text{end}} \leftarrow \text{Steer}(\mathbf{x}_{\text{near}}, \mathbf{p}_{\text{new}})$ ;  $\triangleright$  Steer from candidate parent towards position to
      get state
11      if  $\text{Valid}(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{end}})$  and  $\text{Cost}(\mathbf{x}_{\text{near}}) + c(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{end}}) < c_{\text{min}}$  then
12         $\mathbf{x}_{\text{new}} \leftarrow \mathbf{x}_{\text{end}}$ ;  $\mathbf{x}_{\text{min}} \leftarrow \mathbf{x}_{\text{near}}$ ;  $c_{\text{min}} \leftarrow \text{Cost}(\mathbf{x}_{\text{near}}) + c(\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}})$ ;
13       $V \leftarrow V \cup \{\mathbf{x}_{\text{new}}\}$ ;  $\triangleright$  New state added to set of vertices
14       $E \leftarrow E \cup \{(\mathbf{x}_{\text{min}}, \mathbf{x}_{\text{new}})\}$ ;  $\triangleright$  New edge added to set of edges
15       $G \leftarrow \text{Rewire}(G, \mathbf{x}_{\text{new}})$ ;  $\triangleright$  Rewire graph as in original RRT*
16 return  $\xi \leftarrow \text{GetPath}(G, \mathbf{x}_g)$ ;
```

Table 2: Surrogate problem on space of smooth trajectories

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathcal{P}$ $\xi[0, 1] \mapsto (x, y, z)$	$\mathbf{x}_s = (s_x, s_y, s_z)$ $\mathbf{x}_g = (g_x, g_y, g_z)$	$\tilde{J}(\xi) = \int_0^1 \left(\lambda \left\ \dot{\xi}(\tau) \right\ _2^2 + \frac{1}{2} (\text{ttc}_{\text{max}} - \text{ttc}(\xi(\tau)))^2 \left\ \dot{\xi}(\tau) \right\ \right) d\tau$	$\tilde{\mathcal{H}} : \frac{\dot{z}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \leq \gamma_{\text{max}}$ $\tilde{\Xi}_{\text{valid}} : \text{corr}(x, y, z) \leq 0$ $\text{nfz}(x, y, z) = 0$

We now describe the ConstrainedCHOMP algorithm. This algorithm forms the basis of commonly used expert planners CHOMP¹, CHOMP². Algorithm 6 states the algorithm. The key trick is to linearize the glideslope inequality constraints and no-fly-zone / corridor constraints. The steepest descent direction can then take into account these constraints while updating the trajectory. The gradient of the cost function can be computed in a similar fashion as the original CHOMP algorithm (Ratliff et al., 2009).

Algorithm 6: ConstrainedCHOMP

```
1  $\xi_1 \leftarrow \text{GetInterpolatedPath}(\mathbf{x}_s, \mathbf{x}_g, \delta)$ ;  $\triangleright$  Create path by interpolating at  $\delta$  distance
2  $i \leftarrow 1$ ;
3 while planning time budget left do
4    $G \leftarrow \nabla \tilde{J}(\xi_i)$ ;  $\triangleright$  Evaluate gradient
5   Convert constraint  $\tilde{\mathcal{H}}(\xi_i) \leq \gamma_{\text{max}}$  to linear inequality  $H\xi \leq I$ ;
6   Convert constraint  $\tilde{\Xi}_{\text{valid}}(\xi_i) \leq 0$  to linear inequality  $K\xi \leq I$ ;
7   Solve  $\xi_{i+1} \leftarrow \arg \min_{\xi} (\xi - \xi_i)^T G + \frac{\eta}{2} \|\xi - \xi_i\|_A$  s.t.  $H\xi \leq I, K\xi \leq I$ ;  $\triangleright$  Solve QP
8 return  $\xi \leftarrow \min_{\xi_i} \tilde{J}(\xi_i)$  s.t.  $\tilde{\mathcal{H}}(\xi_i) \leq \gamma_{\text{max}}, \tilde{\Xi}_{\text{valid}}(\xi_i) \leq 0$ ;
```

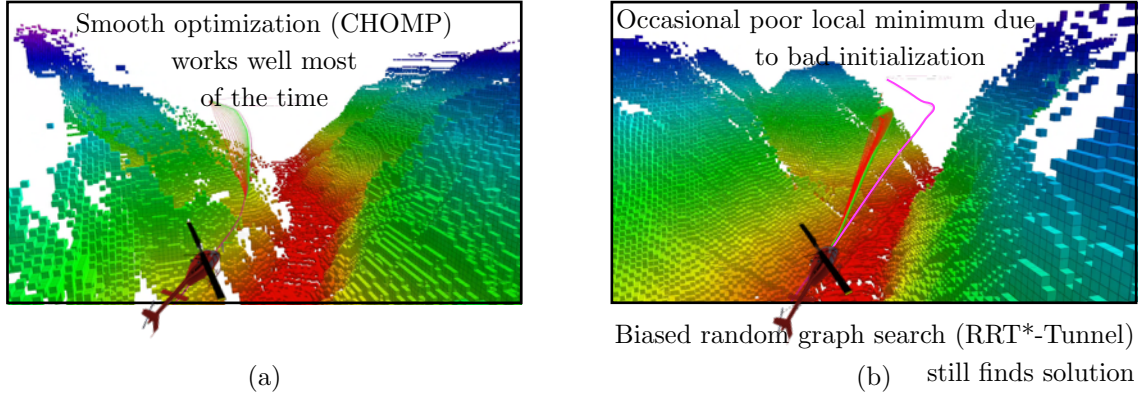


Figure 20: Merits and pitfalls of precision planners. (a) A smooth optimization approach about an initial guess, CHOMP¹ (Appendix B) acts as a precision planner by dedicating its planning effort to search around the initial guess. This works remarkably well in these cases and outperforms other general purpose planners. (b) However, there are several cases when this planner fails catastrophically. In such a scenario, a relatively more general purpose planner, RRT*¹ Tunnel¹ (Appendix B), is still able to find an acceptable solution.

7.4.3 Distribution of planning problems

Our objective is to design an expert planner that maximizes real-time performance on the problems the robot encounters. An expert planner \mathcal{P} as an operator that takes a planning problem Γ as input and produces a trajectory as output, i.e., $\sigma = \mathcal{P}(\Gamma)$. A real-time expert planner is a module that is allocated a fixed time budget. If the planner is unable to find a feasible, collision free trajectory within this time budget, it returns an empty trajectory $\sigma = \emptyset$. Hence the cost of the output of the planner is always defined, i.e. $J(\mathcal{P}(\Gamma)) \in [0, J_{\max}]$ for all planning problems Γ .

We also define the *distribution over planning problems* the robot encounters as $P(\Gamma)$. This distribution can be derived from a mission flown by the helicopter. A typical mission is a series of flights that the helicopter has to do in a particular geographical terrain. The planning problem encountered by the robot during such a mission are said to be sampled from this distribution.

We have a database of terrain maps corresponding to different geographical regions. In simulation, we can load these maps and sample different start and goal locations to create a database of problems $\{\Gamma\}_{i=1}^N$. Whenever the system has to fly a real mission, we can retrieve the corresponding database. If the system is operating across multiple terrains, databases can be merged to create a mixture distribution.

We define *performance* of an expert planner on a distribution of planning problems as the expected cost of trajectories returned by the planner. Given a planner \mathcal{P} and a planning problem distribution $P(\Gamma)$, the performance of the planner is $\mathbb{E}_{\Gamma \sim P(\Gamma)} [J(\mathcal{P}(\Gamma))] \approx \sum_{i=1}^N J(\mathcal{P}(\Gamma_i))$. We wish to design an expert planner \mathcal{P} that has high performance, i.e., that sufficiently minimizes expected cost.

We can quite easily create a library of expert planners for general UAVs as shown in Appendix B. Note that there is no universally effective expert planner in the library. In-fact, based on the profile of their performance, we define two categories of expert planners: *general purpose planners* and *precision planners*. General purpose planners are those that work reasonably well on a large number of problems, but are generally unable to find the optimal path on any problem (within the prescribed time budget). Examples are algorithms such as RRT* (Karaman and Frazzoli, 2011), BIT* (Gammell et al., 2015), RRT-Connect (Kuffner and LaValle, 2000), A* on state lattice (Likhachev and Ferguson, 2009), etc. Precision planners are those that finds near-optimal solutions on a small number of problems but fail to produce any solution on most problems. Examples are approaches using local trajectory optimization (Ratliff et al., 2009; Schulman et al., 2013), shooting methods (Bryson and Ho, 1975) or custom samplers as illustrated just now. Fig. 20 shows a real world example of the trade-off between these two classes.

7.5 Training a meta-planner to select an ensemble of planners

We advocate the use of an *ensemble of expert planners* as illustrated in Fig. 9. At a given decision step, the planning problem is provided to a meta-planner which selects an ensemble of planners to execute in parallel. Each planner is *independent* of another and produces its own plan to solve the problem. The plans are provided to a trajectory selector that takes the best of the solutions and sends it out.

7.5.1 Predicting a static ensemble

A static ensemble meta-planner takes as input a planning problem distribution $P(\Gamma)$ and outputs a fixed ensemble \mathcal{E} at every decision step. The motivation for a static ensemble is design simplicity and easier verification.

Algorithm 7: Greedy Selection of a Static Ensemble ($P(\Gamma), \mathcal{L}$)

- 1 Sample planning problem dataset $\{\Gamma_i\}_{i=1}^N$ from $P(\Gamma)$;
 - 2 Evaluate all planners $\mathcal{P} \in \mathcal{L}$ on $\{\Gamma_i\}_{i=1}^N$ to compute a cost matrix $\bar{\mathbf{J}} \in \mathbb{R}^{N \times |\mathcal{L}|}$;
 - 3 $\mathcal{E} \leftarrow \emptyset$;
 - 4 **while** $|\mathcal{E}| < B$ **do**
 - 5 Using $\bar{\mathbf{J}}$, select $\mathcal{P}^* \leftarrow \arg \min_{\mathcal{P} \in \mathcal{L} \setminus \mathcal{E}} \hat{R}(\mathcal{E} \cup \mathcal{P})$;
 - 6 $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{P}^*$
 - 7 **return** \mathcal{E} ;
-

Algorithm 7 describes the greedy ensemble selection which iteratively picks the planner that minimizes the combined risk the most. Line 1 samples a dataset of planning problems $\{\Gamma_i\}_{i=1}^N$ from the distribution $P(\Gamma)$. Line 2 applies each planner $\mathcal{P} \in \mathcal{L}$ on the dataset $\{\Gamma_i\}_{i=1}^N$ to create a cost matrix $\bar{\mathbf{J}} \in \mathbb{R}^{N \times |\mathcal{L}|}$. Then line 5 is iteratively repeated to compute the planner that minimizes the cumulative risk computed from $\bar{\mathbf{J}}$ and is appended to the selected ensemble \mathcal{E} . We can see that this greedy selection algorithm naturally results in a diverse set of planners.

7.5.2 Predicting a dynamic ensemble

There are certain situations where a static ensemble does not meet a desired performance criteria. In such situations, the ensemble of planners being used has to be dynamic - change with every decision step. This requires extracting *context or features* from the planning problem. These features are then used by a classifier to predict which ensemble to use. For our problems, we create a feature vector by sampling the workspace at a fixed resolution, checking if a point is valid or not to create a 3D bitmap. This bitmap is flattened to create the feature vector. We then train an ensemble using the procedure prescribed in (Tallavajhula et al., 2016).

Algorithm 8: List Prediction for Dynamic Ensemble

- 1 Sample planning problem dataset $\{\Gamma_i\}_{i=1}^N$ from $P(\Gamma)$;
 - 2 Compute feature projections $\{f_i\}_{i=1}^N$ for each problem;
 - 3 Evaluate all planners $\mathcal{P} \in \mathcal{L}$ on $\{\Gamma_i\}_{i=1}^N$ to compute a cost matrix $\bar{\mathbf{J}} \in \mathbb{R}^{N \times |\mathcal{L}|}$;
 - 4 Create loss matrix $\bar{\mathbf{L}}$ from $\bar{\mathbf{J}}$;
 - 5 **for** $k \in \{1, \dots, B\}$ **do**
 - 6 Train a predictor π^k to do loss sensitive classification using features $\{f_i\}_{i=1}^N$ and loss matrix $\bar{\mathbf{L}}$;
 - 7 Use the trained predictor π^k to update loss matrix $\bar{\mathbf{L}}$;
 - 8 **return** $\langle \pi^1 : \pi^B \rangle$;
-

Algorithm 8 describes the algorithm. There is an interesting interpretation of the training rule. As the predictors are trained sequentially, they focus on different planning problems. The second predictor will focus on the planning problems where the first predictor had low performance. The third will focus on planning problems where the first two had low performance, and so on.

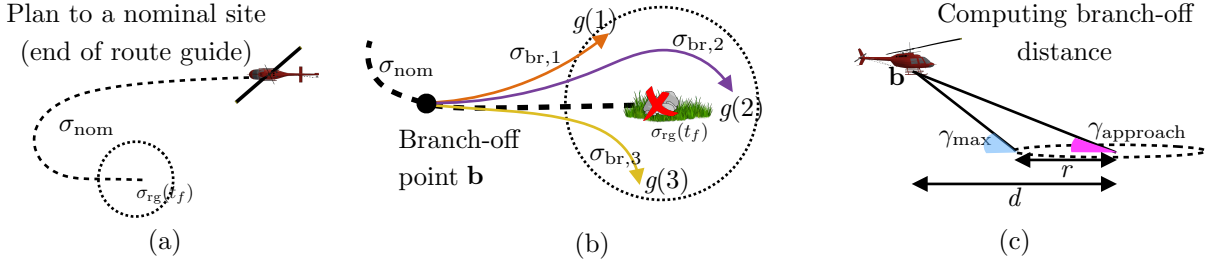


Figure 21: Planning from a branch-off point to plan to multiple touchdown sites (a) We first plan a nominal trajectory σ_{nom} to the nominal touchdown site which is the terminal point of the route guide $\sigma_{\text{rg}}(t_f)$. (b) We compute a branch-off point \mathbf{b} on σ_{nom} . From \mathbf{b} , we compute trajectories $\sigma_{\text{br},1}, \sigma_{\text{br},2}, \sigma_{\text{br},3}$ to each touchdown site $g(1), g(2), g(3)$. (c) The branch-off point is calculated at a distance d from the terminal point based on a landing zone radius of r , an approach glide slope of γ_{approach} and a maximum glide slope of γ_{max} . This is to ensure that all points in the landing zone are reachable from the branch-off in absence of obstacles.

7.6 Design decisions in planner implementation

We now discuss a handful of design decisions we had to employ to deal with various mission requirements and flight performance charts. While these discussions are relatively less general, they do highlight some of the techniques we use to make real-time planning tractable.

7.6.1 Computing a branch-off point to plan to multiple touchdown sites

As we mentioned before, the planning problem Γ_{pp} has a goal state that may correspond to one or more touchdown sites. We begin with only one nominal touchdown site as specified by the user. As the helicopter approaches the nominal touchdown site, the perception system is able to scan a landing zone around the nominal site. It then dynamically populates the goal state with these sites. The planner is expected to plan to all these sites and return a set of solutions. These solutions are then either processed by a higher level mission executive or examined by a human operator near the ground to make a selection.

Since we do not have the computational budget to plan individually to each of the touchdown sites, we make a simplifying assumption. As all the touchdown sites are spatially close (within a circle of 100 m diameter), we assume there exists a *branch-off point* from which all these sites are reachable (in absence of obstacles). Hence, our strategy is to plan a *single trajectory* to the nominal site first. We can then plan from the branch off-point to each of these touchdown sites which is computationally simpler than planning to each of the sites.

Fig. 21(a)(b) illustrates this approach. We briefly sketch out the algorithm and then describe how to locate the branch-off point. The following are the main steps:

1. We plan a nominal trajectory σ_{nom} to the nominal touchdown site. The nominal touchdown site is indicated by the terminal point of the route guide $\sigma_{\text{rg}}(t_f)$. There is a requirement that this trajectory satisfies constraints everywhere except within an ϵ_{land} distance from the terminal point. This is because the terminal point may have obstacles, which has precipitated planning to alternate touchdown points.
2. We then compute a branch-off point \mathbf{b} on σ_{nom} using (23). σ_{nom} is then clipped up to this point.
3. For each touchdown site $g(i)$, we plan a path $\xi_{\text{br},i}$ from \mathbf{b} using a Dubins primitive and interpolating in \mathcal{Z} .
4. For each path $\xi_{\text{br},i}$, we copy the speed profile $v(\tau)$ from the route guide σ_{rg} and invoke the dynamics projection filter to get a trajectory $\sigma_{\text{br},i}$ as $\sigma_{\text{br},i} = \text{DPF}(\mathbf{b}, \xi_{\text{br},i}, v)$.
5. If $\xi_{\text{br},i}$ is valid, we concatenate it with σ_{nom} and append the trajectory to the set of solutions to return.

Fig. 21(c) illustrates how such a point can be calculated using the maximum glide-slope during the landing segment. Given a landing zone radius of r , an approach glide slope of γ_{approach} and a maximum glide slope of γ_{max} , we choose

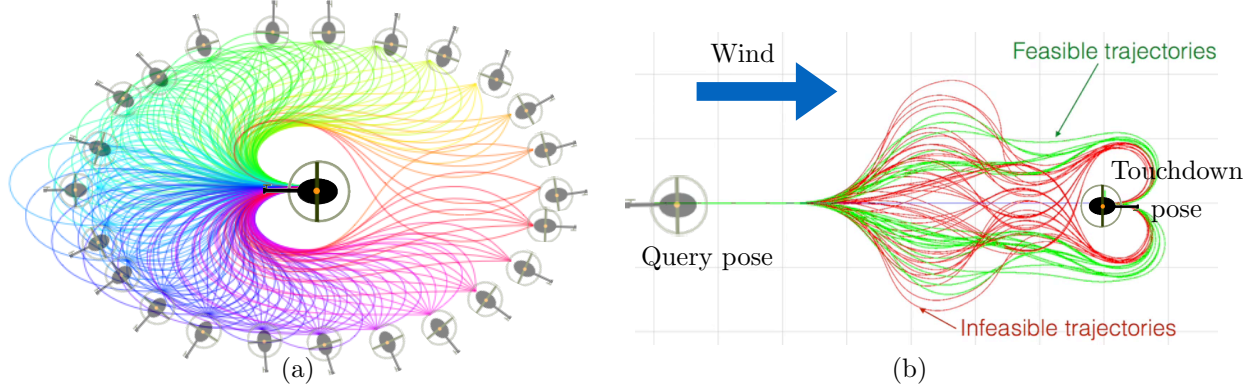


Figure 22: (a) A unit circle library of maneuvers. This is the recursive unit of the end game library. From the origin, maneuvers are computed backwards to different poses on the unit circle. (b) Test time use of the end game library. Given the wind direction, the library is oriented to point into it. Given the query, a set of candidate trajectories is returned. All infeasible trajectories are pruned.

the branch-off point to be at a distance d from the terminal point computed as:

$$d = r \frac{1}{1 - \frac{\tan \gamma_{\text{approach}}}{\gamma_{\text{max}}}} \quad (23)$$

For our application, $r : 50$ m, $\gamma_{\text{approach}} : 5^\circ$ and $\gamma_{\text{max}} : 10^\circ$, $d \approx 100$ m.

7.6.2 Glide slope from torque constraints and autorotation

We now discuss how we deal with two of the helicopter chart constraints - the torque and autorotation limits charts in Section 3.3. We want to solve surrogate path planning problems that would result in trajectories that end up respecting such charts. We will do this by deriving a bound on the glide slope γ (positive for descending, negative for climbing). This is then used in surrogate path planning problems as described in Example 1.

We first describe a bound obtained from the torque charts in Section 3.3. Let v_a be the maximum commanded airspeed. Given the operating temperature and pressure, we can look up the maximum level flight torque $\tau_{\text{level}} = \text{GetLevelTorque}(v_a)$ from the chart in Fig. 6(b). Since the total maximum torque is τ_{max} , this gives us a budget of $\tau_{\text{climb}} = \tau_{\text{max}} - \tau_{\text{level}}$ for climbing. We can use Fig. 6(c) to back out a bound on the vertical climbing speed $v_z = \text{GetClimbSpeed}(\tau_{\text{climb}})$. Note v_z is negative. This is used to obtain a negative bound on the glide slope $\gamma \geq \frac{v_z}{v_a}$.

We next describe a bound obtained from the auto-rotation charts in Section 3.3. Fig. 6(c) shows a chart of the descent speed versus the level flight speed, where certain sections of the chart are not in autorotation. Let v_a be the maximum commanded airspeed. In Fig. 6(c), we find the maximum descent speed v_z such that for all speeds below it, the combination is not in autorotation. Note v_z is positive. This is used to obtain a glide slope $\gamma \leq \frac{v_z}{v_a}$.

Note that we may have an asymmetric glide slope constraints as a result of these different charts. There is an additional constraint arising from the maximum vertical speed (both climbing and descending). All these constraints are combined.

7.6.3 End game library to deal with constrained landing and LTE charts

So far in our discussions, we have made an implicit assumption about the descent leg of the mission. We have assumed that the approach direction as specified by the waypoints of the mission is in fact desirable. However, when the approach direction is not aligned to face the wind direction, this is not the case. This is because of potential violations of the LTE chart as described in Section 3.3.

Satisfying the LTE constraints can be tricky. Nominally if the approach direction is aligned at π rad with the wind direction, then this constraint is trivially satisfied and we do not need to take any action. However, in the extreme

case when the approach direction is aligned at 0 rad with the wind direction, this requires a maneuver known as the *button-hook*. This is a maneuver that changes the heading of the vehicle by π rad at such a speed so as to always stay in valid areas of the LTE chart. This maneuver can get difficult since it requires reasoning spatially and in velocity space. It can get additionally difficult when there are obstacles near the landing zone.

We solve such problems with an *end game library* depicted in Fig. 22. The end game library is a set of motion primitives computed in a backwards fashion from the touchdown point outwards. Primitives in the library are dynamically feasible trajectories created offline. The motivation for creating a library is that the pre-landing phase is relatively challenging requires planning both spatially and velocity to satisfy all constraints. On the other hand, since the maneuvers are restricted to a radius around the landing site, a lot of pre-computation can be leveraged. Hence we employ a precomputed library.

7.6.4 Take-off library respecting the H-V curve

The final design consideration we discuss is the constraint due to the Height-Velocity curve as described in Section 3.3. We observed empirically, that the only time such a constraint is in danger of being violated is during take-off. This happens when the take-off profile is too steep. On the other hand, a very shallow take-off profile is likely to collide with trees around the take off location.

We found that this constraint can be satisfied by pre-computing a library of take-off maneuvers. Each of these maneuvers are computed by forward integrating different choices of forward and vertical velocity profiles. At execution time, the planner iterates through the library to select the maneuver that is not only collision free but also achieves the desired terminal climb rate.

8 Guaranteeing Safety – the Trajectory Executive

The role of the trajectory executive is to guarantee safety of the system (Fig. 9). It does so by taking as input the finite horizon trajectory computed by the planning algorithm and checking whether it is guaranteed safe, i.e. if there exists an evasive (or emergency) maneuver that originates from the trajectory. An evasive maneuver is an infinite horizon trajectory that lies in the known free state space. Usually such trajectories are loiter patterns (at high speeds) or hover commands (at lower speeds).

Determining such maneuvers is computationally challenging, especially when the robot has non-linear dynamics. Additionally, it is required that the safety evaluation have a low worst-case response time so that it can keep up with the rate of online motion planning solutions and recently detected obstacles, such as popup no fly zones. In order to ensure online capability of the safety evaluation, the problem is decoupled from goal-based planning and split into an offline and online part. The offline part precomputes an optimized set of trajectories enabling the robot to reach a safe state and stay within the known obstacle free region for an infinite time horizon. These safe trajectories (shown in Fig. 23) respect dynamic limits, account for wind, and are generated in a manner similar to the trajectory optimization approach described in Section 6. The online part stitches these to the safe portion of the planned trajectory for possible execution.

8.1 Safety definition

Let $x(t)$ be the state of the robot at time t . The workspace of the robot is defined as \mathcal{W} and the occupancy of the robot system in the workspace at a certain state is given as $\mathcal{A}(x(t)) \subset \mathcal{W}$. The observed space of the workspace, i.e. the area cleared by the sensor at a given time t is denoted as $\mathcal{K}_t \subset \mathcal{W}$. The occupancy of the known obstacles at time t is given by $\mathcal{O}_t \subset \mathcal{K}_t \subset \mathcal{W}$. Let $\sigma(x)$ be a trajectory rooted at state x and let $\sigma(x, \tau)$ be the state of the vehicle at time τ . In a static environment, any state x along a trajectory followed by the vehicle can be considered terminally safe if there exists a safe trajectory $\sigma(x)$ which completely lies inside the known obstacle-free space at that time. Equation (24) presents this definition formally:

Definition 1 (Motion Safety).

$$\forall t, \forall \tau, \exists \sigma(x) : \mathcal{A}(\sigma(x, \tau)) \subset (\mathcal{K}_t \setminus \mathcal{O}_t) \quad (24)$$

In the next sub-section we discuss how this safety definition is used to ensure vehicle safety.

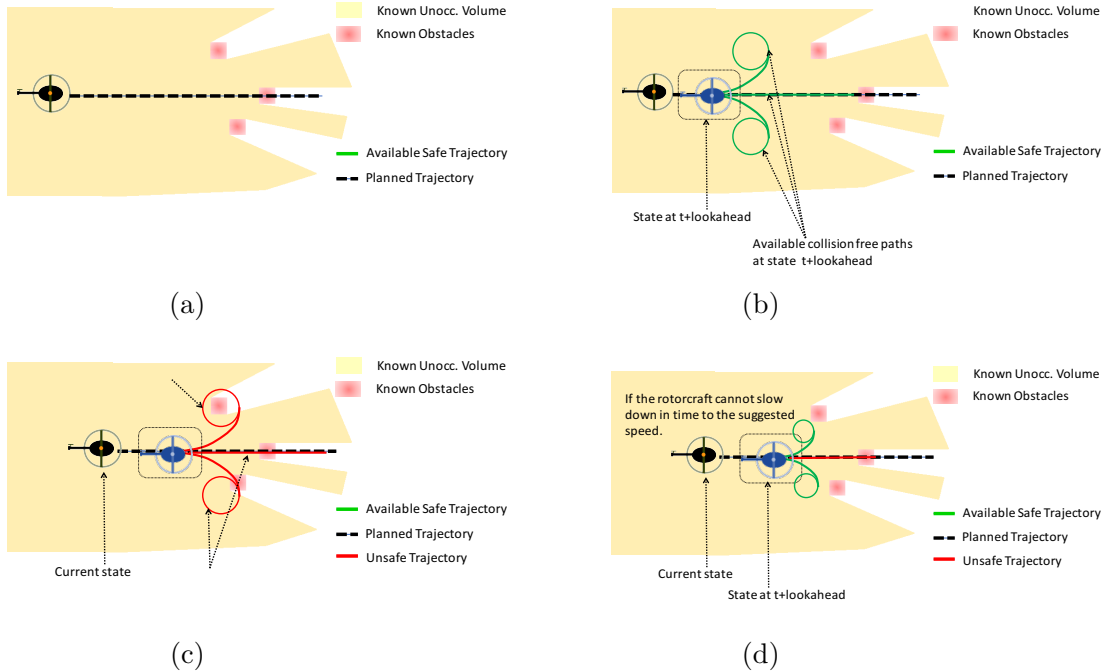


Figure 23: The executive ensures trajectories are guaranteed safe. (a) The motion planner sends a planned trajectory to the executive (b) The safety checker queries an emergency maneuver library to ensure maneuvers exist from the trajectory that lie entirely in known free space. The maneuver is attached to the trajectory and the guaranteed safe trajectory is passed to the control system (c) Eventually the safety checker encounters a case where the library is unable to find a maneuver for the trajectory (d) The safety checker tries to slow down the vehicle until the library is successful. If not, the safety checker reverts to the previous guaranteed safe trajectory.

8.2 Ensuring safety

Safety is ensured by two sub-modules - safety checker and evasive maneuver library (Fig. 9). The library consists of a diverse set of evasive maneuvers $\{M_{evas}\}$ that fulfill the definition of safety as described in (24), and whose construction is detailed in (Arora et al., 2015). The safety checker is a sub-module whose purpose is to ensure that the vehicle stays in known, safe space, by consistently checking the safety of the last planned trajectory and stitching maneuvers from the aforementioned library to it. Given the complexity of the algorithm we separate the significant parts into four sections: *New Solution Validation*, *Maneuver Retrieval*, *Maneuver List Update*, and *Executing Trajectory Update*.

8.2.1 New solution validation

Algorithm 9 shows how the executive nominally operates during a mission. At each iteration, the executive is allowed to query for a new planning solution σ_{new} , should one exist (as seen on line 5). If part of the received solution is perceived to be safe at least beyond some time threshold (as described in algorithm 10) it is stitched to the trajectory set aside for execution σ_{exec} (to be followed by the rotorcraft). To ensure safety along a trajectory above some time horizon, one only needs to check if the trajectory up to that horizon is safe and ends in a terminally safe state. Similarly, in algorithm 10 we check if there exists at least one safe maneuver between t_{min} and t_{safe} to ensure the trajectory is safe at or above some time t_{min} . As time moves forward so does the look ahead time for following a trajectory $t_{lookahead}$. We pass this along with some buffer to 10 to ensure the safety of the trajectory in question is at least above some horizon from the current look ahead time.

8.2.2 Maneuver retrieval

Algorithm 11 shows how a list of evasive maneuvers are retrieved for a given trajectory σ along a timespan between t_{min} and t_{max} . Starting from the end, a set of states along σ are queried for corresponding evasive maneuvers. In the

event the *single* parameter is set to true (as it's passed in algorithm 10, line 3) the function will break upon the first safe maneuver found, as seen on line 9. When the *Evasive Maneuver Library* is queried for a set of maneuvers for a given state on line 5, it is important to note that each maneuver returned consists of both an entry part (computed online) and terminal part (computed offline). This allows for the stitching of loiters with velocity profiles that differ significantly from that of a given state \vec{x}_{curr} along the trajectory without sacrificing dynamic constraints.

8.2.3 Maneuver list update

In algorithm 9, the stitched trajectory σ_{curr} can be thought of as the trajectory the rotor-craft would be commanded to follow if all space was free. Due to obvious limitations in sensing space over large distances, we only check the safety of this trajectory up to some shorter horizon $t_{horizon}$, as specified in line 9. We also keep a list of safe evasive maneuvers found thus far and update that list in algorithm 12. We keep a list of full maneuvers on purpose to account for cases of late obstacle detection or popup no fly zones. In updating the current set of evasive maneuvers, we retrieve a new set that is further out towards the horizon, then add any from the old set that satisfy a number of conditions, as seen on lines 3 - 6. In addition to the conditions of uniqueness (line 4) and safety (line 5), we check if the beginning time of the entry portion of the maneuver is behind some threshold t_{min} (line 3). This condition is used to clear maneuvers that have been passed by the current look ahead time $t_{lookahead}$. Note that by passing $t_{horizon}$ as t_{max} to algorithm 12 from algorithm 9, line 10, we can only ensure the safety of the stitched trajectory σ_{curr} up to that horizon.

8.2.4 Executing trajectory update

In line 11 we call algorithm 13 to update the soon to be executed (at least partially) trajectory σ_{exec} and the furthest safe maneuver found for it M_{exec} . After retrieving the maneuver with the furthest entry time, we determine if it is still beyond the look ahead time, passed as t_{min} . If the entry time is ahead, we set the corresponding maneuver as the new M_{exec} and update σ_{exec} by stitching it to a portion of the stitched trajectory (passed as σ_{stich}), and this maneuver's entry and terminal parts in lines 3-7. As a separate condition (in line 8), if the entry time of the last cached maneuver M_{exec} is under the look ahead time we expect the rotor-craft to continue the following the maneuver indefinitely until a new safe trajectory is processed. In case the terminal portion of the maneuver happens to be a loiter motion, we compute the lookup time within the loiter $t_{terminal}$ and stitch this to our commanded trajectory σ_{exec} . The calls to retrieve terminal motions in lines 9 and 14, imply a circular indexing scheme is used to return a finite portion of the motion when it is so happens to be a loiter. Otherwise, it can be ignored for motions that end in a complete stop.

Algorithm 9: Executive ()

```

1 while mission active do
2    $t_{lookahead} = \text{GetLookAheadTime} ();$ 
3    $\sigma_{new} = \text{GetNewPlannerSolution} ();$ 
4   if  $\sigma_{new} \neq \emptyset$  then
5     if  $\text{IsTrajectorySafe} (\sigma_{new}, d_{buffer} + t_{lookahead})$  then
6        $\sigma_{curr} = \text{Stitch} (\sigma_{exec}, \sigma_{new});$ 
7   if  $\sigma_{curr} \neq \emptyset$  then
8      $t_{horizon} = t_{lookahead} + d_{horizon};$ 
9      $MSet_{cur} = \text{UpdateEvasiveManeuvers} (t_{lookahead}, t_{horizon}, \sigma_{curr}, MSet_{cur});$ 
10     $\sigma_{exec}, M_{exec} = \text{UpdateExecutingTrajectory} (\sigma_{exec}, M_{exec}, \sigma_{curr}, MSet_{cur}, d_{buffer} + t_{lookahead});$ 
11   $\text{Execute} (\sigma_{exec});$ 

```

8.3 Improvements to trajectory executive

Over the course of flight testing, we added a few features to address commonly occurring issues. We discuss them briefly.

Algorithm 10: IsTrajectorySafe (σ, t_{min})

```
1  $M_{evas} = \emptyset$ ;  
2  $t_{safe} = \text{TimeToCollision}(\sigma)$ ;  
3  $M_{evas} = \text{RetrieveEvasiveManeuvers}(t_{min}, t_{safe}, \sigma_{curr}, true)$ ;  
4 if  $M_{evas} = \emptyset$  then  
5   return false;  
6 return true;
```

Algorithm 11: RetrieveEvasiveManeuvers ($t_{min}, t_{max}, \sigma, single = false$)

```
1  $t = t_{max}$ ;  
2  $MSet_{safe} = \emptyset$ ;  
3 while  $t \geq t_{min}$  do  
4    $\vec{x}_{curr} = \sigma_{curr}(t)$ ;  
5    $MSet = \text{LookupManeuvers}(\vec{x}_{curr})$ ;  
6   for  $M \in MSet$  do  
7     if  $\text{IsSafe}(M)$  then  
8        $MSet_{safe} = \{MSet_{safe}, M\}$ ;  
9       if  $single = true$  then  
10        break  
11   Decrease  $t$ ;  
12 return  $MSet_{safe}$ ;
```

Algorithm 12: UpdateEvasiveManeuvers ($t_{min}, t_{max}, MSet, \sigma$)

```
1  $MSet_{new} = \text{RetrieveEvasiveManeuvers}(t_{min}, t_{max}, \sigma)$ ;  
2 for  $M \in MSet$  do  
3   if  $\text{EntryTime}(M) \geq t_{min}$  then  
4     if  $M \notin MSet_{new}$  then  
5       if  $\text{IsSafe}(M)$  then  
6          $MSet_{new} = \{MSet_{new}, M\}$ ;  
7 return  $MSet_{new}$ ;
```

Algorithm 13: UpdateExecutingTrajectory ($\sigma_{exec}, M_{exec}, \sigma_{stich}, MSet, t_{min}$)

```
1  $t_{safe} = 0$ ;  
2  $M_{evas} = \text{getFurthestEntryManuever}(MSet)$ ;  
3 if  $M_{evas} \neq \emptyset$  then  
4    $t_{safe} = \text{EntryTime}(M_{evas})$ ;  
5 if  $t_{min} < t_{safe}$  then  
6    $M_{exec} = M_{evas}$ ;  
7    $\sigma_{exec} = \text{StichUpTill}(\sigma_{exec}, \sigma_{stich}, t_{safe})$ ;  
8    $\sigma_{exec} = \text{Stich}(\sigma_{exec}, \text{GetEntry}(M_{exec}))$ ;  
9    $\sigma_{exec} = \text{Stich}(\sigma_{exec}, \text{GetTerminal}(M_{exec}, 0))$ ;  
10  $t_{entry} = \text{EntryTime}(M_{exec})$ ;  
11 if  $t_{min} \geq t_{entry}$  then  
12    $d_{entry} = \text{EntryDuration}(M_{exec})$ ;  
13    $t_{terminal} = t_{min} - t_{entry} - d_{entry}$ ;  
14    $\sigma_{exec} = \text{Stich}(\sigma_{exec}, \text{GetTerminal}(M_{exec}, t_{terminal}))$ ;  
15 return  $\sigma_{exec}, M_{exec}$ ;
```




Characteristics	Boeing H-6U (Unmanned Little Bird)	Bell 206B (Jet Ranger)	Bell 206L and 206L-3 (Long Ranger)
			
Project phase	Phase I	Phase II and III	Phase III
Engine power	485 kW (650 hp)	310 kW (420 hp)	485 kW (650 hp)
Empty weight	998 kg (2200 lbs)	730 kg (1609 lbs)	998 kg (2200 lbs)
Max takeoff weight	1610 kg (3100 lbs)	1451kg (3200 lbs)	1882 kg (4150 lbs)
Rotor diameter	8.38 m (27.4 ft)	11.28 m (37 ft)	11.28m (37 ft)
Max speed	268 km/h (145 KTAS)	219 km/h (118 KTAS)	232 km/h (125 KTAS)
Rate of climb	10.5 m/s (2070 ft/min)	6.6 m/s (1300 ft/min)	6.6 m/s (1300 ft/min)
Service ceiling	6096 m (20000 ft)	3114 m (13500 ft)	6096 m (20000 ft)

Figure 24: The 3 different helicopter platforms used for evaluation

8.3.1 A more robust safety check with multiple maneuvers

In the basic version of safety checking, the algorithm only has to ensure if atleast one state in the trajectory being executed is safe. However, this can be quite brittle in practice if the known space has pockets of unknown space. This can prematurely trigger evasive maneuvers even though the currently executed trajectory is in fact safe.

A simple fix is to search for multiple safety maneuvers along the current path and maintain them. Even if some of the maneuvers become invalid with time, others are always available and the system continues following the nominal trajectory

8.3.2 Checking against no-fly-zones

A commonly occurring situation is when an popup no-fly-zone appears in close proximity to the system such that the trajectory planner is unable to compute a feasible solution and no evasive maneuver exists. The system in this case continues to fly through the no-fly-zone in an undefined manner as the safety assumptions are violated.

A simple fix to this is to find an evasive maneuver that spends the minimal time inside the no-fly-zone. This requires not deleting maneuvers intersecting with no-fly-zones but keeping track of the amount of time spent inside one. When it comes to selecting a maneuver, the one with the least time is selected.

8.3.3 Incorporating wind in evasive maneuvers

Initial set of evasive maneuvers ignored the wind and were circular in nature. These would be dynamically infeasible given a wind vector. These maneuvers were changed to instead contain a series of arcs that respected the wind using the optimization techniques in Section 6.

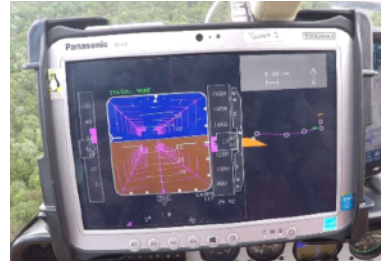
9 Experiments

9.1 System description

Over the course of approximately 3 years, the trajectory planning system (Paduano et al., 2015) was tested on a number of helicopters as shown in Fig. 24. Tests were facilitated through Boeing, Near Earth Autonomy, Executive Helicopters, and Aurora Flight Sciences. Computation was completely on-board on proprietary flight-ready computers provided by Near Earth Autonomy. The sensor suite was also proprietary Near Earth Autonomy which included an actuated 2D Laser, a strap down fiber-optic IMU, high resolution cameras and GPS. The LIDAR scans have a 100



(a)



(b)

Figure 25: Setup for pilot-in-the-loop experiments. (a) View from the cockpit of the pilot where the display is next to the instrument panel (b) Close-up of the display. Left panel shows desired attitudes and airspeeds. Right panel shows the trajectory up to a horizon.

degree FOV providing 42,000 point measurements per second with a max range of 1.1 km. All sensors are time synchronized.

The system was executed in one of two modes - fly-by-wire or pilot-in-loop. In both of these modes, there is a safety pilot on-board.

1. *Fly-by-wire (FBW)*: The system is fully autonomous mode where it sent commands directly to the helicopter flight control system.
2. *Pilot-in-loop (PIL)*: Exactly same as FBW, but the command is visualized in a pilot display (Fig. 25) which is then tracked by a pilot. The pilot tracks the commanded positions and speeds as long as it is safe to do so. The decision to use this setup was made at the program level to allow the system to transition easily to newer helicopters.

9.2 Summary of real flight tests

9.2.1 Overview of testing and capability progression

Table 3: Number of flight tests performed during the project

Test Type	Phase I (runs)	Phase I (days)	Phase II (runs)	Phase II (days)	Phase III (runs)	Phase III (days)
Unit Test	(*)	(*)	34	8	214	28
Integrated	24	2	61	11	284	51
Total Test	24	2	95	19	498	79

The testing of this system spanned over a period of 4 years, and was divided into three phases:

1. *Phase I: Test full-stack autonomy* - Fly-by-wire, manually specified routes, system asks a human operator for approval when selecting touchdown sites.
2. *Phase II: Test complex mission* - Pilot-in-loop, deal with complex auto-generated routes, no human operator for touchdown selection.
3. *Phase III: Test system robustness* - Pilot-in-loop, fly longer missions with multiple take-offs and touchdowns, deal with varying wind conditions.

There were two categories of flight tests -

1. *Unit tests* - Flights that tested certain capabilities, not necessary for entire software stack to be active.
2. *Integrated tests* - Flights that executed the software stack as it would be used when deployed.

Table 3 shows the total flight tests performed across all phases². The takeaway from this table is *number of tests increase exponentially with time* - each bug discovery needs a new verification test, each added feature results in regression testing, the more complex the system becomes the more it needs to be tested to verify behaviors.

Note on test process. These tests were performed with the help of a large staff of engineers and developers. The actual test procedure is solely carried out by flight personnel - developers are not allowed to interfere with this process. Performing tests with a full scale helicopter is a complex task that requires a lot of time and resources for preparation, execution, and post flight evaluation and data analysis. From the table, one can infer that a typical test day has ≈ 6 runs, each run takes around 15 minutes from from takeoff to land. The low number of test runs per day, reflects the amount of preparation required prior to flight, fixing software or hardware issues, wait time or a test canceled due to weather related events, and unexpected air traffic in the test locations.

Capability progression. To describe progression of the project, we divide each component of the trajectory planner into atomic units called *capabilities* as described in Table. 4. These capabilities represent features that were developed together and integrated into the system. We now show how these capabilities evolved over time in Fig. 26. Interestingly, while initially the focus was on safety and completion of the mission, over time the importance shifted to respecting wind and performance chart constraints. The development time also increased as the system got more complicated. Finally, the receding horizon arose out of the need to scale the system to long missions. The key takeaway is *wind and smooth flight matters* - since this influences both the architecture and methods, it should be dealt with sooner rather than later.

Table 4: List of capabilities for different components of the trajectory planner

Capability	Section	Description
Route Optimizer		
TimeOPT	6.3	Time optimization for optimal scheduling of speeds given a set of straight lines and arcs
Corridor	6.2	Enforce flight corridor constraints on the computed route
CurvOPT	6.2,6.4	Optimize the curvature of arcs for smoothness and to satisfy higher order constraints
WindOPT	6.5	Add wind vector to dynamics constraints and solve for corrected ground speed trajectory
Trajectory Planning Algorithm		
Obs	7.4	Designing expert planners that satisfy time to collision constraints
NFZ	7.4	Designing expert planners that satisfy no-fly-zone constraints
MultiTS	7.6.1	Plan to multiple touchdown sites and switch when nominal one is infeasible
Learn	7.5	Learn an ensemble of expert planners from data
Chart	3.3	Enforce the chart constraints in the expert planners
RHP	7.1	Receding horizon planning architecture for invoking expert planners
Trajectory Executive		
SafeCheck	8.2	Trajectory executive checks if the executed trajectory is safe
Robust	8.3.1	A more robust safety logic that searches for multiple maneuvers
NFZCheck	8.3.2	Evasive maneuvers checked against NFZ to minimize the time spent in it
WindEML	8.3.3	Evasive maneuvers that satisfy wind constraints

Tests and Failures. Fig. 26 shows a sample set of 4 key flight tests from different phases and which components were tested. As the tests progress, it is not possible to exercise all components. The key takeaway is that *route optimizer and safety are heavily tested* - invest more resources into them. For many tests it suffices to get by with a route guide and safety running. However, a failure in the route optimizer disqualifies the entire flight test. The red boxes indicate that there were modules still failed (did not perform as expected) during the test despite significant simulation testing. These are *all failures with live sensor data* - undetected obstacles in the landing zone (MultiTS), not estimating wind correctly (Chart) and sensor not clearing free space (Robust).

Fig. 27 shows the fraction of failures *over the entirety of Phase III*. This was the most stable configuration of flight software, hence the failures are meaningful. 39% of the failures were due to system errors - this is to be expected in a

²We do not have accurate logs of unit testing during Phase I - however the number is higher than unit tests for any of the other Phases.

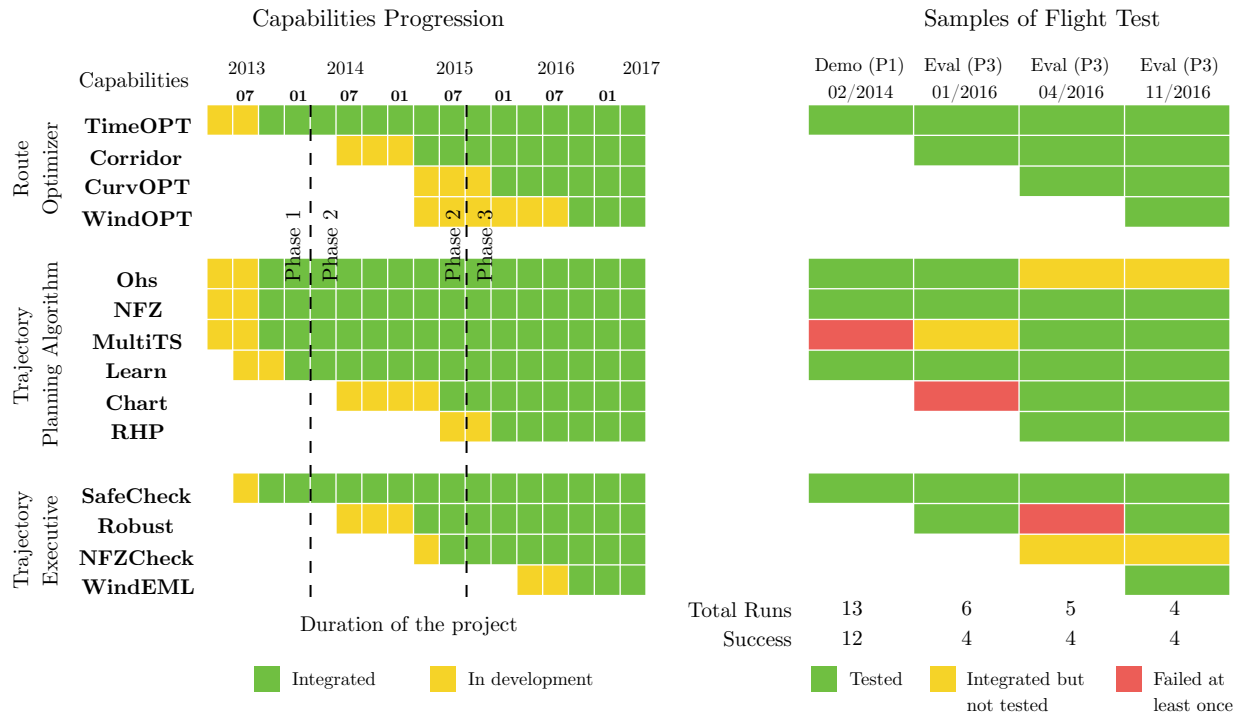


Figure 26: Capability progression and testing. (Left). A list of capabilities and when they were developed / integrated into the system. (Right). Four key flight tests and the components that were tested. (Below). The total number of runs during the test and the ones that were successful.

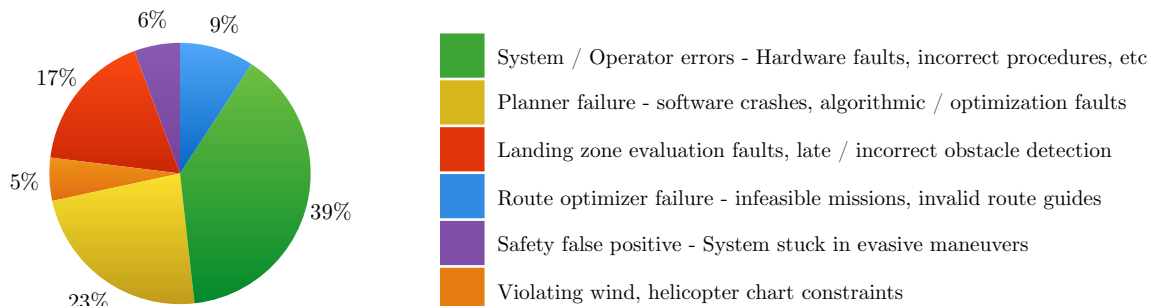


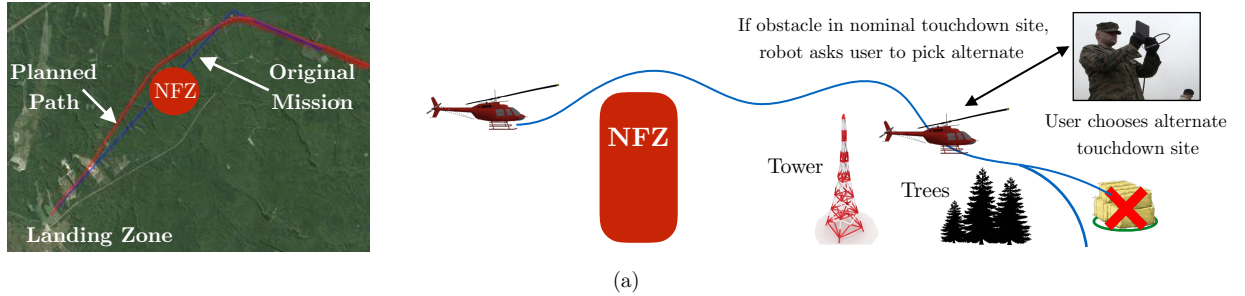
Figure 27: Breakup of the failures from all flight tests in Phase III.

complex system that is continually under-development, tested by external parties and where the hardware is exposed to difficult atmospheric conditions. 23% of the failures were due to the planning software - the planner is at the end of the pipeline and at the mercy of outputs arising from all of the other components. 17% of the failures were due to landing zone detection - detecting obstacles in the landing zone from hundreds of meters away is very difficult and error prone.

9.2.2 Analysis of a set of fly-by-wire flight tests

Setup. We now describe a flight test day comprising of 9 fly-by-wire runs as shown Fig. 28(a). The test was conducted during Phase I in Quantico, VA, using the ULB helicopter. The components for this test is specified in Fig. 26 under Demo (P1) column.

The mission is as follows: the system has to fly a L-shaped mission while avoiding no-fly-zones, towers, trees and obstacles on the landing site. As it comes in to land, if the nominal touchdown site is not feasible, it communicates



Run	Enroute Phase				Approach Phase					Landing Phase					Success
	Time (s)	Max speed (m/s)	Altitude (m)	Max roll (deg)	Max descent (m/s)	Max glide (deg)	NFZ reaction time (s)	Tower reaction dist (m)	Tree reaction dist (m)	Obstacle in landing?	Replan time (s)	User response (s)	Replan distance to LZ (m)		
1	197.6	52.27	650	30	6	5	-	-	245	N	0.20	2.1	274	Y	
2	195.7	51.60	640	30	6	5	-	-	210	N	0.38	2.0	233	Y	
3	198.4	52.22	640	22	5	5	1.4	-	270	N	0.30	3.2	221	Y	
4	238.3	51.60	650	30	4	5	1.0	-	400	Y	-	-	-	Y	
5	265.3	52.83	610	30	4	6	-	-	-	Y	-	-	-	Y	
6	210.7	52.73	650	30	5	6	-	-	635	Y	-	-	-	Y	
7	214	52.63	640	30	5	5.5	-	-	-	N	0.30	2.7	248	Y	
8	185.1	52.47	650	36	6	7	4.4	800	-	N	-	-	-	N	
9	207.2	52.52	650	30	5	5.5	-	-	-	N	0.40	2.2	214	Y	

(b)

■ Near critical ■ Critical

Figure 28: A series of fly-by-wire flight tests (a) (Left) One of the missions in the flight test overlaid on the satellite image of Quantico, VA. (Right) An illustration of the missions which included NFZ, tower and tree avoidance as well as a user interfacing during touchdown. (b) Statistics of all the runs in the flight test

with a user (a marine on the ground) and offers them alternate touchdown sites. The user picks one and the system comes in to touchdown.

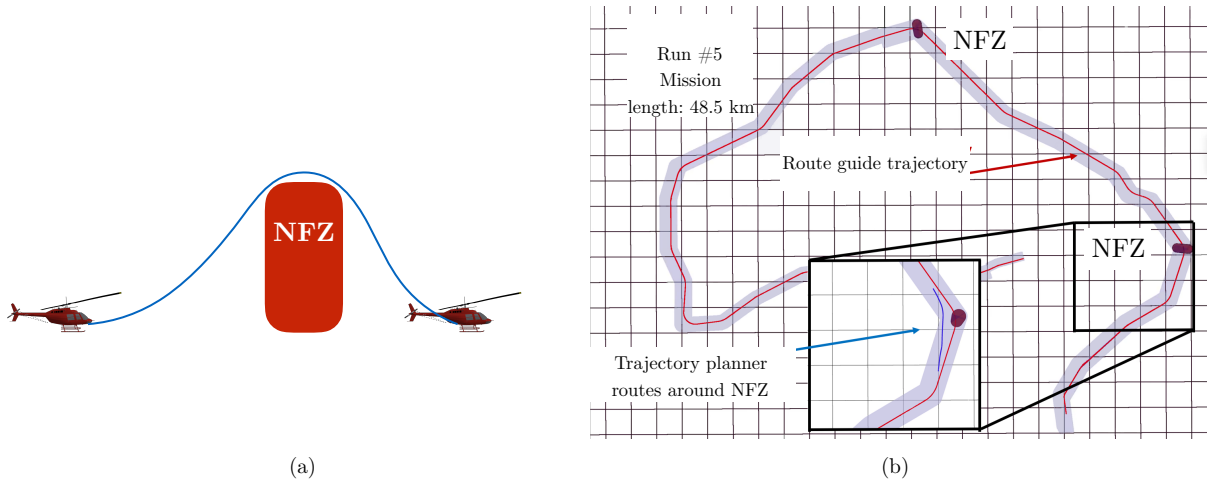
The selected planners in the static ensemble were: $\{ \text{CHOMP}^1, \text{CHOMP}^2 \}$. These planners were selected based on a database of planning problems generated in simulation.

Analysis. The following are the highlights of the experiment:

1. 8/9 runs were successful. On the run that failed, the system was not able to verify whether the landing zone was feasible or not. Hence, it chose to abort and wave-off.
2. The system was banking aggressively up to 30 deg (although within limits). This was because there was no optimization for smoothness in route guide generation (was introduced in Phase III). It did violate the constraint 36 degrees - this was due to a spline fitting error (which was subsequently enforced).
3. The system occasionally came close the glide slope limit of 6 deg (violating it once at 7 deg) - this is on account of not incorporating auto-rotation constraints (incorporated in Phase III).
4. It was 100% successful in avoiding obstacles / no-fly zones.
5. It was successful at re-planning to alternate touchdown sites, and getting approval from user early enough. There is a point of no-return beyond which the system must abort landing due to reachability constraints - this distance is 200m. It did come close to that point on the final run.

Takeaways. The experiment led to the following takeaways

1. The system should commit as late as possible to a touchdown site, i.e. able to taxi at low speeds.



Run	Overall			Takeoff		Enroute				Approach / Landing				Success
	Time (s)	Max ground speed (m/s)	Altitude (m)	Error in tracking speed (m/s)	Error in tracking speed (m/s)	Max roll (deg)	NFZ reaction distance (m)	Planner selected from ensemble	Max Tracking Error (m)	Safety invoked ?	Error in tracking speed (m/s)	Landed at nominal ?	Replan distance to LZ (m)	
1	427.5	40.12	548.6	1.12	13.9	25	526.3	CHOMP ¹	45.72	N	3.6	Y	-	Y
2	467.1	34.47	548.6	3.1	8.2	20	684.9	RRT*Tun ¹	30.48	N	3.1	Y	-	Y
3	510.9	40.12	548.6	2.57	12.86	25	696.0	RRT*Tun ¹	60.9	N	4.1	N	140	Y
4	507.7	40.12	548.6	3.1	14.92	26	510.1	RRT*Tun ¹	81.3	Y	-	-	-	N
5	1641	40.12	650.0	2.57	10.29	22	1666.7	RRT*Tun ¹	60.9	N	1.54	Y	-	Y

(c)

Figure 29: A series of pilot-in-the loop flight tests (a) Illustration of the mission, i.e. take-off to landing with NFZ avoidance enroute (b) Plot of Run 5, the toughest of the runs, which spanned 48.5 km and had multiple NFZ. (c) Statistics of all the runs in the flight test

2. The system should incorporate performance charts and wind information.

9.2.3 Analysis of a set of complete pilot-in-the-loop missions

Setup. We now describe a flight test day comprising of 5 pilot-in-loop runs as shown Fig. 29. The test was conducted during Phase 3 in Pittsburgh, PA, using the BEI-206 helicopter. The components for this test is specified in Fig. 26 under Eval (P3) column - all modules other than **WindOPT** and **WindEML**.

The missions area as follows: the system has to takeoff, avoid no-fly-zones that dynamically appear and land in a landing zone. As it comes in to land, there may be small obstacles causing it to choose alternate touchdown sites on its own. No communication occurs with a user.

The selected planners in the static ensemble were: $\{ \text{CHOMP}^1, \text{RRT}^*\text{Tunnel}^1 \}$. These planners were selected based on a database of planning problems generated in simulation.

Analysis. The following are the highlights of the experiment:

1. 4/5 runs were successful. On the run that failed, the system executed a safety maneuver. Analysis showed that the sensor was unable to properly clear the volume of free space due to presence of rain. A more robust safety module would have succeeded.
2. Tracking error of speed during take-off was good (≤ 3.1 m/s), i.e. the system is taking off like a pilot

3. Tracking error of speed enroute was poor ($> 10\text{m/s}$). This was because the system did not incorporate live wind feedback at the time. The pilot had to adjust airspeed to take into account wind.
4. Both of the planners in the ensemble were used, but $\text{RRT}^*\text{Tunnel}^1$ produced the selected trajectory more often. Upon analysis, we found that CHOMP^1 get stuck on a local minima because of the placement of no-fly-zones and flight corridors.
5. Maximum tracking error in terms of distance is within limits - the pilots use a different banking profile in presence of winds.
6. The re-planning distance to alternate touchdown site shows the ability of the system to react late to small obstacles in the landing zone.
7. Run 5 was the longest recorded flight in all of our missions - 48.5 km. To the best of our knowledge, this is the longest run by an autonomous helicopter avoiding no-fly-zones along the way.

Takeaways. The experiment led to the following takeaways

1. The system should incorporate live wind feedback - this was done soon with the incorporation of **WindOPT**.
2. The evasive maneuvers should also incorporate wind - this was done soon with the incorporation of **WindEML**.

9.3 Simulation: Evaluation of components

The following set of experiments are based on evaluation in simulation. A simulation is performed as follows: A digital elevation map is loaded and used for simulating sensor readings. The dynamics of the helicopter is simulated by model designed by sys-id of the actual helicopter response. The same planning software executed on real flight tests is executed in simulation. The use of ROS nodes for all of the components allows seamless switching from simulation to real flight testing.

9.3.1 Trajectory planning algorithm - static ensemble selection

Objective. We evaluate the static ensemble selection using the lazy greedy algorithm (Section 7.5.1). The objective is to quantify the performance gain over individual planners and examine the nature of the ensemble in different environments.

Setup. We create a training dataset of 6000 planning problems (Fig. 30(a)). In these, the helicopter is flying at 30 m/s from position $(0, 0, 0)$ to $(2500, 0, 0)$ with both headings are 0° . We evaluate the entire library of expert planners (Appendix B) on this. We extract a fixed dimension feature vector that measures the density of obstacles at a coarse granularity and samples the distance field locally. A random forest (Breiman, 2001) is used to learn the model. The two target planning problem distributions we test are “Forest” and “City”. There is a further categorization of “Known” (where the full environment is accessible, Fig. 30(b),(c)) and “Unknown” (where the environment is incrementally revealed, Fig. 30(d),(e)). More details in (Choudhury et al., 2015).

Analysis. Fig. 30(f) shows the ensemble selected for different environments. In all of the datasets, using an ensemble of planners reduced the loss significantly.

O 1. *A static ensemble of $\text{RRT}^*\text{Tunnel}^1$ and CHOMP^1 has good performance on most environments.*

Interestingly, on 3 out of the 4 datasets, the ensemble selected was $\text{RRT}^*\text{Tunnel}^1$ and CHOMP^1 . The reason these two planners performed well is because the optimal solution was always within a tunnel of the initial guess and the cost function was usually convex around the initial guess. For the “Known City” dataset, A^*^2 and $\text{RRT}^*\text{Tunnel}^1$ were chosen. This is because this environment required reasoning in a global sense, discretization effects were acceptable and on occasion when the solution happened to lie near the initial guess, the $\text{RRT}^*\text{Tunnel}^1$ was the only algorithm able to sample densely enough to get a solution.

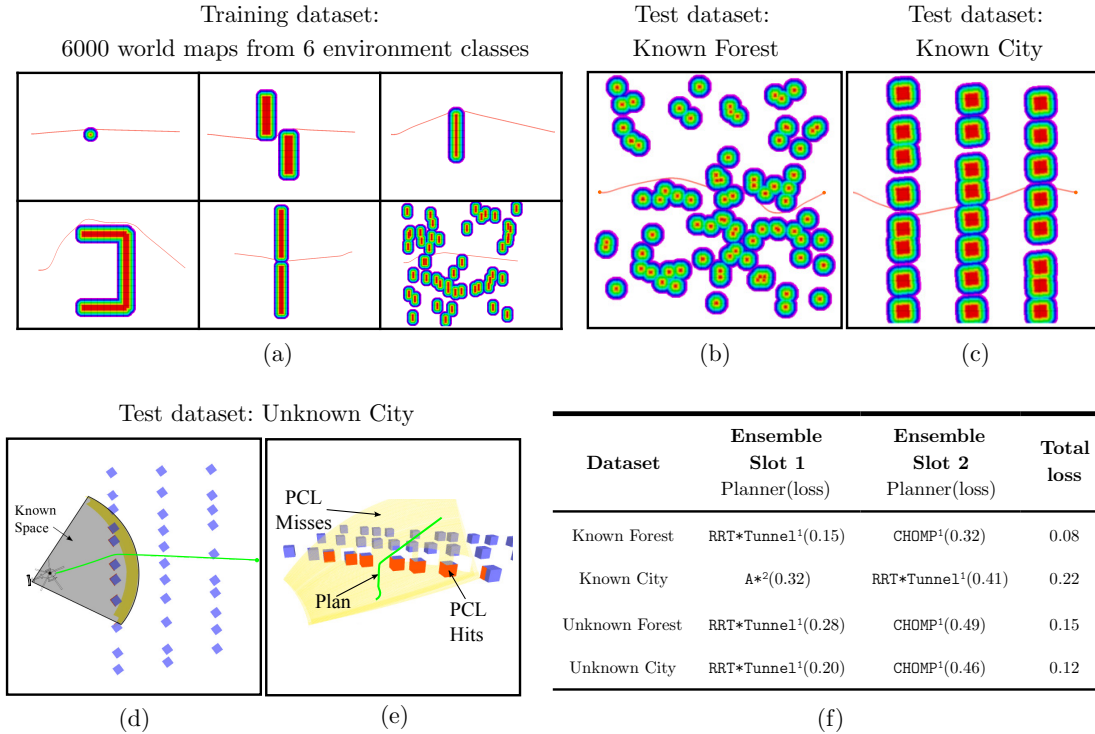


Figure 30: Evaluation of a static planner ensemble selection on simulated datasets. (a) A training dataset of diverse problems (b) “Known” category, i.e. the robot can see all the obstacles in the horizon. “Known Forest” has obstacles drawn from a Poisson distribution. (c) “Known City” has obstacles arranged in the form of a city block (d)(e) “Unknown” category which limits what the robot can see. (f) The performance of the ensemble on different test datasets.

9.3.2 Trajectory planning algorithm - dynamic ensemble selection

Objective. We evaluate the dynamic ensemble selection algorithm (Section 7.5.2). The objective is to quantify the performance gain over a static ensemble.

Setup. Fig. 31 shows a dataset of planning problems where no-fly-zones have to be avoided at high speeds. The planning problems are harder, requiring avoidance maneuvers in x,y, and z with tighter curvature constraints. A database of 1000 planning problems are created by placing 4 random no fly zones in a stretch of 2km. We evaluate the entire library of expert planners (Appendix B) on this *except* CHOMP¹ and CHOMP² (these problems were too non-convex for them to solve).

Analysis. Fig. 31(b) shows statistics of different planners. If we examine RRT*Tunnel¹, we see that its average cost is 0.256 and conditional cost on success is 0.107. This shows the general purpose planner nature - while it solves many problems, when it succeeds its performance is not drastically better. On the other hand, if we examine FixedDescent³, we see that it has an average cost of 0.973 but when conditional cost on success of 0.009. This shows the precision planner nature - it fails on most of the problems, but the small fraction on which it succeeds, it is near optimal.

O 2. *When faced with a complex combination of no-fly-zones, a static ensemble no longer suffices. A dynamic ensemble is required.*

Fig. 31(c) shows that the best static ensemble of 3 planners is { RRT*Tunnel¹, RRT*Workspace², RRT*Tunnel² } with loss 0.065. Fig. 31(d) shows the behaviour of the dynamic ensemble with different learners. The dynamic ensemble brings down the risk dramatically from dynamically selecting a single planner. The random forest model has the lowest validation loss of 0.021, which is significantly lower than the static ensemble.

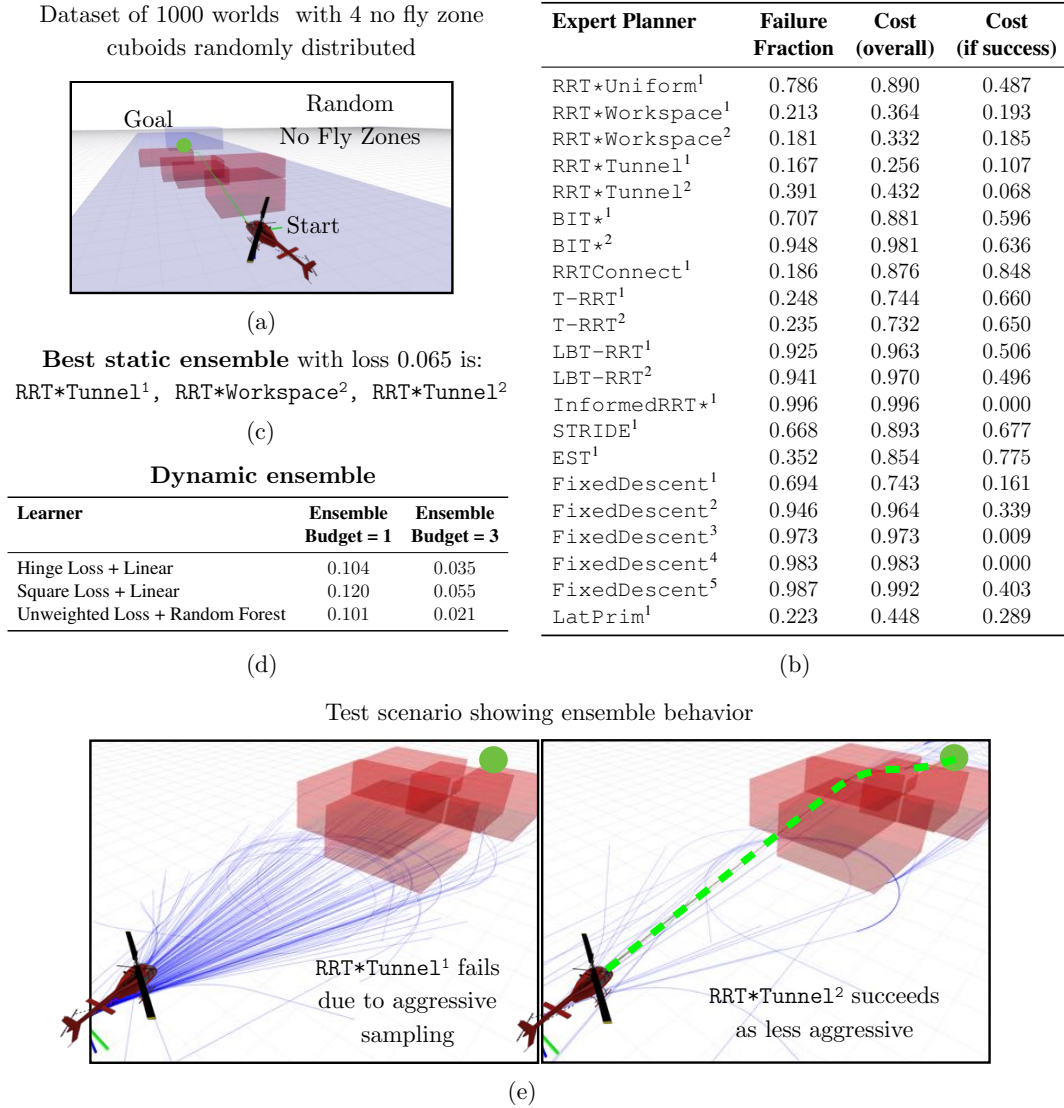


Figure 31: Evaluation of dynamic ensemble selection on simulated datasets (a) The dataset is collected by randomly sampling cuboidal no-fly-zones (b) The 21 planners used in the experiment (refer to Appendix B) for details. The statistics of the planner on the dataset are fraction of times they fail, their average cost, and the average conditional cost on success. (c) Performance of the static ensemble on validation set (d) Performance of different models for learning an ensemble with 1 and 3 elements (e) A scenario where the first predictor selects $RRT^*Tunnel^1$ which fails to solve the problem due to sampling too aggressively. The second predictor selects $RRT^*Tunnel^2$ which is able to solve the problem.

9.3.3 Trajectory planning algorithm - characterization of the chosen ensemble

Objective. We evaluate the ensemble $\{RRT^*Tunnel^1, CHOMP^1\}$ in a full mission simulation in two different environments. The objective is to quantify the usage of the two planners.

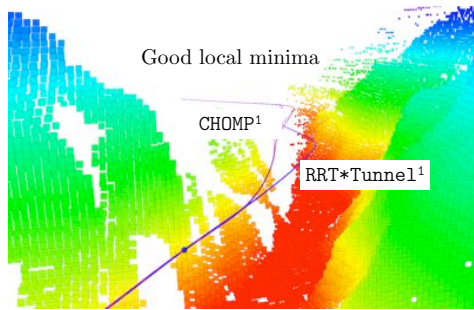
Setup. Fig. 32 shows a simulated mission in the Grand Canyon flying through the canyons. We also execute a simulated mission in San Diego as shown in Fig. 33 flying through unmapped buildings.



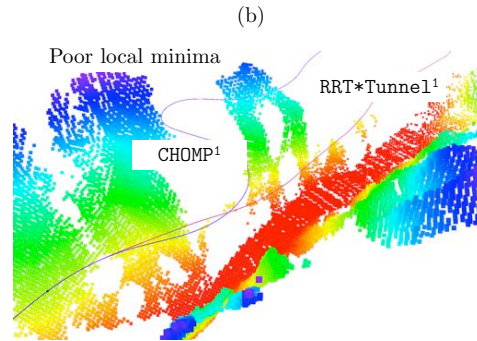
(a)

Statistics of which planner solution is selected

Planner	Selection %
RRT*Tunnel ¹	34.18%
CHOMP ¹	65.82%



(c)



(b)

(d)

Figure 32: Evaluation of the ensemble $\{RRT^*Tunnel^1, CHOMP^1\}$ on a simulated mission in a map of the Grand Canyon, AZ. (a) Illustration of the mission (b) Percentage of times each planner in the ensemble is used during the mission (c) Example scenario where $CHOMP^1$ finds a good solution due to presence of a good local minima. $RRT^*Tunnel^1$ has artifacts due to random sampling (d) $CHOMP^1$ gets trapped in a bad local minima while $RRT^*Tunnel^1$ finds a good solution.

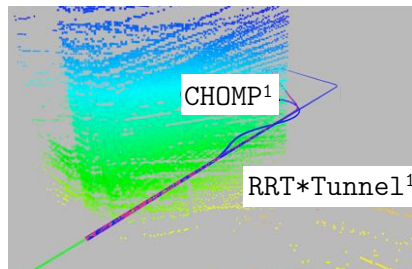


(a)

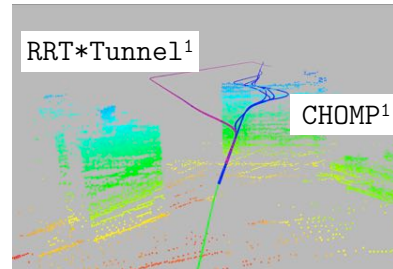
Statistics of which planner solution is selected

Planner	Selection %
RRT*Tunnel ¹	70.83%
CHOMP ¹	29.17%

(b)



(c)



(d)

Figure 33: Evaluation of the ensemble $\{RRT^*Tunnel^1, CHOMP^1\}$ on a simulated mission in a map of San Diego, CA. (a) Schematic of a mission where the city is unmapped and the mission passes through buildings (b) Statistics of the percentage of times each planner in the ensemble is used during the mission (c) Example scenario where $CHOMP^1$ finds a good local minima while $RRT^*Tunnel^1$ computes path that take a longer detour (d) $CHOMP^1$ gets trapped in a local minima inside the building. $RRT^*Tunnel^1$ plans around it.

Analysis. In the canyon, CHOMP^1 is able to find this solution most of the time as shown in Fig. 32(c). This explains the statistic in Fig. 32(b) where we see CHOMP^1 is used 65.82% of the time. On the other hand, the undulating nature of the canyon results in occlusions that prevent obstacles from being mapped. This results in holes in the map as shown in Fig. 32(d). Such situations trap CHOMP^1 in bad local minima. $\text{RRT}^*\text{Tunnel}^1$ on the other hand is a global search technique and is able to find a solution.

O 3. *The relative usage of $\text{RRT}^*\text{Tunnel}^1$ over CHOMP^1 reverses from Grand Canyon mission to San Deigo mission.*

In the San Diego mission, CHOMP^1 is unable to always find a good path as shown in Fig. 33(b). There are some situations where CHOMP^1 finds good solutions due to presence of good local minima as shown in Fig. 33(c). On the other hand, there are many more situations where it gets trapped. As shown in Fig. 33(d), $\text{RRT}^*\text{Tunnel}^1$ is able to find an acceptable path.

9.3.4 Trajectory planning algorithm - torque constraints satisfaction

Objective. We qualitatively evaluate the performance of $\text{RRT}^*\text{Tunnel}^1$ satisfying torque constraints by examining the change in trajectories as the parameters of the constraint vary.

Setup. We focus on the torque constraints during high speed flight. Torque consumption is affected by the airspeed of the helicopter, the speed at which it is climbing and the cargo weight. Hence at low wind speed and light cargo weight, the helicopter can climb at much higher speed and hence has a large glide slope constraint. At high wind speed and large cargo weight, the helicopter can not climb as fast and has a limited glide slope. Fig. 34 shows two different simulated scenarios where the helicopter is flying in Mesa, AZ under the different aforementioned conditions.

Analysis. In scenario 1, the glide slope constraint is much more relaxed. $\text{RRT}^*\text{Tunnel}^1$ plans to climb above the mountains. Fig. 34(c) shows the vehicle climbs at speeds of up to 5 m/s while the maximum torque is far below the limit of 0.8. In scenario 2, the glide slope constraint is stricter. $\text{RRT}^*\text{Tunnel}^1$ plans to fly around the mountains. Fig. 34(e) shows the maximum climb speed reached is ~ 3 m/s. At the peak climb speeds, the torque touches the 0.8 constraint. Hence we can see the planner utilizes the full torque budget to plan.

O 4. *Torque constraints can severely restrict the reachability of the vehicle. A global search technique such as $\text{RRT}^*\text{Tunnel}^1$ is crucial in search for paths under such constraints.*

9.3.5 Trajectory planning algorithm - autorotation constraint satisfaction

Objective. We qualitatively compare CHOMP^1 and CHOMP^2 for respecting autorotation constraints during the descent phase.

Setup. Fig. 35 shows the comparison on a simulated tree avoidance problem. The system is commanded to touch-down near a line of trees. Given the wind speed and the maximum speed during approach, according to the calculations in Section 7.6.2, the glide slope constraint is 8° .

Analysis. As CHOMP^1 is unconstrained, in order to avoid the trees by a sufficient clearance, ends up violating the constraints significantly (12.8°). CHOMP^2 is able to solve this problem by enforcing a trajectory wide constraint of 8° . As can be seen in Fig. 35, this results in a much earlier descent. This is because the gradients from the inequality constraint pushes the obstacle gradients to further back in the trajectory. Note though that CHOMP^2 is not always effective because it applies this constraint trajectory wide which is conservative.

9.3.6 Trajectory planning algorithm - LTE constraint satisfaction

Objective. We qualitatively evaluate the end game library on respecting the loss of tail rotor effectiveness constraint while coming in to land

Setup. We plan a mission where the wind is in the same direction as the approach to landing direction, i.e. the helicopter has tail wind on descent. Unlike having a headwind, where low airspeed keeps the system in a region where LTE is respected, tail wind is quite dangerous during landing. Hence, the planner has to plan to land into the wind without violating the performance charts.

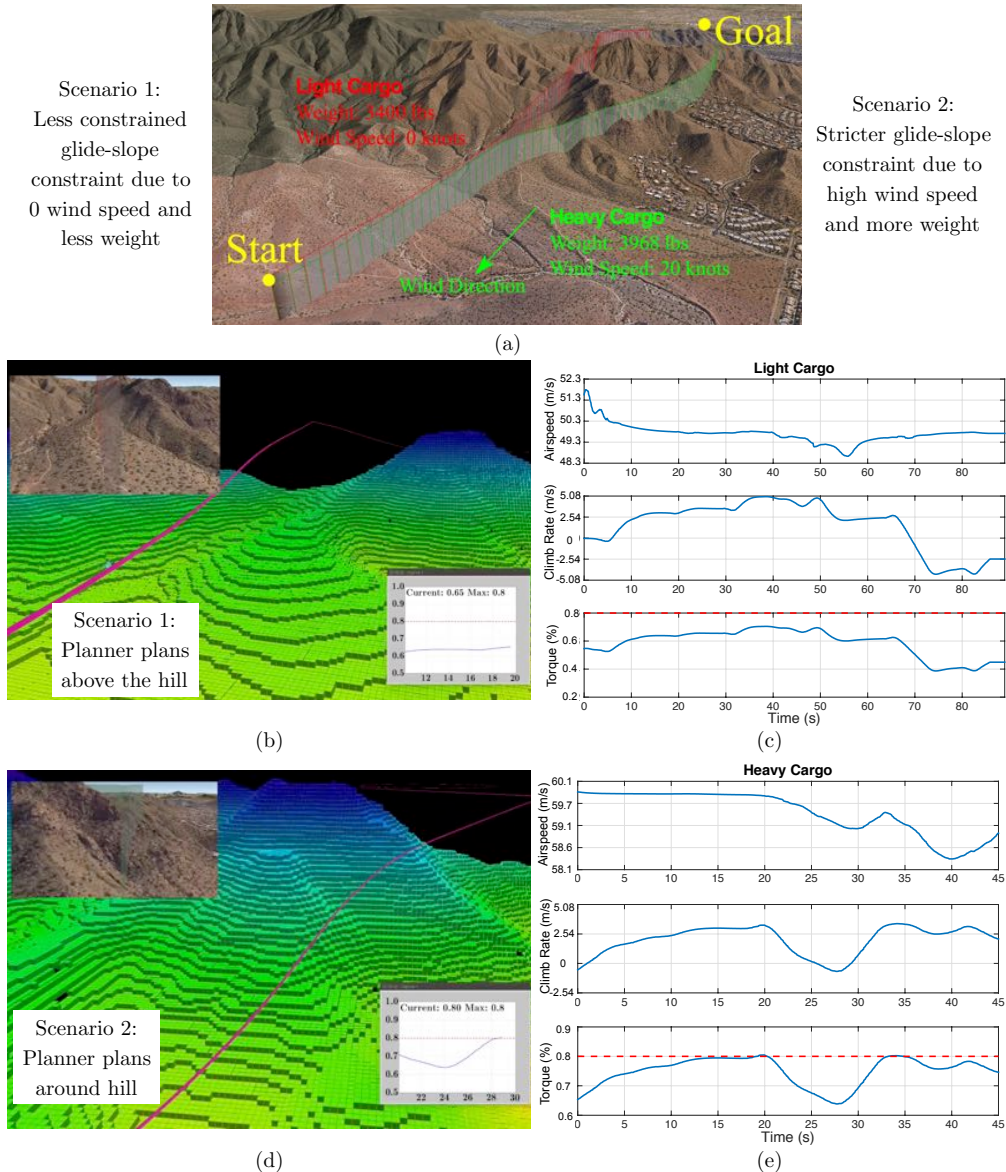


Figure 34: Simulated missions illustrating effect of torque constraints on obstacle avoidance (a) Two different environment conditions resulting in varying torque constraints. (b) Robot path in scenario 1 climbs above the mountain (c) Plots of airspeed, climb rate and torque consumed in scenario 1. Note that torque is below 0.8 limit (d) Robot path in scenario 2 goes around the mountain (e) Plots of airspeed, climb rate and torque consumed in scenario 2. Note that the constraint is active as the robot plans around the mountains.

Analysis. Fig. 36 shows a simulated experiment of such a scenario. As discussed in Section 7.6.3, the planner uses a library of landing primitives to deal with such a situation. It selects a maneuver that turns the aircraft by 180° as shown in Fig. 36(a). The turn has to be such that for a given airspeed, the relative heading is not in an invalid region as shown in Fig. 36(b).

9.3.7 Route optimizer - Dealing with wind

Objective. We evaluate the trajectory generated by the route optimizer in the presence of strong winds.

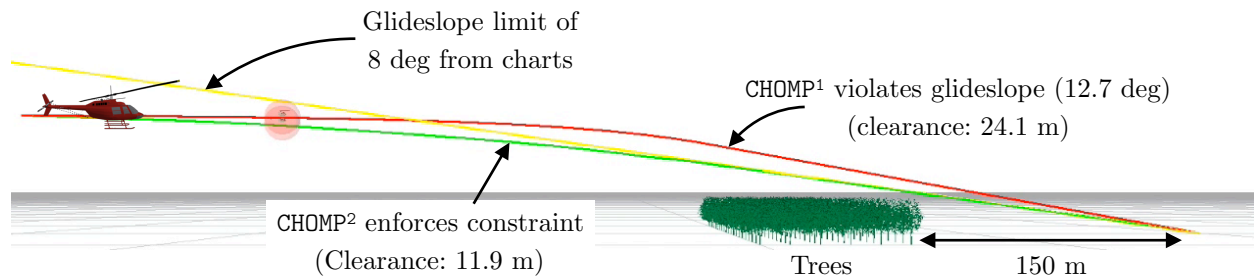


Figure 35: Simulated scenario where CHOMP^1 violates glide slope constraint but CHOMP^2 is able to solve. The scenario is created as a controlled experiment to mimic the situation encountered in Fig. 41(c). The maximum glide slope computed from the performance charts is 8 deg. CHOMP^1 , in order to avoid the trees by 24.1 m, ends up violating the glide slope (12.8 deg). CHOMP^2 on the other hand enforces a trajectory wide constraint and is able to solve the problem.

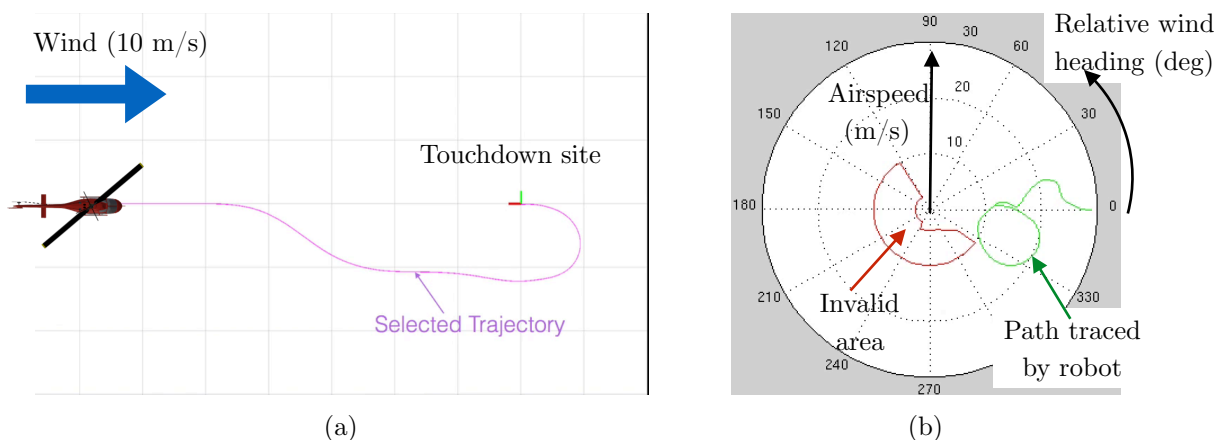


Figure 36: Planning trajectories to deal with loss of tail rotor effectiveness (LTE) (a) The planner selects a trajectory to do a ‘button-hook’ maneuver to land into the wind (b) The LTE constraint and the motion of the helicopter with avoids the invalid region.

Setup. We plan a mission where the helicopter has to fly a loop in the presence of a 20m/s wind directed along the $+ve$ X-axis as shown in Fig. 37. The loop results in the optimizer having to deal with both tail and head wind which have opposing effects.

Analysis. As Fig. 37(a) shows, a feedback controller that attempts to follow the trajectory computed without taking wind into account (red trajectory in Fig. 37(a)) would have to exceed roll and roll-rate limits at the same airspeed. Wind-cognizant κ_{ITE} , on the other hand, generates a trajectory that is dynamically feasible in this wind regime. One can see how κ_{ITE} makes use of wind by generating sharper-looking turns into the wind direction. Similarly, in Fig. 37(b) we compare κ_{ITE} to a naive baseline that uses constant-curvature arcs to turn, while trying to maintain as high a speed as possible without violating the roll limit. The naive baseline must slow down considerably to execute dynamically feasible turns in this wind regime, while κ_{ITE} is able to maintain high speeds.

O 5. Wind can both increase and decrease the reachability of the system - the optimization must be able to exploit this.

9.4 Real flight tests: Evaluation of components

The following set of experiments are based on real flight tests - both fly-by-wire and pilot-in-loop.

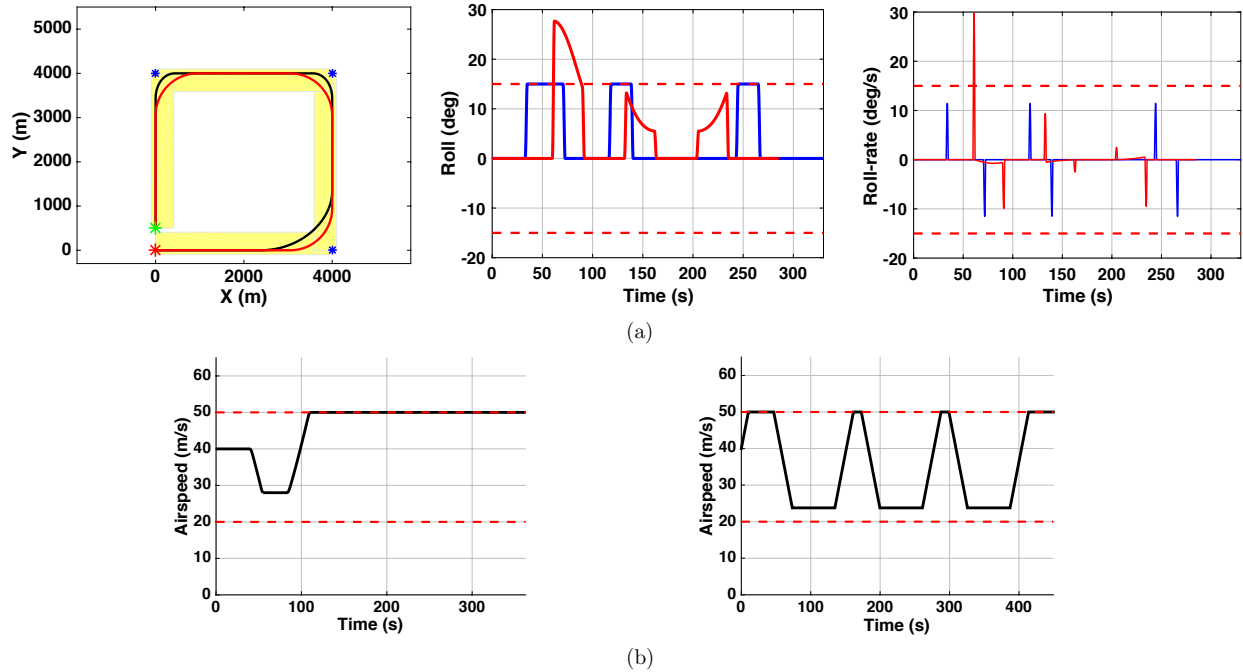


Figure 37: Demonstrating the importance of wind-cognizance in the trajectory planning stage. (a) compares the spatial profiles of wind-aware κ ITE (black) with a wind-agnostic variant of κ ITE (red) in the presence of a $20m/s$ wind along the x -axis (left). A feedback controller used to follow both trajectories in this wind violates roll and roll-rate limits with the wind-agnostic trajectory (right, shown in red), while wind-aware κ ITE (blue) produces a feasible trajectory. (b) shows how the naive baseline (right) has to slow down to execute feasible turns in this wind regime, while κ ITE (left) is still able to maintain high speeds.

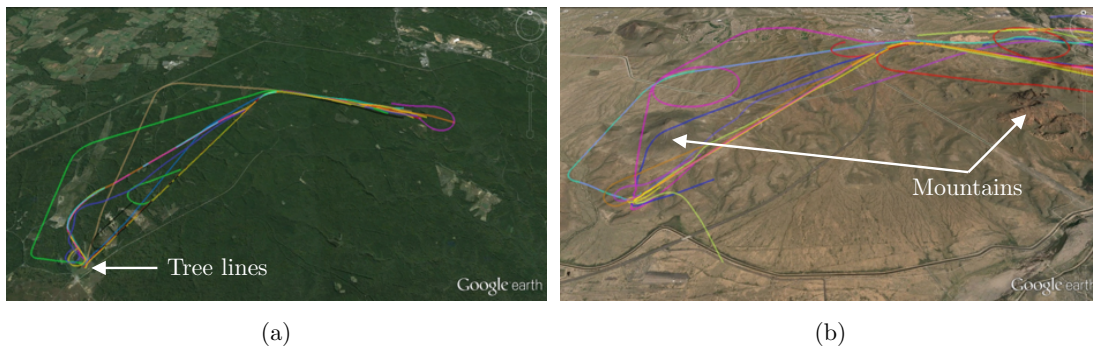


Figure 38: Two sets of real flight tests conducted in different environments. The lines denote the path traced by the helicopter (a) Missions in Quantico, VA, where obstacles are towers, trees and no fly zones (b) Missions in Mesa, AZ, where obstacles are mountains and no fly zones.

9.4.1 Trajectory planning algorithm: Ensemble performance in mountainous terrain

Objective. We qualitatively evaluate the performance of the learned static ensemble of planners in a mission conducted in a mountainous terrain.

Setup. Offline, we created a dataset of planning problems by sampling start and goal locations in maps with mountains and NFZs. This was used to compute a static ensemble $\{RRT^*Tunnel^1, CHOMP^1\}$ used on real flight tests.

We performed flight tests using fly-by-wire on the ULB in Mesa, AZ where the helicopter was commanded to fly

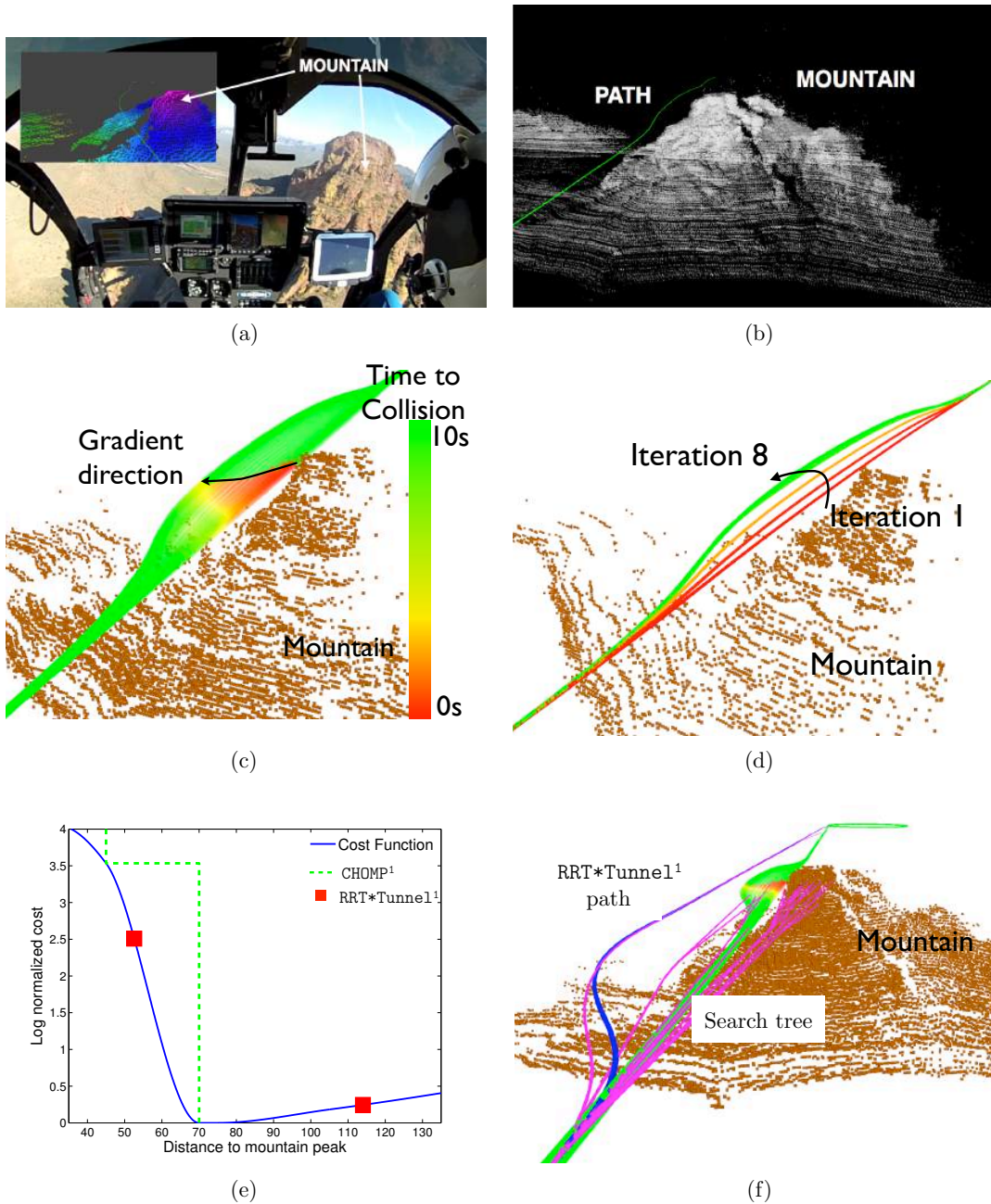


Figure 39: Ensemble $\{ \text{RRT}^*\text{Tunnel}^1, \text{CHOMP}^1 \}$ performance in mountainous terrain - datapoint where CHOMP^1 finds a good solution. (a) Flying around an unmapped mountain in Mesa, AZ (safety pilot's view) (b) The environment mapped by the robot and trace of path (c) The gradient due to time to collision objective (d) CHOMP^1 converges to a good local minimum (e) The log normalized cost function valley with distance from the mountain peak. The optimizer reaches the crest. The $\text{RRT}^*\text{Tunnel}^1$ computes solutions lying on either side of the minimum. (f) The $\text{RRT}^*\text{Tunnel}^1$ optimal path avoids the mountain by a much larger distance than required. The search tree contains branches on either side of the cost valley

routes that intersected with mountains and NFZs at 30 m/s as shown in Fig. 38(a).

Analysis. All flight tests were successful, i.e. the trajectory planner successfully navigated around the mountain. We highlight two datapoints.

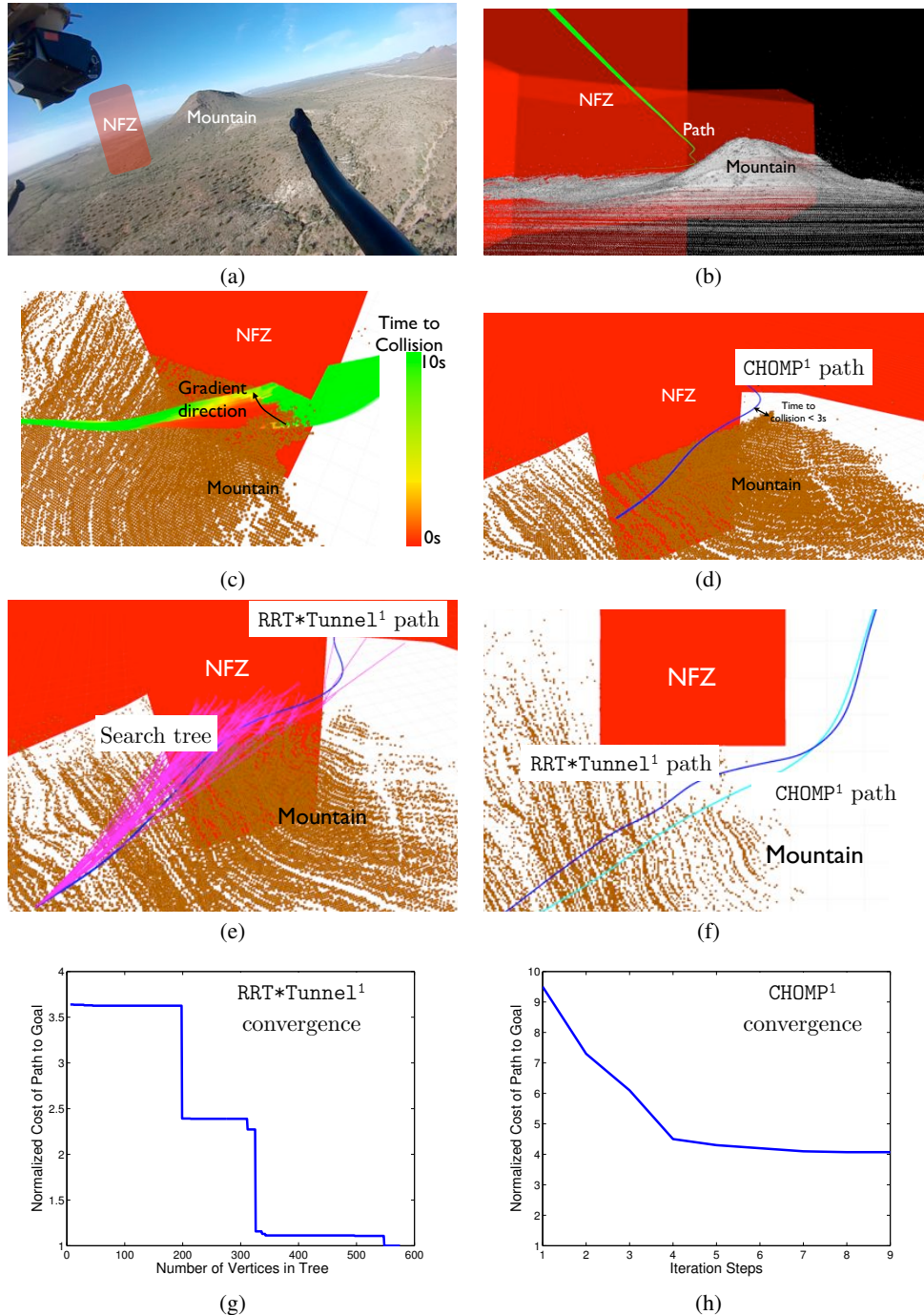


Figure 40: Ensemble $\{ \text{RRT}^*\text{Tunnel}^1, \text{CHOMP}^1 \}$ performance in mountainous terrain - datapoint where $\text{RRT}^*\text{Tunnel}^1$ finds a good solution. (a) Flying between a NFZ and an unmapped mountain in Mesa, AZ (The skid camera view) (b) The mapped environment and traced path (c) The gradient due to the time to collision points into the no-fly-zone (d) CHOMP^1 gets stuck in a bad local minimum and has a critically low time to collision (e) The $\text{RRT}^*\text{Tunnel}^1$ is very diverse and contorts to find a near optimal trajectory (f) Comparison of the $\text{RRT}^*\text{Tunnel}^1$ trajectory to CHOMP^1 shows that $\text{RRT}^*\text{Tunnel}^1$ is safer (g) The best path in the $\text{RRT}^*\text{Tunnel}^1$ tree converges near optimal after sampling around 320 vertices. (h) CHOMP^1 cannot lower cost below a certain limit because perturbations violate no-fly-zone constraint.

The first datapoint is shown in Fig. 39 where the helicopter autonomously plans around an unmapped mountain while flying at 30 m/s. In this scenario the helicopter was following a mission to a waypoint behind the mountain. The obstacle is detected and a trajectory is planned such that the minimum time to collision is 5.0s. Fig. 39 (c) shows the time to collision heat map and resultant gradient. Fig. 39 (d) shows the iterations of CHOMP^1 which converges quickly to the local minimum. Fig. 39 (e) shows CHOMP^1 reducing the cost with iterations, while $\text{RRT}^*\text{Tunnel}^1$ lies on either side of the cost valley. The jump to the minimum by the optimizer indicates the rapid effect of covariant descent. The $\text{RRT}^*\text{Tunnel}^1$, in this case, cannot compete with the optimizer as shown in Fig. 39(f). The tree doesn't get a sample in the valley of the cost function and hence produces a path takes a more extreme diversion and is no longer optimal. We summarize it in the following observation:

O 6. *Gradient based planners like CHOMP^1 are essential in quickly locating the minima of the cost functional.*

The second datapoint is shown in Fig. 40 where the planner has to navigate between a known NFZ and an unmapped mountain at 30 m/s in Mesa, AZ. In this scenario the helicopter has going to land at a point behind the NFZ. This scenario leads to a failure case of sorts for CHOMP^1 . Since CHOMP^1 is an unconstrained descent, it deals with no-fly-zones by backtracking every time there is a violation. As a result, after 4 iterations, CHOMP^1 gets stuck in a bad local minimum. The computed path has a critically low time to collision of less than 3s. $\text{RRT}^*\text{Tunnel}^1$ on the other hand samples enough vertices to pass safely through the gap. As seen from Fig. 40 (g), the cost has distinct stepwise decrements. This is indicative of the search eventually able to find a good edge through the gap and subsequently repairing the search tree. It finally finds a good enough solution that is safer than that of CHOMP^1 . We summarize it in the following observation:

O 7. *A global search technique such as $\text{RRT}^*\text{Tunnel}^1$ is critical in environments which may have potentially bad local minima or where multiple homotopies are present.*

9.4.2 Trajectory planning algorithm: Ensemble performance in constrained landing zones

Objective. We qualitatively evaluate the performance of the learned static ensemble of planners in a mission conducted in a forest terrain with a constrained landing zone.

Setup. Offline, we created a dataset of planning problems by sampling start and goal locations in maps with NFZs and simulated tree lines. This was used to compute a static ensemble $\{\text{CHOMP}^1, \text{CHOMP}^2\}$ used on real flight tests. CHOMP^2 is a planner that is identical to CHOMP^1 except it enforces a glide-slope constraint at all points along the trajectory. Note that CHOMP^2 conservatively chooses a uniform glide slope limit. Hence there are many situations where it does not find a good solution which CHOMP^1 does.

We performed flight tests using fly-by-wire on the ULB in Quantico, VA and Manassas, WV where the helicopter was commanded to fly routes that required avoiding a line of trees as shown in Fig. 38(b).

Analysis. The flight tests were successful in planning above tree lines. Fig. 41 shows a set of scenarios that the system encountered. While CHOMP^1 solves most problems, there are situations where it is unable to compute a feasible solution due to violation of glide-slope constraints. One such scenario is shown in Fig. 41(c) where the helicopter must fly above a tall tree line and immediately descend to land. For this situation, enforcing a strict glide slope limit is crucial. CHOMP^2 is able to solve such problems. Another such situations is where an obstacle is detected late as shown in Fig. 41(e), forcing CHOMP^1 to climb steeply to avoid it. Again, CHOMP^2 is able to find acceptable solution to such problems.

O 8. *Even though CHOMP^2 applies a trajectory wide conservative glide slope constraint, it plays a critical role in problems requiring steep ascent / descent.*

9.4.3 Trajectory planning algorithm: Optimizing time to collision objective

Objective. We evaluate the avoidance behavior generated by executing CHOMP^1 optimizing time to collision.

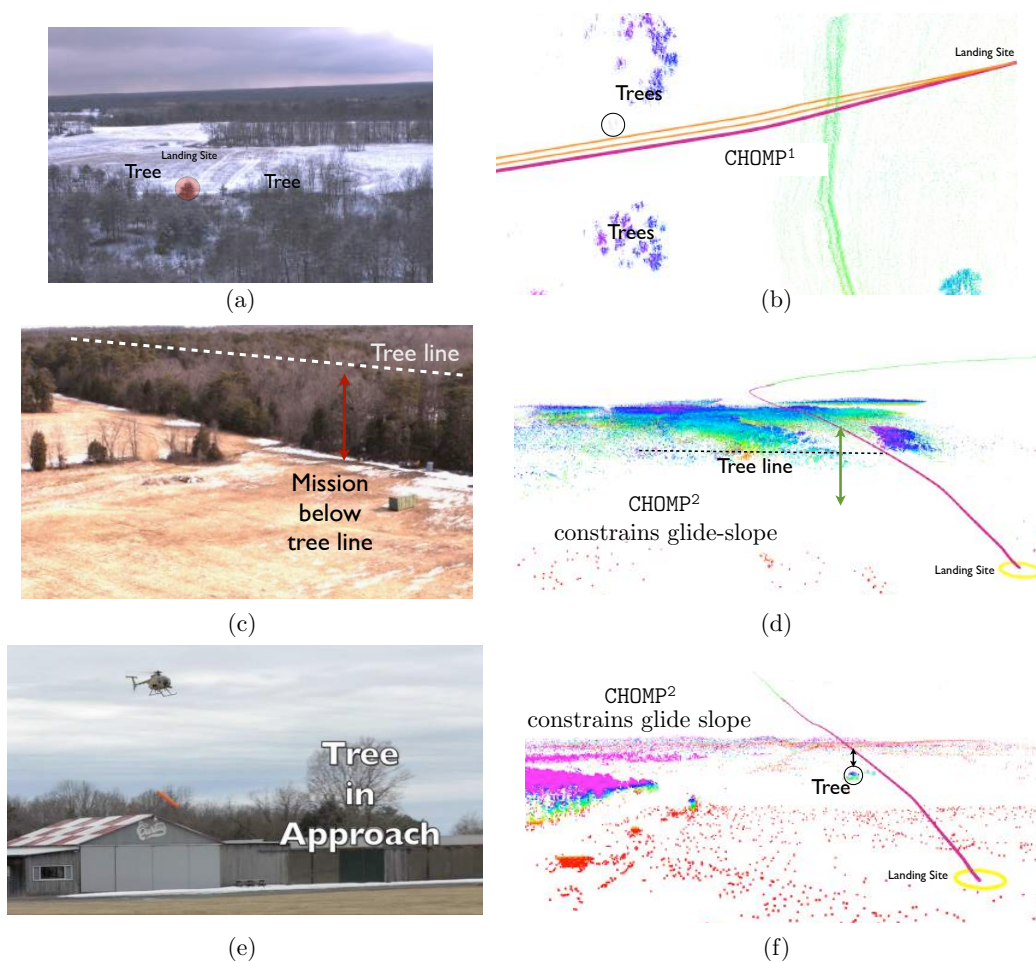


Figure 41: Ensemble $\{ \text{CHOMP}^1, \text{CHOMP}^2 \}$ performance in forest terrain - datapoint where CHOMP^2 finds a good solution. (a) Perspective from the helicopter where the robot must avoid a tree while it comes down to a landing site (b) The system uses CHOMP^1 to optimize a trajectory to avoid the tree-line (c) A taller tree line that requires the helicopter to descend late (d) CHOMP^2 avoids the tree by optimizing over it but respecting glide slope constraints at all points on the trajectory (e) A tree detected late on approach (f) CHOMP^2 constrains glide slope while avoiding this tree

Setup. These flight tests were performed pilot-in-loop on the Bell-206L in Pittsburgh, PA. Fig. 42 illustrates the mission where a helicopter is flying at high speed towards a tower at 50m/s. The static ensemble in this case was executing $\{ \text{CHOMP}^1, \text{CHOMP}^2 \}$.

Analysis. The statistics of these runs is shown in Fig. 42(a). We can see that depending on the line of approach, the avoidance is either predominantly vertical or horizontal. The time to collision cost influences the clearance distance and the shape of the path at convergence. Fig. 42(d) shows a profile for vertical avoidance. The avoidance commences at a distance of 596.80m and clears the top of the tower by 24.01 m. Fig. 42(e) shows a profile for horizontal avoidance. In this case, the tower is detected much later 393.80 m. The trajectory avoids the tower by 78.94m which is much larger than the clearance for vertical avoidance. According to the safety pilot, this avoidance profile is not only desirable, but it follows naturally from the definition of the time to collision objective.

O 9. Time to collision objective captures the avoidance behavior that pilots use.

Run #	Type of avoidance	Distance of final trajectory from tower (m)	Distance at which avoidance commences (m)
1	Vertical	24.08	596.80
2	Vertical	38.01	589.79
3	Horizontal	78.94	393.80

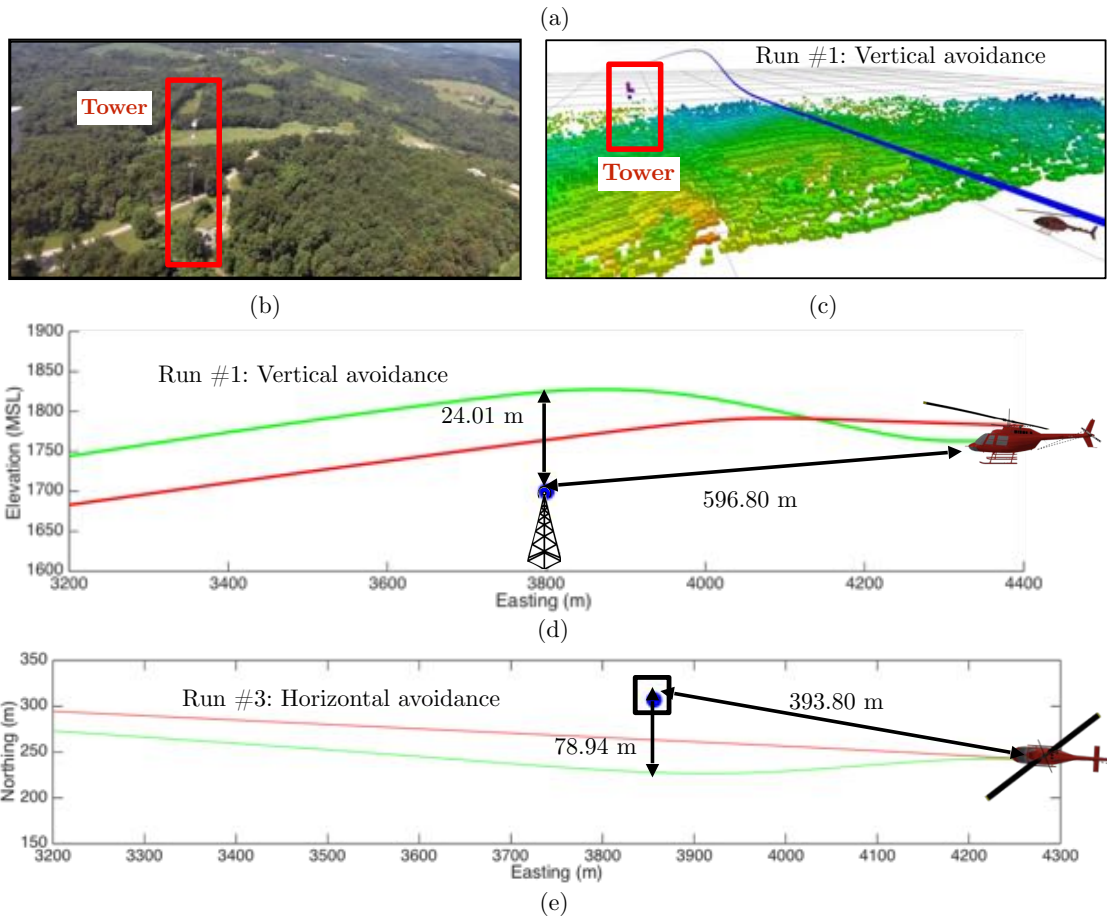


Figure 42: Performance of CHOMP¹ on 3 flight tests where the helicopter has to avoid an unmapped tower. (a) Statistics of the 3 runs where the system avoided the tower both vertically and horizontally (b) The tower as seen from the onboard camera of the sensor (c) Run 1 where the system avoids the tower vertically. The environment as mapped by the robot and the planned path is shown (d) Plot of the path for Run 1. The red line is the commanded path from the mission and the green is the path followed by the system. (e) Plot of the path for Run 3. The avoidance is mainly horizontal in this case.

9.4.4 Trajectory planning algorithm: Satisfying torque constraints from performance charts

Objective. We qualitatively evaluate the performance of RRT*_{Tunnel}¹ satisfying torque constraints and the change in trajectories as environmental variables vary.

Setup. Two different flight tests were performed pilot-in-loop on the Bell-206L as shown in Fig. 43 (a). In each of these flight tests, the helicopter had to avoid a no-fly-zone. For one of the tests, the system was carrying high cargo weight in strong winds. For the other mission, the system was carrying light cargo in negligible winds.

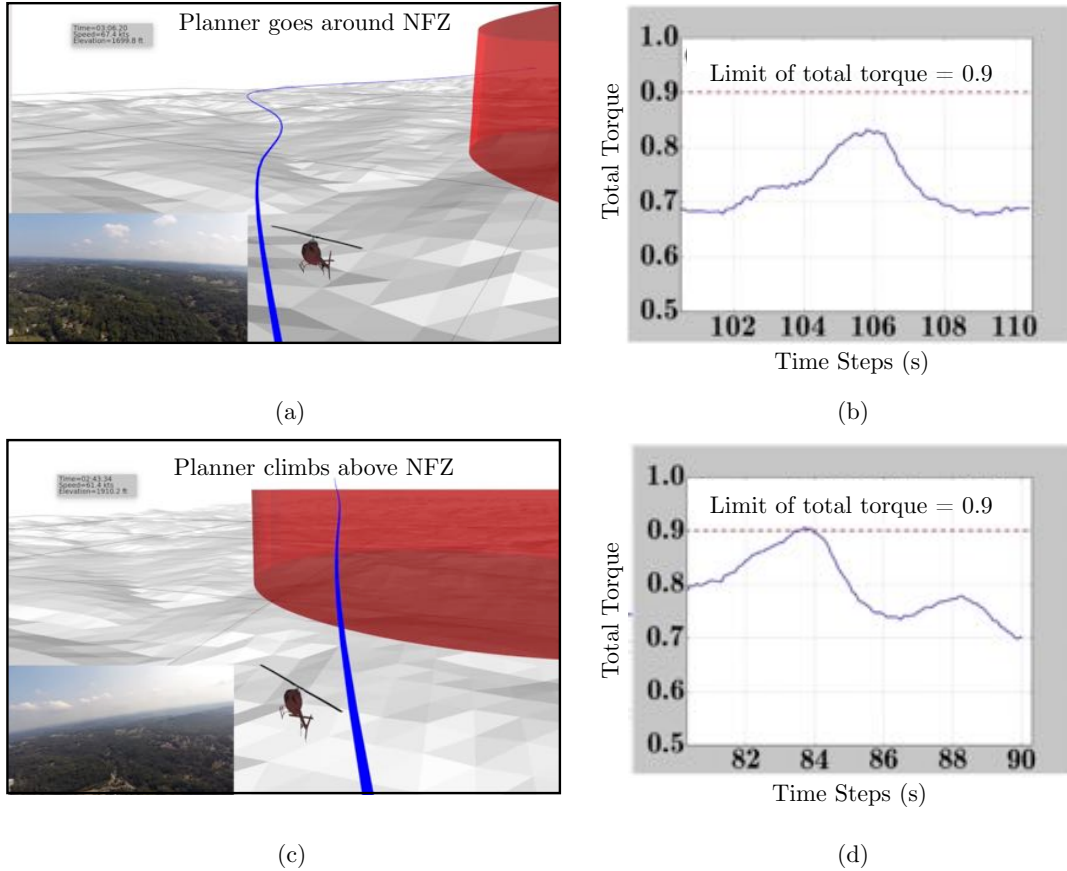


Figure 43: Performance of $RRT^*Tunnel^1$ respecting torque constraints under varying conditions of wind and cargo weight. (a) In flight test 1, the wind speed is high and the robot weight is large resulting in increased torque consumption. $RRT^*Tunnel^1$ plans around the no-fly-zone. (b) Plot of torque consumption. Note that constraint is satisfied. (c) In flight test 2, the wind speed is 0 and the robot weight is small resulting in a small torque consumption. $RRT^*Tunnel^1$ plans above the no-fly-zone. (d) Plots of torque consumption. Note that the constraint is active as the robot climbs above the no-fly-zone.

Analysis. Fig. 43 (a) shows the scenario where the wind is high and the cargo is heavy. Under such situations, the torque constraint limits the glide-slope forcing the planner to plan around the NFZ. Fig. 43 (b) shows the scenario where there is less wind and the cargo is light, which implies the system can climb above the NFZ. Hence, the global search nature of $RRT^*Tunnel^1$ is able to find these solutions that in fact lie in different homotopies.

O 10. *Changes in wind and environmental conditions can significantly affect the type of motions the system can execute. In presence of obstacles, this necessitates the use of global search techniques such as $RRT^*Tunnel^1$.*

9.4.5 Trajectory planning algorithm: Satisfying height-velocity constraint from performance charts

Objective. We qualitatively evaluate the behavior of the take-off when satisfying height-velocity constraints.

Setup. One can show that the only situation where the height-velocity constraint is in danger of being violated is during takeoff. The height-velocity curve constraint gets violated when the helicopter is at a significant height above the ground at low speeds. This typically happens when the take-off is steep.

We performed a flight-test that focusses on repeated take-offs performed pilot-in-loop on the Bell-206L. We compared a trajectory library that simply satisfied the torque constraints versus a library that satisfied the height-velocity constraints.

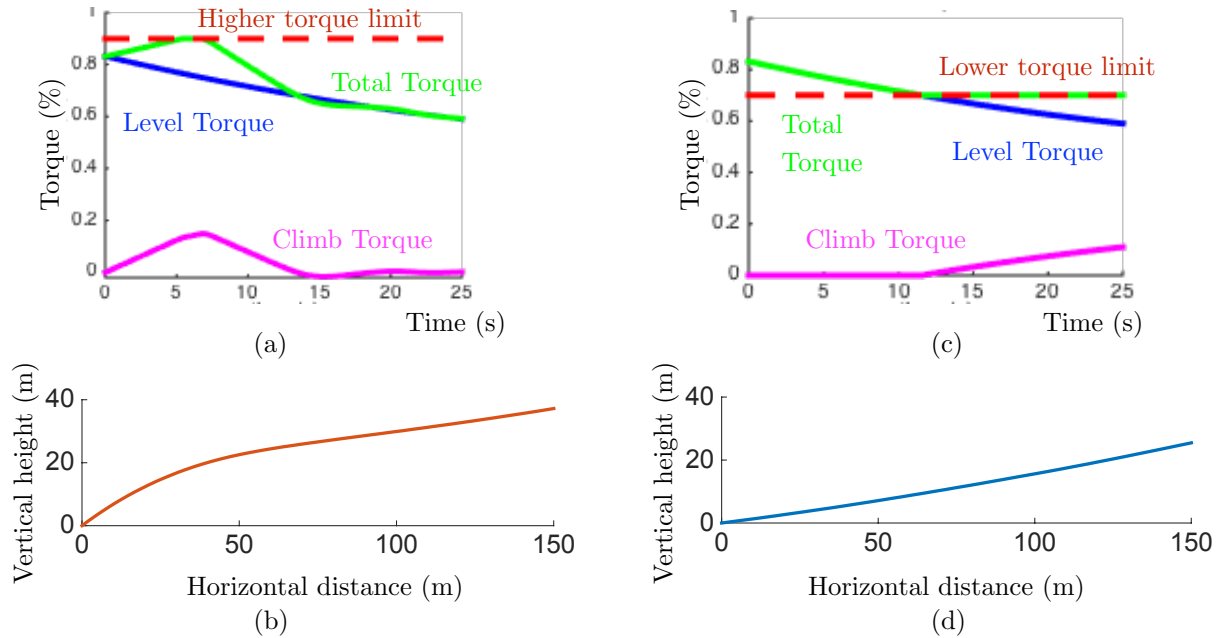


Figure 44: Difference in take-off profiles on enforcing height-velocity constraints (a) When only satisfying torque limit of 0.9, the helicopter begins to climb instantly while increasing airspeed to control the total torque (b) This leads to a steeper climb during takeoff that violates the height velocity curve (c) Torque profile on enforcing the height velocity constraint. (d) This leads to a shallower climb.

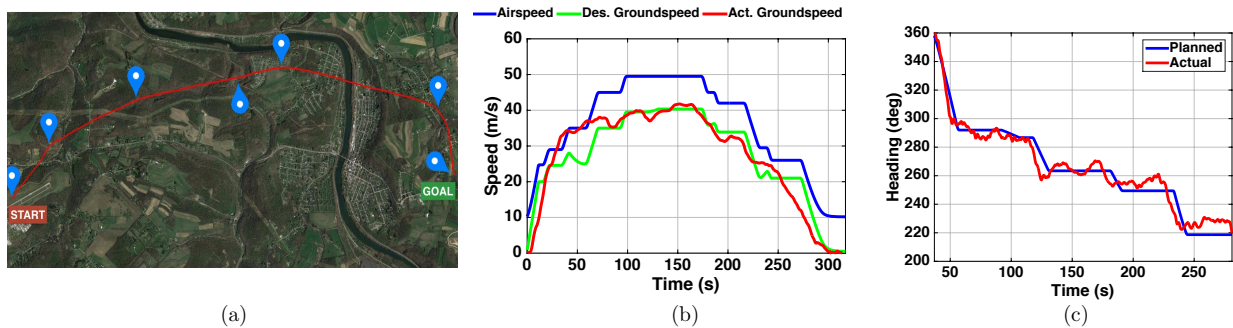


Figure 45: Results from a test conducted on a full-size helicopter with a ~ 10 m/s wind from 250° . (a) shows the complete trajectory along with the waypoints; (b) shows the tracking performance for ground speed and (c) ground-frame heading. This trajectory was computed in real-time onboard the helicopter.

Analysis. The preliminary version of the take-off library computed maneuvers by selecting from trajectories that satisfied the torque limit of 0.9 as shown in Fig. 44(a)(b). This led to a steep enough climb to violate the height velocity constraints.

On enforcing the constraints, we arrive at a simple policy that increases forward speed enough till the torque consumed drops below 0.7. At that point the system can climb while maintaining the total torque to be 0.7 as shown in Fig. 44(c)(d).

9.4.6 Route optimizer - Dealing with wind

Objective. We evaluate the trajectory generated by the route optimizer in the presence of strong winds.

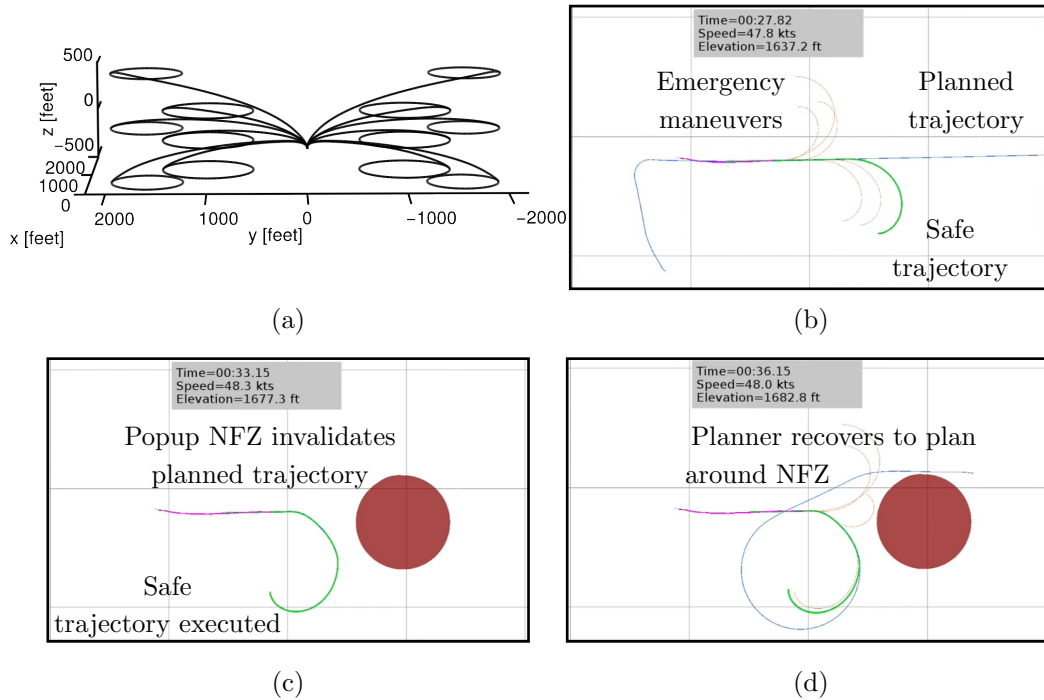


Figure 46: Guaranteeing safety when trajectory planning fails (a) The evasive maneuver library (b) The planner plans a nominal trajectory which is checked to be safe (c) A no-fly-zone appears, and the safety system rejects the old plan. System eventually executes the evasive maneuver (d) Planner recovers and nominal behaviour resumes.

Setup. This mission was performed pilot-in-loop on the Bell-206L. In this mission, a trajectory has to be planned in the presence of a $\sim 10\text{m/s}$ wind blowing from 250° (measured with the onboard pitot tube).

Analysis. Fig. 45 shows the system was able to track the commanded trajectory while remaining within its control margins at all times.

9.4.7 Guaranteeing Safety: Evasive maneuver when trajectory planning fails

Objective. We evaluate the ability of the system to remain safe (in a known free state) when the trajectory planning fails.

Setup. This mission was performed pilot-in-loop on the Bell-206L. In this mission, a popup no-fly-zone appears very near the vehicle. Fig. 46 shows the scenario.

Analysis. Fig. 46 (c) shows that the planner is unable to compute a path. As a result the system eventually executes the evasive maneuver part of the current trajectory. Fig. 46 (d) shows that the evasive maneuver moves the system to a state from where the planner can find a solution. The system then resumes nominal flight.

9.4.8 Guaranteeing Safety: Evasive maneuver when sensor fails

Objective. We evaluate the ability of the system to remain safe (in a known free state) when the sensor fails.

Setup. This mission was performed pilot-in-loop on the Bell-206L. In this mission, the sensor was disconnected mid-flight. Fig. 47 shows the scenario where the unknown space remains unchanged.

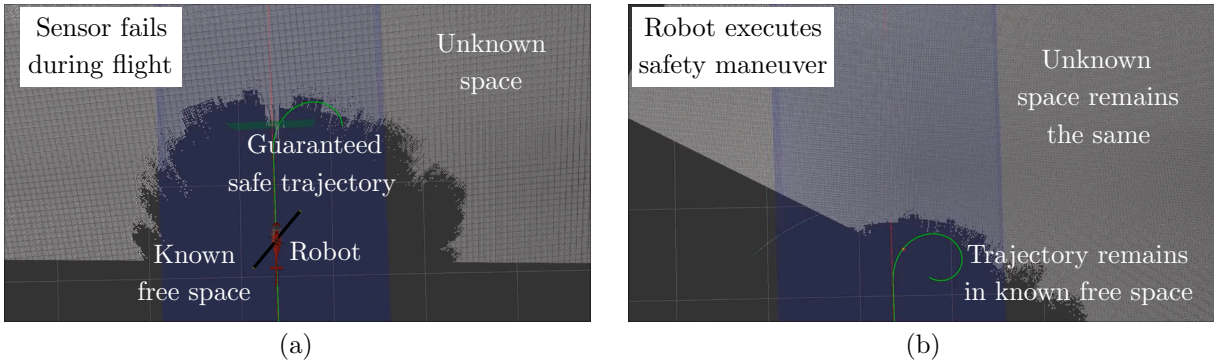


Figure 47: Guaranteeing safety even when the sensor fails (a) A situation where the sensor is disconnected. The current trajectory being followed is guaranteed safe by construction. (b) The robot eventually executes the evasive maneuver.

Analysis. Fig. 47(a) shows that the current trajectory being followed is guaranteed safe as it lies in the known free space. In the subsequent timesteps, even though the trajectory planning algorithm continues to plan new paths, the trajectory executive does not update the trajectory as the unknown space remains unchanged. The robot continues to follow this trajectory and eventually executes the evasive maneuver section as shown in Fig. 47(b). Hence, the robot perpetually remains in the known free space ensuring the system remains safe till a human operator can intervene.

10 Discussions

This paper describes a first comprehensive approach to motion planning for autonomous helicopters. True autonomous flight of full-scale systems needs to respect a large set of constraints typically ignored in the literature, because incorporating these constraints increases the planning dimension and time beyond real-time limits. We presented several ideas and an architecture on how to effectively build a scalable planning system. While the idea of decoupling planning into multiple layers is common, we present several contributions on how to maintain high-quality solution and safety in real-time, incorporate wind, solve more difficult problems via an ensemble of planners approach. Unique to this work is the extensive evaluation of the approach in more than 700 hours of flight tests and the corresponding analysis over several years.

While there is some prior work that described motion planning approaches for autonomous flight none has had the opportunity to go in as much depth in this problem. As can be seen in the result the nominal case of flying without wind and decoupled dynamics was achieved quickly in the first phase as long as the systems stays within the “envelope” (e.g. constraints). So far the state-of-the-art has demonstrated prior motion planning results in this realm. However, as the helicopter gets closer to the boundary of the envelope in power, wind, and other limits the complexity of the motion planning problem increases significantly as can be seen in the amount of testing required in phase 2 and 3 to validate the approach.

Our architecture allowed us to scale the number of constraints considered in the motion planning problem addressed while maintaining safety by using the ensemble and safety executive approach. We also demonstrated that a single static motion planning algorithm is not able to solve the problem in real-time and which algorithms are successful depends on the environment and speed-regime of the system. In particular for our problem the range of obstacle sizes ranges from thin wires to large buildings and NFZs, and speeds range from 0 to 60 m/s. The ensemble approach was able to significantly improve the robustness of the system.

To summarize this type of decomposition and interface has advantages and disadvantages:

The advantages of our approach are a

- *Time and planning horizon decoupling:* Each module plans over a different horizon and has a correspondingly shorter or longer reaction time, while maintaining safety at the fastest reaction time.

- *Ensemble of planners*: A particular planner can be designed to handle a custom set of problems, constraints etc. The ensemble selection can then choose a planner based on the distribution of problems encountered by the system.
- *Time-to-collision objective*: This objective captured pilot like avoidance maneuvers which was otherwise hard to reproduce via standard objectives like proximity to obstacles.
- *Decoupling of safety from performance related components*: The safety system is simple and its guarantees are decoupled from the performance and correctness of the nominal algorithms.

While in practice our approach has shown to be useful there are some disadvantages to our approach:

- *Potential deadlocks*: While we have tried our best to mitigate deadlocks, they do arise when faced with complex obstacles or when the safety system deems the vehicle to be in an unsafe state. Future work should focus on recovery strategies or relax conservative safety assumptions.
- *Downsides of approximation techniques*: The choice of the planner using an approximate model to plan has disadvantages if the margins between the required commands to be able to track the trajectory and the available authority of the control system decreases. This means that even if the system can be more aggressive in theory, the current planner cannot express such commands.
- *Fixed planning frequency*: We ask the planners to re-plan at a fixed timestep which restricts the difficulty of the problem that these planners can solve. An alternate technique would be to have a perpetually anytime strategy that quickly returns a solution first for the aircraft to follow and proceeds to improve it over time.

We conclude with a summary of lessons learned in the course of this project:

- *Wind and smooth flight matters*: Our focus initially was primarily on obstacle avoidance. However, the primary point of failure was always due to lack of consideration of wind and smooth pilot like maneuvers early on. These are not simply optimization objectives, but also affect the methods and representation (ground frame vs air frame).
- *Coordinate systems matter*: We found the standard UTM system did not suffice given that we were flying across UTM zones. Latitude / Longitude also are complex to deal with as they lie on a manifold. We ultimately settled on ECEF but this required a significant overhaul of the planning code.
- *Planning system should correct, not reject*: As shown in Fig. 27, the planner system contributes to 23% of failures. This primarily happens because the planner is at the end of the pipeline and at the mercy of outputs arising from all of the other components. Hence it gets incorrect requests which it rejects - as a result the system does not move and the flight test is a failure. Instead, if the planning system were to correct the input as much as possible, as the user for approval and proceed to complete the mission, this number could have been negligible. This is difficult to do and requires investment of resources.
- *Low speed behaviors*: The second biggest cause of failure as shown in Fig. 27 is due to the landing zone detection finding small obstacles that require moving to another touchdown site. This detection is usually very late, causing our system to abort. The remedy is to ensure the system should commit as late as possible to a touchdown site, i.e. able to taxi at low speeds.

Acknowledgments

This work would not have been possible without the dedicated efforts of the entire AACUS TALOS team and was supported by ONR under contract N00014-12-C-0671. This document has been approved for public release (DCN # 43-4164-18). Distribution A: Unlimited.

References

Althoff, D., Kuffner, J. J., Wollherr, D., and Buss, M. (2011). Safety assessment of robot trajectories for navigation in uncertain and dynamic environments. *Springer Autonomous Robots*, SI Motion Safety for Robots.

- Althoff, D., Werling, M., Kaempchen, N., Wollherr, D., and Buss, M. (2012). Lane-based safety assessment of road scenes using Inevitable Collision States. In *Proc. of the IEEE Intelligent Vehicles Symposium*.
- Anderson, E. P., Beard, R. W., and McLain, T. W. (2005). Real-time dynamic trajectory smoothing for unmanned air vehicles. *IEEE Trans. Contr. Sys. Techn.*, 13:471–477.
- Arora, S., Choudhury, S., Althoff, D., and Scherer, S. (2014). A principled approach to enable safe and high performance maneuvers for autonomous rotorcraft. In *American Helicopter Society Forum 70*.
- Arora, S., Choudhury, S., Althoff, D., and Scherer, S. (2015). Emergency maneuver library - ensuring safe navigation in partially known environments. In *IEEE International Conference on Robotics and Automation*.
- Bakolas, E. and Tsiotras, P. (2010). Time-optimal synthesis for the zermelo–markov–dubins problem: the constant wind case. In *American Control Conference*.
- Bakolas, E. and Tsiotras, P. (2013). Optimal synthesis of the zermelo–markov–dubins problem in a constant drift field. *Journal of Optimization Theory and Applications*, 156(2):469–492.
- Bautin, A., Martinez-Gomez, L., and Fraichard, T. (2010). Inevitable collision states: a probabilistic perspective. In *IEEE International Conference on Robotics and Automation*.
- Blachini, F. (1999). Set invariance in control. *Automatica*, 35(11):1747–1767.
- Blum, A. (1998). On-line algorithms in machine learning. In *Online algorithms*, pages 306–325. Springer.
- Bobrow, J. E., Dubowsky, S., and Gibson, J. (1985). Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17.
- Bohlin, R. and Kavraki, L. E. (2000). Path planning using lazy prm. In *IEEE International Conference on Robotics and Automation*.
- Boissonnat, J.-D. and Lazard, S. (1996). A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles. In *Proceedings ACM Symposium on Computational Geometry*, pages 242–251.
- Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.
- Bryson, A. E. and Ho, Y.-C. (1975). *Applied Optimal Control*. Hemisphere Publishing Corp., New York.
- Choset, H. M. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT press.
- Choudhury, S., Arora, S., and Scherer, S. (2014). The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. In *American Helicopter Society Forum 70*.
- Choudhury, S., Arora, S., and Scherer, S. (2015). The Planner Ensemble: Motion planning by executing diverse algorithms. In *IEEE International Conference on Robotics and Automation*.
- Choudhury, S., Gammell, J. D., Barfoot, T. D., Srinivasa, S., and Scherer, S. (2016). Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning. In *IEEE International Conference on Robotics and Automation*.
- Choudhury, S. and Scherer, S. (2015). The dynamics projection filter (DPF) - Real-time nonlinear trajectory optimization using projection operators. In *IEEE International Conference on Robotics and Automation*.
- Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501.
- Dubins, L. E. (1957). On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516.
- Dugar, V., Choudhury, S., and Scherer, S. (2017a). A kite in the wind: Smooth trajectory optimization in a moving reference frame. In *IEEE International Conference on Robotics and Automation*.
- Dugar, V., Choudhury, S., and Scherer, S. (2017b). Smooth trajectory optimization in wind: First results on a full-scale helicopter. In *American Helicopter Society Forum 73*.

- FAA (2012). Helicopter flying handbook. *US Department of Transportation, faah808321 edition*, 91.
- Fabiani, P., Fuertes, V., Piquereau, A., Mampey, R., and Teichteil-Konigsbuch, F. (2007). Autonomous flight and navigation of vtol uavs: From autonomy demonstrations to out-of-sight flights. *Aerospace Science and Technology*, 11(2-3):183–193.
- Fraichard, T. and Asama, H. (2004). Inevitable collision states. a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024.
- Frazzoli, E., Dahleh, M. A., and Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129.
- Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. (2014). Informed RRT*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. (2015). Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation*.
- Hauser, K. (2015). Lazy collision checking in asymptotically-optimal motion planning. In *IEEE International Conference on Robotics and Automation*.
- Heng, L., Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *IEEE International Conference on Robotics and Automation*.
- Howard, T. M. (2009). *Adaptive Model-predictive Motion Planning for Navigation in Complex Environments*. PhD thesis, Pittsburgh, PA, USA.
- Hrabar, S. (2008). 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Hsu, D., Latombe, J.-C., and Motwani, R. (1999). Path planning in expansive configuration spaces. *International Journal Computational Geometry & Applications*, 4:495–512.
- Hsu, D., Sánchez-Ante, G., and Sun, Z. (2005). Hybrid prm sampling with a cost-sensitive adaptive strategy. In *IEEE International Conference on Robotics and Automation*.
- Hwan Jeon, J., Cowlagi, R. V., Peters, S. C., Karaman, S., Frazzoli, E., Tsiotras, P., and Iagnemma, K. (2013). Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *American Control Conference*.
- Hwangbo, M., Kuffner, J., and Kanade, T. (2007). Efficient two-phase 3d motion planning for small fixed-wing uavs. In *IEEE International Conference on Robotics and Automation*.
- Jacobs, P. and Canny, J. (1989). Planning smooth paths for mobile robots. In *IEEE International Conference on Robotics and Automation*.
- Janson, L., Schmerling, E., Clark, A., and Pavone, M. (2015). Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7):883–921.
- Johnson, E. N. and Mooney, J. G. (2014). A comparison of automatic nap-of-the-earth guidance strategies for helicopters. *Journal of Field Robotics*, 31(4):637–653.
- Johnson, O. (2017). Sikorsky making strides with matrix autonomous technology. *Vertical Magazine*.
- Johnson, S. G. (2014). The nlopt nonlinear-optimization package. <http://ab-initio.mit.edu/nlopt>.
- Jung, D. and Tsiotras, P. (2008a). Bank-to-turn control for a small uav using backstepping and parameter adaptation. In *International Federation of Automatic Control IFAC World Congress*.

- Jung, D. and Tsiotras, P. (2008b). On-line path generation for small unmanned aerial vehicles using b-spline path templates. In *AIAA Guidance, Navigation and Control Conference*.
- Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011). Stomp: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation*.
- Karaman, S. and Frazzoli, E. (2010). Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems*.
- Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894.
- Karaman, S. and Frazzoli, E. (2012). High-speed flight in an ergodic forest. In *IEEE International Conference on Robotics and Automation*.
- Karaman, S. and Frazzoli, E. (2013). Sampling-based optimal motion planning for non-holonomic dynamical systems. In *IEEE International Conference on Robotics and Automation*.
- Kelly, A. and Nagy, B. (2003). Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22(7-8):583–601.
- Kendoul, F. (2012). Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378.
- Kerrigan, E. C. and Maciejowski, J. M. (2000). Invariant sets for constrained nonlinear discrete-time systems with application to feasibility in model predictive control. In *IEEE Conference on Decision and Control*.
- Kobilarov, M. (2012). Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871.
- Kuffner, J. J. and LaValle, S. M. (2000). RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*.
- Lantzch, R., Greiser, S., Wolfram, J., Wartmann, J., Mullhauser, M., Luken, T., Dohler, H.-U., and Peinecke, N. (2012). Allflight: Helicopter pilot assistance in all phases of flight. In *European Rotorcraft Forum 38*.
- Lapierre, L., Zapata, R., and Lepinay, P. (2007). Combined path-following and obstacle avoidance control of a wheeled robot. *The International Journal of Robotics Research*, 26(4):361–375.
- Laumond, J.-P. (1986). Trajectories for mobile robots with kinematic and environment constraints. In *International Conference on Intelligent Autonomous Systems*, pages 346–354.
- Laumond, J.-P., Sekhavat, S., and Lamiroux, F. (1998). Guidelines in nonholonomic motion planning for mobile robots. In *Robot motion planning and control*, pages 1–53. Springer.
- LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University.
- LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at <http://planning.cs.uiuc.edu/>.
- LaValle, S. M. and Kuffner Jr, J. J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400.
- Li, Y., Littlefield, Z., and Bekris, K. E. (2015). Sparse methods for efficient asymptotically optimal kinodynamic planning. In *Algorithmic Foundations of Robotics XI*, pages 263–282. Springer.
- Likhachev, M. and Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945.
- Lindemann, S. R. and LaValle, S. M. (2006). Multiresolution approach for motion planning under differential constraints. In *IEEE International Conference on Robotics and Automation*.

- Lipp, T. and Boyd, S. (2014). Minimum-time speed optimisation over a fixed path. *International Journal of Control*, 87(6):1297–1311.
- Luna, R., Şucan, I. A., Moll, M., and Kavraki, L. E. (2013). Anytime solution optimization for sampling-based motion planning. In *IEEE International Conference on Robotics and Automation*.
- MacAllister, B., Butzke, J., Kushleyev, A., Pandey, H., and Likhachev, M. (2013). Path planning for non-circular micro aerial vehicles in constrained environments. In *IEEE International Conference on Robotics and Automation*.
- Mayne, D. (1966). A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95.
- McGee, T. G., Spry, S., and Hedrick, J. K. (2005). Optimal path planning in a constant wind with a bounded turning rate. In *AIAA Guidance, Navigation, and Control Conference*.
- Micaelli, A., Samson, C., et al. (1993). Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. Technical report, INRIA Technical Report No. 2097.
- Michalska, H. and Mayne, D. (1993). Robust receding horizon control of constrained systems. *IEEE Transactions on Automatic Control*, 38(11):1623–1633.
- Nigam, N., Bieniawski, S., Kroo, I., and Vian, J. (2012). Control of multiple uavs for persistent surveillance: algorithm and flight test results. *IEEE Transactions on Control Systems Technology*, 20(5):1236–1251.
- Paden, B. and Frazzoli, E. (2016). A generalized label correcting method for optimal kinodynamic motion planning. *arXiv preprint arXiv:1607.06966*.
- Paduano, J., Wissler, J., Drozeski, G., Piedmonte, M., Dadkhah, N., Francis, J., Bold, J., and Langford, F. (2015). Talos: An unmanned cargo delivery system for rotorcraft landing to unprepared sites. In *American Helicopter Society Forum 71*.
- Parthasarathi, R. and Fraichard, T. (2007). An inevitable collision state-checker for a car-like vehicle. In *IEEE International Conference on Robotics and Automation*.
- Pettersson, P. O. and Doherty, P. (2004). Probabilistic roadmap based path planning for an autonomous unmanned aerial vehicle. In *Proc. of the ICAPS-04 Workshop on Connecting Planning Theory with Practice*.
- Pivtoraiko, M., Knepper, R. A., and Kelly, A. (2009). Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333.
- Prouty, R. W. (1995). *Helicopter performance, stability, and control*. R.E. Krieger Publishing Company.
- Ratliff, N., Zucker, M., Bagnell, J. A., and Srinivasa, S. (2009). CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*.
- Raveh, B., Enosh, A., and Halperin, D. (2011). A little more, a lot better: Improving path quality by a path-merging algorithm. *IEEE Transactions on Robotics*, 27(2):365–371.
- Reeds, J. A. and Shepp, L. A. (1990). Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393.
- Reif, J. and Wang, H. (1997). Non-uniform discretization approximations for kinodynamic motion planning. In *Algorithms for Robotic Motion and Manipulation*, pages 97–112. A.K. Peters.
- Scherer, S., Chamberlain, L., and Singh, S. (2012). First results in autonomous landing and obstacle avoidance by a full-scale helicopter. In *IEEE International Conference on Robotics and Automation*.
- Scherer, S., Singh, S., Chamberlain, L., and Elgersma, M. (2008). Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, 27(5):549–574.
- Schouwenaars, T. (2006). *Safe Trajectory Planning of Autonomous Vehicles*. PhD thesis, Massachusetts Institute of Technology.

- Schulman, J., Ho, J., Lee, A., Awwal, I., Bradlow, H., and Abbeel, P. (2013). Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: Science and Systems*.
- Select, M., Vana, P., Rollo, M., and Meiser, T. (2013). Wind corrections in flight path planning. *Int J Adv Robotic Sy*, 10(248).
- Staff, R. (2017). Lockheed martin demos tablet-controlled unmanned helicopter. *Robotics Business Review*.
- Stambler, A., Spiker, S., Bergerman, M., and Singh, S. (2016). Toward autonomous rotorcraft flight in degraded visual environments: experiments and lessons learned. In *Degraded Visual Environments: Enhanced, Synthetic, and External Vision Solutions 2016*. International Society for Optics and Photonics.
- Taamallah, S., Bombois, X., and Van den Hof, P. M. (2017). Trajectory planning and trajectory tracking for a small-scale helicopter in autorotation. *Control Engineering Practice*, 58:88–106.
- Tallavajhula, A., Choudhury, S., Scherer, S., and Kelly, A. (2016). List prediction applied to motion planning. In *IEEE International Conference on Robotics and Automation*.
- Tallavajhula, A. and Choudhury, S. (2015). List prediction for motion planning: Case Studies. Technical Report CMU-RI-TR-15-25, Robotics Institute, Pittsburgh, PA.
- Techy, L. (2011). Optimal navigation in planar time-varying flow: Zermelos problem revisited. *Intelligent Service Robotics*, 4(4):271–283.
- Techy, L., Woolsey, C. A., and Morgansen, K. A. (2010). Planar path planning for flight vehicles in wind with turn rate and acceleration bounds. In *IEEE International Conference on Robotics and Automation*.
- Urmson, C. and Simmons, R. (2003). Approaches for heuristically biasing RRT growth. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Vachtsevanos, G., Tang, L., Drozeski, G., and Gutierrez, L. (2005). From mission planning to flight control of unmanned aerial vehicles: Strategies and implementation tools. *Annual Reviews in Control*, 29(1):101–115.
- Verscheure, D., Demeulenaere, B., Swevers, J., De Schutter, J., and Diehl, M. (2009). Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327.
- Webb, D. J. and van den Berg, J. (2013). Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *IEEE International Conference on Robotics and Automation*.
- Whalley, M. S., Takahashi, M. D., Mansur, M. H., Ott, C., Minor, J., Morford, Z., Morales, E., Berrios, M., Goerzen, C., and Schulein, G. (2016). Flight test results for a new mission-adaptive autonomy system on the rascal juh-60a black hawk. In *American Helicopter Society Forum 72*.
- Whittle, R. (2016). Darpa do-it-all drone among new vtols nearing flight. *Breaking Defense*.
- Wzorek, M., Conte, G., Rudol, P., Merz, T., Duranti, S., and Doherty, P. (2006). From motion planning to control-a navigation framework for an autonomous unmanned aerial vehicle. In *21th Bristol UAV Systems Conference*.
- Yomchinda, T., Horn, J., and Langelaan, J. (2011). Autonomous control and path planning for autorotation of unmanned helicopters. In *American Helicopter Society Forum 68*.
- Zimmermann, M. (2016). *Sensor Based Online Path Planning for Rotorcraft Landing*. Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV.
- Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., Bagnell, J. A., and Srinivasa, S. S. (2013). CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193.

A Time To Collision

We now specify exactly how to compute the time to collision objective described in Section 3.5. We begin by talking about the representation offered to us by the perception system. The perception system internally stores \mathcal{P}_{obs} - the set

of all occupied positions under its current belief. It offers a function $\text{obs}(\mathbf{p}) = \mathbb{I}(\mathbf{p} \in \mathcal{P}_{\text{obs}})$ to say if a position is in an obstacle or not.

We also assume the perception system has an efficient representation (such as a distance field) to provide a distance function $d_{\text{obs}}(\mathbf{p}) = \min_{\mathbf{p}_{\text{obs}} \in \mathcal{P}_{\text{obs}}} \|\mathbf{p} - \mathbf{p}_{\text{obs}}\|_2$ and $p_{\text{obs}}(\mathbf{p}) = \arg \min_{\mathbf{p}_{\text{obs}} \in \mathcal{P}_{\text{obs}}} \|\mathbf{p} - \mathbf{p}_{\text{obs}}\|_2$

We now define a time to collision function $\text{ttc}(\sigma(t))$ in Algorithm 14. This function takes a trajectory point $\sigma(t)$ as input and returns a positive value in the range $[0, t_{\text{ttc,max}}]$ corresponding to the time to collision. The overall idea is to iterate over all relevant obstacles \mathcal{P}_{rel} , compute the time to collision and return the minimum over the set. To exactly compute this function we need $\mathcal{P}_{\text{rel}} = \mathcal{P}_{\text{obs}}$. This may be computationally intractable. Hence an approximation that we use in practice is $\mathcal{P}_{\text{rel}} = p_{\text{obs}}(\mathbf{p})$, i.e. the nearest obstacle.

Algorithm 14: $\text{ttc}(\sigma(t))$

```

1  $t_{\min} \leftarrow \infty$ ;
2 for  $\mathbf{p}_{\text{obs}} \in \mathcal{P}_{\text{rel}}$  do
3    $\mathbf{p} \leftarrow [x_{\sigma}(t) \ y_{\sigma}(t) \ z_{\sigma}(t)]^T$ ;
4    $\mathbf{v} \leftarrow [\dot{x}_{\sigma}(t) \ \dot{y}_{\sigma}(t) \ \dot{z}_{\sigma}(t)]^T$ ,  $\mathbf{v}_{xy} \leftarrow [\mathbf{v}(1) \ \mathbf{v}(2)]^T$ ,  $\mathbf{v}_z \leftarrow [|\mathbf{v}_{xy}|_2 \ \mathbf{v}(3)]^T$ ;
5    $\mathbf{d} \leftarrow (\mathbf{p}_{\text{obs}} - \mathbf{p})$ ,  $\mathbf{d}_{xy} \leftarrow [\mathbf{d}(1) \ \mathbf{d}(2)]^T$ ,  $\mathbf{d}_z \leftarrow [|\mathbf{d}_{xy}|_2 \ \mathbf{d}(3)]^T$ ;
6    $\Delta_{xy} \leftarrow \frac{\mathbf{d}_{xy}^T \mathbf{v}_{xy}}{\|\mathbf{d}_{xy}\|_2 \|\mathbf{v}_{xy}\|_2}$ ,  $\Delta_z \leftarrow \frac{\mathbf{d}_z^T \mathbf{v}_z}{\|\mathbf{d}_z\|_2 \|\mathbf{v}_z\|_2}$ ;
7    $t_{\text{ttc,obs}} \leftarrow$ 
      
$$\min \left( t_{\text{ttc,max}}, \underbrace{\left( \frac{\|\mathbf{d}\|_2}{\|\mathbf{v}\|_2} \right)}_{\text{time}} \underbrace{\left( 1 + \frac{\eta_{xy}}{2} \max(0, \Delta_{xy,\text{max}} - \Delta_{xy})^2 \right)}_{\text{inflation due to head-on collision}} \underbrace{\left( 1 + \frac{\eta_z}{2} \max(0, \Delta_{z,\text{max}} - \Delta_z)^2 \right)}_{\text{inflation due to vertical collision}} \right);$$

8   if  $t_{\text{ttc,obs}} \leq t_{\min}$  then
9      $t_{\min} \leftarrow t_{\text{ttc,obs}}$ ;
10 return  $t_{\min}$ ;

```

Lines 3,4 compute the position and velocity. Line 5 computes distance vector to nearest obstacle. Line 6 computes the dot product between the velocity vector and the distance vector. This is to measure the offset of the obstacle from the current direction in which the robot is heading. Line 7 computes the main time to collision. This is computed as the ratio of the norm of distance over velocity. This value is inflated based on the dot products. The algorithm computes this term for all relevant obstacle and returns the minimum time to collision.

B Library of Expert Planners

We now provide details on the hand-designed library of expert planners used. We organize these expert planners into different classes. Each of these classes define a surrogate problem and a class of planning algorithms that can efficiently solve problems in this class.

B.1 Sampling-based approaches on a curvature constrained state space

Table 5 shows the surrogate problem. We now specify different sampling strategies which can be used to create different expert planners as mentioned in Table 6.

1. **Uniform** Let $U(\text{Vol})$ denote an uniform probability distribution over volume Vol . Let $\text{Cuboid}(\mathbf{p}_{\min}, \mathbf{p}_{\max})$ be the volume of the cuboid denoting the valid workspace where \mathbf{p}_{\min} and \mathbf{p}_{\max} are the corners.
2. **Workspace** This sampling scheme is specific to the RRT* algorithm. The overview of this sampling strategy is that a 3D workspace point (x, y, z) is sampled leaving the dimension ψ as a free variable. Among the set

Table 5: Surrogate problem on curvature constrained state space

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathcal{P} \circ SO(2)$ $\xi[0, 1] \mapsto (x, y, z, \psi)$	$\mathbf{x}_s = (s_x, s_y, s_z, s_\psi)$ $\mathbf{x}_g = (g_x, g_y, g_z, g_\psi)$	$\tilde{J}(\xi) = \int_0^1 \ \dot{\xi}(\tau)\ d\tau$	$\tilde{\mathcal{F}} : \psi = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$ $\tilde{\mathcal{H}} : \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \leq \kappa_{\max}$ $\frac{\dot{z}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \leq \gamma_{\max}$ $\tilde{\Xi}_{\text{valid}} : \mathbf{nfz}(x, y, z) = 0$ $\mathbf{ttc}(x, y, z) \geq t_{\text{coll}, \min}$

of near neighbours, the best parent to this workspace point is computed. The ψ value is then the resulting value by steering the parent towards this configuration.

3. Tunnel

This sampling scheme is specific to the RRT* algorithm. This extends the workspace sampling strategy by sampling workspace points in a tunnel around a nominal 3D path (usually the analytic Dubins path joining \mathbf{x}_s and \mathbf{x}_g). The tunnel radius is specified by δ_{tunn} .

Table 6: Library of expert sampling based planner for curvature constrained state space

Name	Algorithm	Sampler	Parameters
RRT*Uniform ¹	RRT*	Uniform	Radius (γ) : 3.0
RRT*Workspace ¹	RRT*	Workspace	Radius (γ) : 3.0
RRT*Workspace ²	RRT*	Workspace	Radius (γ) : 1.0
RRT*Tunnel ¹	RRT*	Tunnel	Radius (γ) : 3.0, Tunnel (δ_{tunn}) : $\frac{1}{\kappa_{\max}}$
RRT*Tunnel ²	RRT*	Tunnel	Radius (γ) = 3.0, Tunnel (δ_{tunn}) : $\frac{1}{3\kappa_{\max}}$
BIT* ¹	BIT*	Uniform	Radius (γ) = 3.0, Batch : 300
BIT* ²	BIT*	Uniform	Radius (γ) = 3.0, Batch : 500
RRTConnect ¹	RRTConnect	Uniform	
RRTConnect ²	RRTConnect	Tunnel	Tunnel (δ_{tunn}) : $\frac{1}{\kappa_{\max}}$
T-RRT ¹	T-RRT	Uniform	Temperature (T) : 2.0, Frontier (ρ) : 0.1
T-RRT ²	T-RRT	Uniform	Temperature (T) : 10.0, Frontier (ρ) : 0.1
LBT-RRT ¹	LBT-RRT	Uniform	Approximation (ε) : 0.40
LBT-RRT ²	LBT-RRT	Uniform	Approximation (ε) : 0.04
InformedRRT* ¹	Informed RRT*	Uniform	Radius (γ) : 3.0
STRIDE ¹	STRIDE	Uniform	
EST ¹	EST	Uniform	
EST ²	EST	Tunnel	Tunnel (δ_{tunn}) : $\frac{1}{\kappa_{\max}}$

B.2 Sampling-based approaches on a holonomic 3D state space

Table 7: Surrogate problem on 3D state space

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathcal{P}$ $\xi[0, 1] \mapsto (x, y, z)$	$\mathbf{x}_s = (s_x, s_y, s_z)$ $\mathbf{x}_g = (g_x, g_y, g_z)$	$\tilde{J}(\xi) = \int_0^1 \ \dot{\xi}(\tau)\ d\tau$	$\tilde{\Xi}_{\text{valid}} : \mathbf{nfz}(x, y, z) = 0$ $\mathbf{ttc}(x, y, z) \geq t_{\text{coll}, \min}$

Table 7 shows the surrogate problem. The heading ψ can be computed as $\psi = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right)$ once the path is computed. We now specify different expert planners in Table. 8.

Table 8: Library of expert sampling based planner for 3D holonomic state space

Name	Algorithm	Sampler	Parameters
BIT [*] ³	BIT*	Uniform	Radius (γ) = 3.0, Batch : 300
RRTConnect ³	RRTConnect	Uniform	

B.3 Trajectory Optimization Approaches on the Space of Smooth Trajectories

Table 9: Surrogate problem on space of smooth trajectories

Space	Start / Goal	Objective	Constraints
$\mathcal{S} = \mathcal{P}$ $\xi \in [0, 1] \mapsto (x, y, z)$	$\mathbf{x}_s = (s_x, s_y, s_z)$ $\mathbf{x}_g = (g_x, g_y, g_z)$	$\tilde{J}(\xi) = \int_0^1 \left(\lambda \left\ \dot{\xi}(\tau) \right\ _2^2 + \frac{1}{2} (\text{ttc}_{\max} - \text{ttc}(\xi(\tau)))^2 \left\ \dot{\xi}(\tau) \right\ \right) d\tau$	$\tilde{\mathcal{H}} : \frac{\dot{z}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \leq \gamma_{\max}$

Table 9 shows how a surrogate trajectory optimization problem can be created. We created two such expert planners CHOMP¹, which ignores the glideslope constraint $\tilde{\mathcal{H}}$ and CHOMP² which solves a constrained optimization problem. Both these planners use covariant gradient descent (Ratliff et al., 2009) to solve the optimization problem.

B.4 Discrete search on a graph of dynamically feasible motion primitives

Table 10: Surrogate problem on a graph of dynamically feasible motion primitives

Space	Start / Goal	Objective	Constraints
$\mathcal{S} : \mathcal{G} = (V, E)$ $\xi = (e_1, e_2, \dots, e_l)$	$\mathbf{x}_s = \text{ProjOnGraph}(s, \mathcal{G})$ $\mathbf{x}_g = \text{ProjOnGraph}(g, \mathcal{G})$	$\tilde{J}(\xi) = \sum_{i=1}^l e_i $	$\tilde{\mathcal{F}} : \text{corr}(e_i) = 1$ $\text{nfz}(e_i) = 0$ $\tilde{\mathcal{H}} : \text{ttc}(e_i) \geq t_{\text{coll}, \min}$

Table 10 shows the surrogate problem. Since the planners are discrete search based, the algorithms only deal with the graph. The resolution and primitives of the graph however encode the dynamic constraints. We use two kinds of graphs, ones with primitives constant curvature and others with primitives as straight lines. The former graph is also 4D while the latter is 3D. For the latter, the heading ψ can be computed as $\psi = \tan^{-1} \left(\frac{\dot{y}}{\dot{x}} \right)$ once the path is computed. We now specify the different expert planners in Table 11. All of these planners employ weighted A* heuristic search.

Table 11: Library of expert discrete search based planners

Name	Space	Primitives	Heuristic
A [*] ¹	$\mathcal{S} = \mathcal{P} \circ SO(2)$	$\pm \frac{\pi}{2}$ constant curvature arcs	Dubins, Inflation: 1.0
A [*] ²	$\mathcal{S} = \mathcal{P} \circ SO(2)$	$\pm \frac{\pi}{4}$ constant curvature arcs	Dubins, Inflation: 10.0
A [*] ³	$\mathcal{S} = \mathcal{P}$	straight line, 27-connected, 5m res	Euclidean, Inflation: 1.0
A [*] ⁴	$\mathcal{S} = \mathcal{P}$	straight line, 27-connected, 1m res	Euclidean, Inflation: 10.0

B.5 Hand-designed precision planners

Finally, this last group of planners are designed to overfit to certain planning problems. They are created based on inspecting commonly occurring problems and designing custom sampling strategies / brute force path enumeration strategies.

B.5.1 Sampling detour points

This class of planners use the curvature constrained state space defined in Table 5. They also use the RRT* algorithm to plan. However, they use a highly customized sampling strategy which belongs to one of two categories:

1. *SingleDetour*: In this sampling strategy, we first find a violation point p . To find this, we step through points along the line joining start to the goal till a point p is found to be in collision with obstacles. We then create a surface perpendicular to the direction of the line at this point p . The samples are constrained to lie on this surface. This is equivalent to finding a “detour point”. Depending on the width of the surface (expressed as fraction of $\frac{1}{\kappa_{\max}}$) we have two planners - *SingleDetour*¹ (width: $\frac{1}{\kappa_{\max}}$) and *SingleDetour*² (width: $\frac{1}{3\kappa_{\max}}$).
2. *DoubleDetour*: This is similar to *SingleDetour* except we find two violation points p_1 and p_2 - one from start to goal and one in the reverse direction. Hence two surfaces are created to constrain sampling. We also create two such planners. *DoubleDetour*¹ uses a width: $\frac{1}{\kappa_{\max}}$ and samples uniformly in heading space. *DoubleDetour*² uses a width: $\frac{1}{\kappa_{\max}}$ but uses *Workspace* sampling.

B.5.2 Fixed descent direction

In this method, we collect a set of problems which none of the planners in the library were able to solve. We then solve such problems by running an RRT* method for a long time (100s). The solutions of all these problems are collected. All these solutions are then discretized to 50 waypoints. They are then concatenated to create a matrix M with dimensions $n \times 50$ where n is the number of trajectories (one matrix for each dimension). We apply principle component analysis to extract a set of principle axis for this matrix. These directions represent motions along with a trajectory has to descend to recover the paths. The five principle directions then give rise to planners *FixedDescent*¹, *FixedDescent*², *FixedDescent*³, *FixedDescent*⁴, *FixedDescent*⁵.

B.5.3 Brute force path search

The final precision planner that we create is doing a brute force path search by perturbing the path laterally to create a planner *LatPrim*¹.

C Dynamics Projection Filter

The DPF (Choudhury and Scherer, 2015) accepts an infeasible trajectory (workspace trajectory) as input, uses a controller to track this trajectory and the outputs the configuration trajectory traced out by the system. To provide guarantees about the output, the first component required is a control-Lyapunov function (CLF) that stabilizes around a feasible workspace trajectory. Since the controller can observe all states, has a perfect model and is free from any disturbance, approaches such as feedback linearization and backstepping can be applied (Micaelli et al., 1993; Jung and Tsiotras, 2008a; Lapiere et al., 2007).

Let the workspace trajectory be $\xi : [0, 1] \rightarrow \mathbb{R}^w$. A control-Lyapunov stabilized trajectory firstly requires the existence of a feedback control $u = K(x, \xi, \tau)$ where $x \in \mathbb{R}^d$ is the configuration space coordinate of the robot, τ is the index of the workspace trajectory. It also requires the definition of index dynamics $\dot{\tau} = \Gamma(x, \xi, \tau)$. Careful selection of these functions can ensure a function $V(x, \xi, \tau)$ to satisfy the Lyapunov criteria when ξ is dynamically feasible. If $x \in X_\xi$ implies perfect tracking, the Lyapunov criteria is $V(x, \cdot) > 0$, $\dot{V}(x, \cdot) < 0$, $\forall x \in X \setminus X_\xi$ and $V(x, \cdot) = 0$, $\forall x \in X_\xi$ (globally asymptotic stable version).

A further local exponential stability is also required, i.e.,

$$\left| \frac{dV(\cdot, \tau)}{d\tau} \right| > \alpha V(\cdot, \tau), \quad \alpha > 0, \quad V(\cdot) < V_{max}.$$

where the rate of exponential stability α determines how fast convergence occurs. This property will be used to guarantee decay properties of the CLF. We used the an extension of the controller presented in (Jung and Tsiotras, 2008a)

The output $x(t) = \text{DPF}(\xi)$ is given by

$$\begin{bmatrix} x(t) \\ \tau(t) \end{bmatrix} = \begin{bmatrix} x(0) \\ 0 \end{bmatrix} + \int_0^t \begin{bmatrix} f(x(t), K(x(t), \xi, \tau(t))) \\ \Gamma(x(t), \xi, \tau(t)) \end{bmatrix} dt \quad (25)$$

In the scope of this work, we consider ξ to be approximated by a set of workspace samples at equal discretization $\Delta\tau$: $\xi \approx (q_1, q_2, \dots, q_n)^T \in \mathbb{R}^{n \times w}$, with q_0 and q_{n+1} the fixed starting and ending points of the trajectory. This will facilitate concrete expressions on bounds that are parameterization dependent. However, the method remains valid for other parameterizations, such as splines.

Under the assumption that the linear segments between waypoints are dynamically feasible (there exists $u \in U$ which allows perfect tracking), the Lyapunov function $V(\cdot)$ converges in the straight line portion of ξ . However, at every waypoint it increases by $\Delta V > 0$ because of the angle change at the waypoint. The more jagged ξ is the more the cumulative effect of ΔV will be. The maximum $V(\cdot)$ at anytime determines how much deviation $x(\cdot)$ has from ξ . We establish in (Choudhury and Scherer, 2015) that under certain assumptions about $V(\cdot)$, a bounded Lyapunov value implies a bounded intersegment deviation.

Theorem 1 (Bounded Intersegment Deviation). *Given a desired bound on the Lyapunov function $V(x, \xi, \tau) \leq V_{max}$, $\forall x, \tau \in \text{DPF}(\xi)$, the intersegment deviation is also bounded, i.e.*

$$\left(1 + \frac{(q_i - q_{i-1})^T (q_i - q_{i+1})}{\|q_i - q_{i-1}\| \|q_i - q_{i+1}\|} \right) \leq \rho_{max}.$$