The Planner Ensemble and Trajectory Executive: A High Performance Motion Planning System with Guaranteed Safety

Sanjiban Choudhury
sanjiban@cmu.edu
PhD Student
Carnegie Mellon University
Pittsburgh, PA

Sankalp Arora
asankalp@cmu.edu
Masters Student
Carnegie Mellon University
Pittsburgh, PA

Sebastian Scherer
basti@cmu.edu
Systems Scientist
Carnegie Mellon University
Pittsburgh, PA

ABSTRACT

Abstract— Autonomous helicopters are required to fly at a wide range of speed close to ground and eventually land in an unprepared cluttered area. Existing planning systems for unmanned rotorcrafts are capable of flying in unmapped environments, however they are restricted to a specific operating regime dictated by the underlying planning algorithm. We address the problem of planning a trajectory that is computed in real time, respects the dynamics of the helicopter, and keeps the vehicle safe in an unmapped environment with a finite horizon sensor. We have developed a planning system that is capable of doing this by running competing planners in parallel. This paper presents a planning architecture that consists of a trajectory executive - a low latency, verifiable component - that selects plans from a planner ensemble and ensures safety by maintaining emergency maneuvers. Here we report results with an autonomous helicopter that flies missions several kilometers long through unmapped terrain at speeds of upto 56 m/s and landing in clutter. In over 6 months of flight testing, the system has avoided unmapped mountains, popup no fly zones, and has come into land while avoiding trees and buildings in a cluttered landing zone. We also present results from simulation where the same system is flown in challenging obstacle regions - in all cases the system always remains safe and accomplishes the mission. As a result, the system showcases the ability to have a high performance in all environments while guaranteeing safety.

INTRODUCTION

Autonomous helicopters have a high demand in applications such as cargo delivery, emergency rescue operations and surveillance due to their dexterity in operating in close vicinity to ground. These operations typically involve traveling at high speeds at low altitudes and eventually landing in an unstructured and partially occluded environment. From a motion planning perspective, this problem has several challenging aspects. Firstly, the operating environment is mostly unmapped, requiring rapid online planning as the sensor incrementally discovers new obstacles. Secondly, the speed of operation scales across a wide spectrum requiring plans to adapt accordingly. Thirdly, the missions require the vehicle to proceed as fast as possible to touchdown, thus demanding high performance as well as guarantees on safety. Finally, the planner has to respect the dynamics of the under-actuated non-linear system.

Motion planning literature (LaValle (Ref. 11)) is divided into various classes based on the representation of the search graph and the methodology for updating this graph. Each class has its advantages and disadvantages and usually the application dictates the effectiveness of a given approach. Surveys in the field of autonomous rotorcraft systems by Kendoul (Ref. 9) and Goerzen et al. (Ref. 6) have revealed however that all classes of approach are used in practice. These algorithms are usually modified to leverage some assumption about the specific problem statement to gain performance advantages over other approaches.

Fig. 1: Boeing's Unmanned Little Bird autonomously lands in snow. The planning system plans direct to touchdown avoiding enroute no fly zones, unmapped terrain, trees and clutter in the landing zone.

The most general approach to the planning problem is to create a search graph and perform deterministic search. Such methods come with guarantees of resolution optimality and complexity bounds. In the area of field robotics, the nomi-

Presented at the AHS 70th Annual Forum, Montréal, Québec, Canada, May 20–22, 2014.

nal graph search has undergone several augmentations to plan in real time. Anytime search methods as reviewed by Ferguson et al. (Ref. 3) take advantage of the incremental cost update and need to replan in robotics. Differential constraints are handled using custom lattice (Pivtoraiko et al. (Ref. 16)) and multi-resolution graphs (Likhachev et al. (Ref. 12)) focus the search to improve performance. There has been some promising results for micro-aerial vehicles (MacAllister et al. (Ref. 14)), however, such methods have been used in a limited way for aerial vehicles. The state of the art results use 3D A* to generate waypoints (Tsenkov et al. (Ref. 20), Whalley et al. (Ref. 21)). A possible reason for not having extended use is that designing an appropriate search graph in the high dimensional space to get quality trajectories in real time is a non-trivial problem.

Numerical optimization techniques are popular for planning trajectories offline. Situations where obstacles are known, the problem can be framed as a NLP problem or more efficient techniques such as MILP (Schouwenaar et al. (Ref. 19)) can be applied. However, these methods are not suited for real-time obstacle avoidance. Another school of thought in optimization is perturbing around an initial guess of the optimal. Quinlan and Khatib introduced the concept of a trajectory as an elastic band (Ref. 17) where obstacles stretch the elastic band. More recently, methods such as CHOMP (Ratliff et al. (Ref. 18)) have had a lot of success by taking covariant gradients. However, in a complex environment, gradient based methods tend to get trapped in a local minimum.

Sampling based approaches have had the widest popularity for high dimensional search, primarily due to the ease of their implementation. These methods did not come with any guarantee of optimality, until recently with the introduction of RRT* by Karaman and Frazzoli (Ref. 4). This method provides asymptotic optimality by the virtue of repairing the search tree. For certain environments, sampling provides quick feasible solutions. However, once quality of the trajectory becomes an issue, the stochasticity of the samples cause oscillatory results. Usually some level of post processing and relaxation is required.

Motion planning algorithms by themselves cannot guarantee the safety of the system. Even in the case where an algorithm can be proved to be sub-optimal, an architecture is required to handle cases where the planner is unable to find a feasible path. Goerzen and Whalley (Ref. 7) present an architecture where the speed is modulated based on the ability of the vehicle to stop within the sensor range. However, this method is far too conservative to enable high performance. Moreover, there is no discussion of how one can introduce formal verification techniques in the planning paradigm to ensure safety.

In this paper, we present a systems architecture that uses an ensemble of complimentary planners running in parallel to deal with diverse situations. This process of letting planners compete takes away the need of reasoning about which planner best suits a planning problem. We also present the trajectory executive, a low latency verifiable module which picks a plan and ensures safety of the system. We have tested the architecture by conducting numerous flight tests on the Boeing's Unmanned Little Bird (The ULB) as part of the Autonomous Aerial Cargo Utility System (AACUS)) program as shown in Fig. 1.

PROBLEM DEFINITION

The general planning and control structure is as follows. The trajectory planner subsystem computes a dynamically feasible command trajectory and sends it to the Flight Control System (FCS). The FCS tracks a reference point on this trajectory, known as a lookahead, and sends feedback to the trajectory planner. In order to respect controller assumptions, the trajectory planner is only allowed to replan from beyond the lookahead.

Let $X \subset \mathbb{R}^4$ be a compact set. Let $\sigma(t) = \{x(t), y(t), z(t), \psi(t)\} \in X$ be the time parameterized command trajectory of the coordinates of the centre of mass of the helicopter and the heading of the vehicle (North East Down convention). $\sigma(t)$ is defined over the domain $[0, t_f]$, where t_f is the total time duration.

Let σ_0 denote the lookahead point of the controller following the trajectory $\sigma(t)$. The look ahead point is the reference point for the controller. In order to prevent step jumps in the reference, this point should remain unchanged. Let $\Sigma_f \subset X$ denote a terminal invariant set. This set can be of two distinct classes. Firstly, it can be a set of steady state allowable touchdown positions. Secondly, it can also be a loiter pattern. In that case the goal of the planner is to reach a state on the pattern and engage a repetitive loiter loop.

The trajectory $\sigma(t)$ is subjected to a set of dynamics constraints $g(\sigma(t), \dot{\sigma}(t), \ddot{\sigma}(t), \dots) \leq 0$. These inequality constraints are defined over derivatives of $\sigma(t)$ up to order 3. Since the underlying helicopter control is a hybrid controller, these constraints are defined by two distinct behaviours based on a transition speed. Firstly, the high speed behaviour enforces a non-holonomic coordinated turn constraint with bounded sideslip. This implies that the heading, $\psi(t)$, must be closely aligned with the velocity vector $\tan^{-1} \frac{\dot{y}(t)}{\dot{x}(t)}$. The difference $\beta(t) = |\psi(t) - \tan^{-1} \frac{\dot{y}(t)}{\dot{x}(t)}|$ is the align angle. In this mode acceleration constraints are place along $\psi(t)$ (longitudinal) and $\ddot{z}$ (vertical). Secondly, the low speed behaviour decouples the heading from velocity vector. Acceleration constraints are now also placed on lateral acceleration. This along with other aerodynamics constraints are described in detail in Table 1.

The cost J is a line integral over the cost functional $c(\sigma(t))$ over the domain $[0, t_f]$. This cost functional is a weighted sum of several functionals $c_i(\sigma(t))$ as shown in Table 2. Firstly, total time duration is penalized. Secondly, proximity to obstacles is penalized. This is based on a time to collision metric (Arora et al. (Ref. 1)). Thirdly, a penalization is applied for entering a human defined area known as No Fly Zone, $X_{NFZ} \subset X$. Finally, a penalization is applied for entering a

Table 1: Constraints on trajectory

Definition	Equation
Transition speed	v_{trans}
Align angle	$\beta(t) = \lvert\psi(t) - \tan^{-1}\frac{\dot{y}(t)}{\dot{x}(t)}\rvert$
Roll angle	$\phi(t) = \tan^{-1}\frac{v(t)\psi(t)}{g}$
Longitudinal velocity Lateral velocity	$\begin{bmatrix} v_{long}(t) \\ v_{lat}(t) \end{bmatrix} = R^T(\psi(t)) \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix}$

High Speed Constraint $(v(t) \geq v_{trans})$	**Equation**
Roll	$\lvert\phi(t)\rvert < \phi_{max}$
Roll rate	$\lvert\dot{\phi}(t)\rvert < \dot{\phi}_{max}$
Sideslip	$\lvert\beta(t)\rvert < \beta_{max}$
Sideslip rate	$\lvert\dot{\beta}(t)\rvert < \dot{\beta}_{max}$
Longitudinal velocity	$\lvert v_{long}(t)\rvert < v_{long,max}$
Lateral velocity	$\lvert v_{lat}(t)\rvert < v_{lat,max}$
Vertical velocity	$\lvert\dot{z}(t)\rvert < v_{vert,max}$
Longitudinal acceleration	$\lvert\dot{v}_{long}(t)\rvert < a_{long,max}$
Vertical acceleration	$\lvert\dot{v}_{vert}(t)\rvert < a_{vert,max}$
Glide slope	$\frac{\dot{z}}{v_{vert}} < \gamma_{max}$

Low Speed Constraint $(v(t) < v_{trans})$	**Equation**
Heading rate	$\lvert\psi(t)\rvert < \psi_{max}$
Longitudinal velocity	$\lvert v_{long}(t)\rvert < v_{long,max}$
Lateral velocity	$\lvert v_{lat}(t)\rvert < v_{lat,max}$
Vertical velocity	$\lvert\dot{z}(t)\rvert < v_{vert,max}$
Longitudinal acceleration	$\lvert\dot{v}_{long}(t)\rvert < a_{long,max}$
Lateral acceleration	$\lvert\dot{v}_{lat}(t)\rvert < a_{lat,max}$
Vertical acceleration	$\lvert\dot{v}_{vert}(t)\rvert < a_{vert,max}$
Glide slope	$\frac{\dot{z}}{v_{vert}} < \gamma_{max}$

low velocity state while a high tail wind. This situation leads to a destabilizing state for the helicopter.

The goal of the trajectory planner is to find an optimal path that respects the constraints. This is defined as follows:

Table 2: Cost functionals

Cost Functional $\int c_i(\sigma(t))dt$	**Equation**
Time duration	t_f
Time to collision	$\int(t_{max} - t_{coll}(\sigma(t)))^2 dt$
No Fly Zone	∞ if $\exists t, \sigma(t) \in X_{NFZ}$
Wind Cost	∞ if $\exists t, \sigma(t) \in X_{tail,wind}$

$$
\begin{array}{rc}
find & \sigma(t) = \{x(t), y(t), z(t), \psi(t)\}, t_f \\
minimize: & J = \int_0^{t_f} c(\sigma(t))dt \\
constraints: & \sigma(0) = \sigma_0 \\
 & \sigma(t_f) \in \Sigma_f \\
 & g(\sigma(t), \dot{\sigma}(t), \ddot{\sigma}(t), \dots) \leq 0 \\
 & J < \infty
\end{array}
\tag{1}
$$

The key reason which makes this non-linear optimization problem hard to solve is the finite sensing horizon. Given the fact that the helicopter is moving at a high speed and unmapped obstacles are known only when they enter the sensor horizon, there is a demand for planning safe optimal trajectories rapidly.

Before committing to an approach methodology, important design criteria have to be investigated - both from the perspective of the nature of the optimization problem and from a practical system design standpoint. We summarize this as set of observations and requirements.

Difficulty in making assumptions about the environment

Planning algorithm rely on assumptions about the environment to lay claims on feasibility and optimality. However, it is unclear how to map the real environment a helicopter operates in to the space of assumptions. For example, to say that a search graph must have at least one feasible path is enforcing a non-trivial constraint on the required probability distribution of environments. Similarly to say that there must not be a local minima also leads to a non-trivial probability distribution. Moreover, taking each of these assumptions to the conservative limit cancels out environments in which the helicopter is actually required to operate.

The constrained non-linearity of the optimization problem aggravates the need for strong assumptions to still be able to deliver real time solutions. Search based methods have to make graphs on this non-linear manifold, gradient based methods have to enforce sufficient smoothness and sample based planners have to bias samples with considerations on the reachability of the system.

Difficulty in characterizing algorithm performance

In the event that assumptions made by the algorithm holds, the performance of the algorithm can still have large fluctuations. In the case of discrete search algorithms or gradient based optimizers, the stochasticity comes from obstacle distributions. In case of sampling based planners, performance varies even more widely due to their inherent stochasticity. This makes it unclear that given a snapshot of the current environment, which method would be able to produce a plan within the given time budget.

Difficulty in ensuring system safety

In the event that assumptions made by the algorithm fail, it is desired that the system executes a safe response. To know

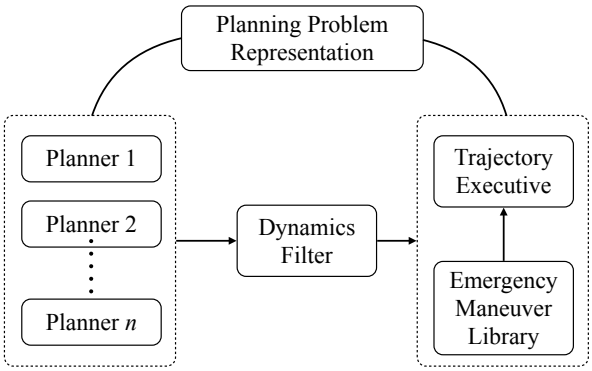


Fig. 2: Planner ensemble and trajectory executive architecture.

a-priori when this will happen is non-trivial. If the event does occur, whether the system is in a state that can reach a safe state is also a requirement.

Difficulty in verification of algorithm functionality

Verification of any practical system is vital. For a general scenario, an implementation of a planning algorithm is non-trivial to verify. This is because when seen as a blackbox unit, the input to a planner is an infinite space of environments and constraints, and the correctness of the output is an optimization problem in itself. As a compromise, it is at least required that the system behave safely at all times and abide by the rules imposed by the controller.

APPROACH OVERVIEW

In the previous section, we defined the problem and established the need for a planning system that can verifiably ensure safety while being able to adjust its search approach as the environment changes. We will now introduce a system architecture that addresses this issue.

The central idea is to have an *ensemble of planners* running in parallel. These planners explore the state spaces in parallel in a complimentary fashion and bid their plans. These plans are then fed to a *trajectory executive*. The executive is a low latency verifiable unit whose job is to receive plans, ensure safety and send the best plan to the controller.

Fig. 2. shows the system architecture. The planner ensemble submit their plans to a *dynamics filter* module. This module is capable of taking sufficiently smooth plans and checking or enforcing dynamics constraints. The trajectory executive receives these feasible plans and selects the best plan. It keeps around alternate plans in order to have a low latency reaction to minor obstacles. It then uses an *emergency maneuver library* to check if a plan is safe and executes the maneuver if not.

TRAJECTORY EXECUTIVE

In this section, we introduce the concept of a trajectory executive. The trajectory executive is a trajectory selection and verification unit that guarantees the safety of the vehicle. We will discuss the rationale behind this modular approach, how trajectories are selected and how the module ensures safety.

Rationale

A nominal planning system consists of a trajectory planner operating at a fixed frequency. Every planning cycle it looks at the current pose of the vehicle and plans a path to the goal while optimizing a cost function along the way. In planning literature as reviewed by LaValle (Ref. 11), various guarantees exist on the properties of planning algorithms, however, it is difficult to translate these guarantees to the system on the whole. These difficulties are summarized below

1. *Safety perspective* - A single planner framework makes it difficult to ensure that the vehicle is safe at all times while attempting to compute the optimal path to goal. This requires a very high frequency monitoring of the environment and regulation of vehicle velocity to be combined with a more lower frequency computationally intense process of computing an optimal path.

2. *Systems perspective* - Even though a single algorithm can be laboriously constructed to handle all corner cases, it is difficult to ensure that the module will behave properly when faced with issues such as latency in the pipeline, inadequate planning budget, memory corruption errors and modules temporarily dying. The planner has to take appropriate evasive actions in such cases making it flight critical.

3. *Algorithm perspective* - A helicopter performing missions over long distances which involves take-off and landing encounters a wide variety of situations of different fidelity. It is not evident that a single best approach can exist. A trajectory optimizer quickly perturbs a plan locally to arrive at a sensible answer but can often get trapped in undesirable local minima. A discrete graph planner always arrives at the optimal solution but its planning time and plan quality is dependent on the resolution of the graph. A sampling based planner attempts to balance between both plans but has an inherent stochastic variation.

In our approach, we make our system modular and create a flight critical component - the trajectory executive. Operating at a high frequency ($\sim$10 Hz), it ensures that the vehicle follows a safe, feasible trajectory. It has the following properties:

1. It collects trajectories, evaluates closed loop behavior of these trajectories and picks the optimal one.

2. It retains a set of alternate trajectories which are sub-optimal but potential candidates for being optimal in case an obstacle appears on the path of the current command trajectory.

3. It ensures the trajectory being followed is guaranteed safe.

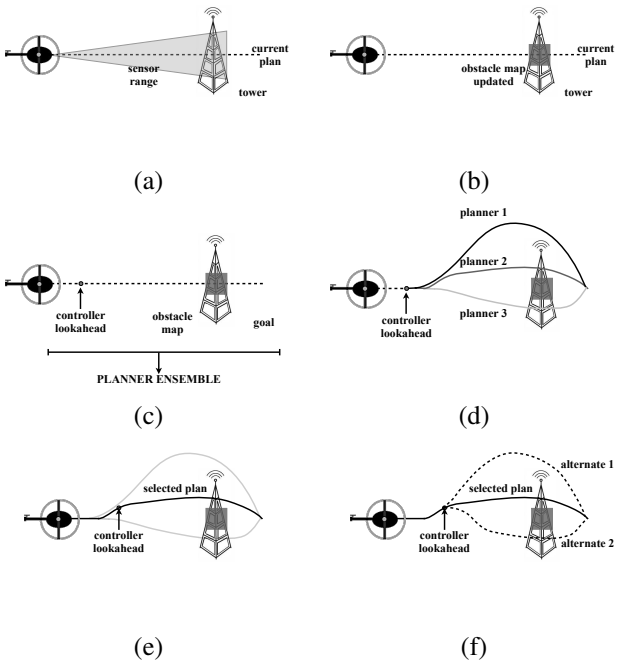


Fig. 3: The sequence of events that occur in the planning subsystem. (a) An obstacle enters the sensor range (b) The obstacle appears in the obstacle map (c) In the next iteration, the planner ensemble receives the planning problem. Note that by now the executive has invalidated the old plan and is either following an alternate or following the old trajectory as long as its safe. (d) The planners asynchronously submit plans to the executive (e) The executive selects a plan which is sent out to the vehicle (f) It repairs the alternates and keeps them in reserve.

4. It guarantees that at least one backup emergency maneuver always exists.

5. It is minimal, constant memory, and constant runtime to enable DO-178B verification.

6. In the event of there being no feasible trajectories or a critical module failure, it executes the backup emergency maneuver to take the vehicle to a safe invariance set.

Because of the well defined nature of these requirements, it becomes possible to fully verify the correctness of the trajectory executive - which implies the correctness of the vehicle.

Trajectory Selection

The trajectory executive needs a pool of trajectories to choose from. In a later section, we will expand on how this pool is generated. In this subsection, we discuss how it selects a trajectory from the pool as well as how it decides to choose M alternate trajectories. An illustration of the sequence of events is shown in Fig. 3. The selection cycle is as follows:

1. Collect at most N trajectories submitted by planners and add them to a queue.

2. Repair old plans, i.e., plans that originate from before the control lookahead point. It repairs them by running a polynomial interpolation from the lookahead to a projection point on the trajectory. If the resultant is not dynamically feasible, it is rejected.

3. Discard plans which are not guaranteed safe.

4. From the list of trajectories, select the one that has the least cost and set the corresponding trajectory as command.

5. Choose M alternate trajectories from remaining trajectories in the queue based on a fitness function and retain them in the queue while throwing the rest out.

This ensures that the trajectory executive selects from at most $N+M$ plans at any given cycle, thus bounding its cycle time.

To choose M alternate trajectories, a fitness function is used. The fitness function awards diversity while ensuring the cost of the paths are within a bounded variation from optimal. Thus it solves the following problem

$$find : \sigma_i = (x(t), u(t)) \quad \forall i = 1 \cdots m$$

subject to the following constraints

1. $(S(\sigma_i) \cap \{S(\sigma_0) \cup \cdots \cup S(\sigma_{i-1})\}) \leq \gamma \|S(\sigma^*)\|$ (Limited sharing of new alternative and previous alternative trajectory swath)

2. $J_{cl}(\sigma_i)$ is $(1+\varepsilon)J_{cl}(\sigma^*)$ (Bounded variation from optimal cost)

where $S(.)$ is the swath of a trajectory, i.e. a volume of safety radius around a trajectory. $J_{cl}(\sigma)$ is the cost of the closed loop trajectory following σ.

Condition 1 ensures that the alternate trajectories are diverse. Specifically, if one of the alternate trajectories ends up in collision with an obstacle, the other trajectories are more likely to not be affected. Condition 2 ensures that the trajectories have bounded sub-optimality - only trajectories of acceptable quality are retained.

Ensuring Safety

A critical purpose of the trajectory executive is to ensure safety. Here safety does not just imply a collision free trajectory, but a trajectory that is guaranteed safe (Arora et al. (Ref. 1)). This is defined as a trajectory from which an emergency maneuver always exists within the known free space volume. The trajectory executive ensure the safety of the rotorcraft by using the emergency maneuver library to enforce the constraint that the current and next state of the helicopter always lies in the positive invariant set, which does not intersect the obstacles and stays within the known volume. The algorithm to ensure safety is explained in Algorithm 1.

Algorithm 1: Emergency Maneuver Trajectory Set Application for Reactive Safety
Input: Safety library at previous timestep $\Phi_{Previous}$, Currently executed trajectory $\sigma(t)$
Output: Command trajectory $\gamma(t)$, Safety library at current timestep Φ_{New}

1. $\Phi_{New} = \{\emptyset\}$.
2. $s_t = \sigma(t)$
3. **for** $\forall \phi_c \in \Phi(s_t)$
4. **if** $\Omega(\phi_c) \subseteq K_t / O_t$
5. $\Phi_{New} = \{\Phi_{New}, \{\phi_c\}\}$
6. **endif**
7. **endfor**
8. **if** $\Phi_{New} = \{\emptyset\}$
9. $\gamma(t) = \phi_e \in \Phi_{Previous}$
10. **else**
11. $\gamma(t) = \sigma(t)$
12. **endif**

The algorithm queries for emergency maneuver library at a future state of the rotorcraft, and ensures it can transition to an emergency maneuver which lies in known obstacle free space. If the there are no such maneuvers, one of the emergency maneuvers computed at the previous step(for the current state) are executed. This operation has a maximum limit on run-time and is guaranteed to keep the vehicle safe. The algorithm is explained with a working example in figure 4.

PLANNER ENSEMBLE AND TRAJECTORY EXECUTIVE

A trajectory planner ensemble is a collection of independent planners attempting to solve the same planning problem. The planners then submit their plans to the executive in an asynchronous fashion. These plans comprise the pool of trajectories from which the executive chooses. It is important to note that these plans no longer require a guarantee to keep the vehicle safe. In the worst case, if no trajectory is submitted the executive follows an emergency maneuver. However, the quality and abundance of plans determines the performance of the vehicle. Thus it is important that the planner ensemble is able to solve problems of varying difficulty.

The ensemble is an effective way of dealing with the problem of having no good measure which planning algorithm is appropriate in a given world configuration. By allowing planners to compete in parallel, the executive picks the overall best trajectory without having to reason about which planner to pick. We pick 3 diverse planning algorithms - a gradient based optimizer, RRT* and Anytime D*. We also use a trajectory library to tackle clutter in the pretouchdown phase.

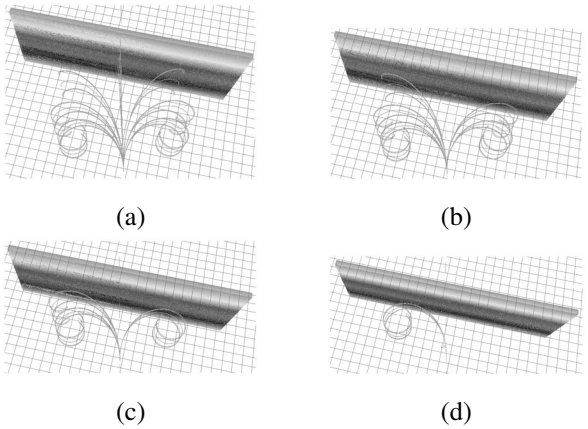


Fig. 4: Emergency maneuver library from data collected in flight test in Manassas, VA. (a) The helicopter approaches a large simulated wall with the emergency maneuver library. b) As the helicopter gets closer to the wall, the emergency maneuvers intersect the wall and become invalid. c) Fewer emergency maneuvers are left but some of them are still valid for future states d) An emergency maneuver is executed as the future state is no longer safe.

Covariant Gradient Descent

We pick CHOMP (Ratliff et al (Ref. 18)) as the basis for our trajectory optimizer. CHOMP is a covariant gradient descent based algorithm. While normal gradient descent creates a perturbation $\Delta\sigma$ such that the step is small in the sense of $\Delta\sigma^T\Delta\sigma < \varepsilon$, a covariant gradient descent choses $\Delta\sigma$ such $\Delta\sigma^T D\Delta\sigma < \varepsilon$ where D is a distance metric. This metric can specify weights on higher derivatives of $\Delta\sigma$ such that the gradient step is smooth. Then the gradient step at iteration k becomes

$$\sigma_{k+1} = \sigma_k - \frac{1}{\lambda} D^{-1} g_k$$

where σ_{k+1} is the new trajectory, σ_k is the old trajectory, λ is a lagrange multiplier and g_k is the gradient. This distribution is akin to the Newton Raphson Steepest Descent gradient distribution. Fig. 5. illustrates how the covariant gradient affects the trajectory.

An analytical dubin's visibility graph is used to compute an initial guess which avoids NFZs and reaches the goal. The time to collision metric is used to derive gradients which are then distributed to get a smooth update. Even though the plan produced is smooth upto $\ddot{\sigma}$, it relies on the dynamics filter for non-linear dynamics enforcement.

The algorithm favours cases where the initial guess needs to be perturbed slightly to stay clear of obstacles. For an autonomous helicopter flying in mostly a sparse environment,

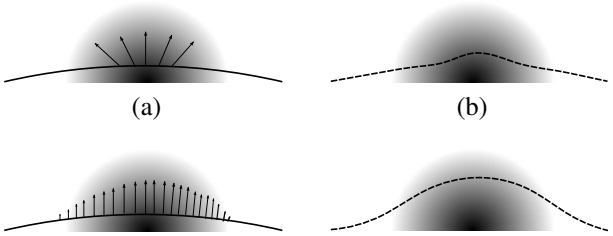


Fig. 5: Advantage of taking a covariant gradient. (a) Gradient along a trajectory point along directions of minima (b) Nominal gradient descent makes a small perturbation in trajectory space. This leads to a small, non-smooth change. Several iterations of this would be required to avoid the minima (c) Covaraint gradient makes a small step in derivative space of the trajectory. This translates to a distribution of the gradient which preserves smoothness (d) A single iteration takes the trajectory out of collision while preserving smoothness.

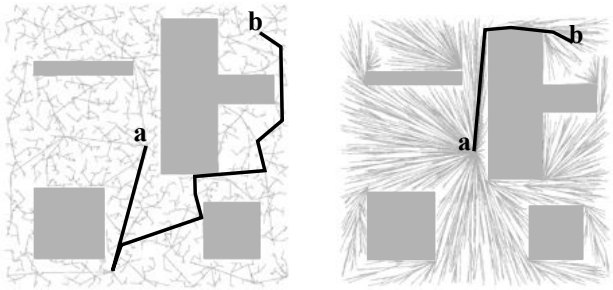


Fig. 6: Improvement of the RRT* tree over RRT. (a) RRT does not repair the tree, leading to answers with no bound on quality (b) RRT* repairs the tree leading to an asymptotically optimal trajectory.

this proves to be the most practical approach. However it has a tendency to get stuck in a bad local minima. This usually the case if the trajectory is passing through an obstacle that has a flat or non-convex outer profile. A typical scenario is flying between a tree line on approach such that it can neither fly over, nor can it escape out of the local minima.

RRT*

The RRT* algorithm (Karaman and Frazolli (Ref. 8)) is an asymptotically optimal sample based planning algorithm. The algorithm grows a search tree by sampling a point in free space, connecting it to the optimal parent in the tree and then rewiring the tree. This has an enormous difference to RRTs as shown in Fig. 6. We generalized this approach to RRT*-AR (Ref. 2) (Choudhury et al.) that can return a set of diverse trajectories. It does so in a rapid fashion by injecting variation in the search tree and using approximation of cost bounds to get large speedups. The RRT*-AR uses a dubin's primitive as a steering function due to its analytic form. It, like the optimizer, run the dynamics filter as a post processing step.

The RRT*-AR generates a random tree in the state space.

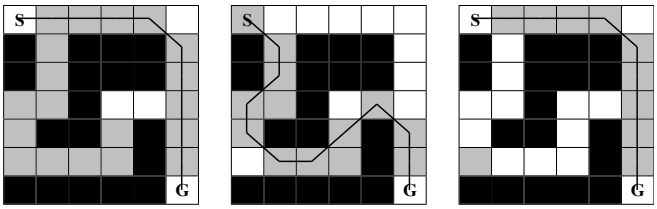


Fig. 7: Anytime D* gives interim suboptimal paths at almost no extra computation (a) The full A* expands 19 cells and has the optimal path (b) AD* with $\varepsilon = 2.5$ gives a path after only 13 expansions (c) Continuing AD* to $\varepsilon = 1.0$ gives a path after 10 expansions. In total the AD* expands 23 cells but is able to provide paths at anytime interim.

Due to this inherent stochasticity, it becomes very difficulty to predict performance and coverage statistics for this planner and is an important topic for future work. However, the ability of the algorithm to jump across the state space and repair the search tree allows it to reach a solution (albeit sub-optimal in quality) very quickly. Empirically, we observed that this planner was more robust in a given time scale than other planner and thus proves to be a vital component in the ensemble. For it to be able to generate high quality trajectories, we applied methods such as warping the search space around the analytic dubin's path and using importance sampling (Kobilarov (Ref. 10)).

Anytime D*

Likhachev et al. (Ref. 13) proposed an anytime version of a discrete optimal graph search. The algorithm quickly finds a suboptimal plan by inflating the search heuristic and then proceeds to repair the path. Thus at anytime, it has a suboptimal path available. Fig. 7. shows the concept behind repairing graphs. This algorithm is used over a custom state lattice in this paper.

This algorithm comes with a lot of guarantees in terms of optimality and complexity. However the representation of a graph becomes very crucial as the search does not scale well in dimension. A state lattice accommodates the non-linear constraints into its construction but to get similar run time performance as the other algorithms, a lot of design choices have to be made. In this paper, we use the standard state lattice version of the algorithm and note that this is considerably slower than the other algorithms.

Trajectory Library

Trajectory libraries (Frazzoli et al. (Ref. 5)) are popular because they offload the nonlinear constraint matching part to an offline optimizer and restrict the optimal solution to lie in the basis of the library. Libraries are pretty efficient to use when the underlying dynamics are too constrained to be solved online and having a trajectory restricted to the basis is an acceptable compromise.

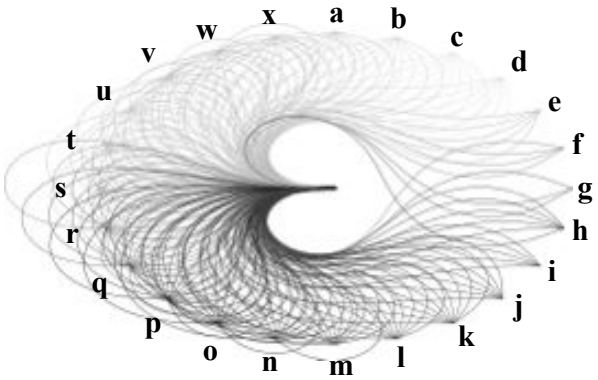


Fig. 8: The end game library unit circle. The unit circle is composed of analytical polynomials in curvature space. By concatenating such circles recursively, vast pool of trajectories can be synthesized.

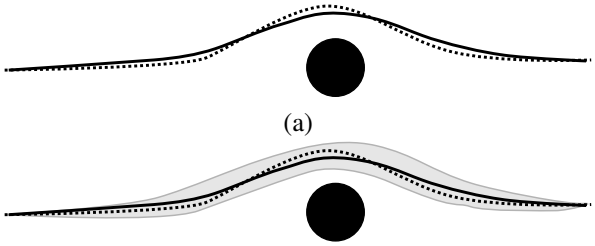


Fig. 9: The dynamics filter creates a feasible trajectory within a bounded volume of infeasible trajectory (a) The planner ensemble plans an infeasible path (dotted lines) while the filter outputs a conservative feasible path (b) The feasible path is guaranteed to stay within a funnel around the infeasible.

Such a case occurs during the pretouchdown phase of a helicopter. It is often the case where the mode of the dynamics undergo a high speed to low speed transition making the optimal trajectory more difficult to compute. Since the touchdown zone is known apriori, the search space is a bounded static area. This allows one to apply rapid collision checks by leveraging a mapping from $\{x,y,z\} \rightarrow \sigma_i$ where σ_i are part of a trajectory collection.

We define this library as the end game library which can provide high quality solutions to cluttered landing zones. In addition to rapid collision checks, this library provides an easy offline way to enforce adhoc close to touchdown rules that a helicopter should obey. Fig. 8. provides an illustration of how the library is grown.

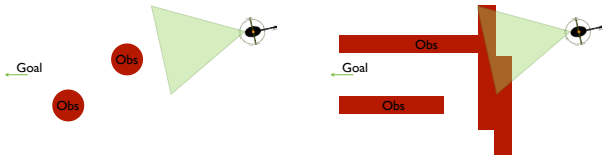
Dynamics Filter

The purpose of this module is to check and enforce dynamic feasibility on a plan. It does so by creating a fictitious vehicle model out of the constraints and then tracks the input plan. The traced out trajectory is dynamically feasible by construction and passed along to the executive. An illustration of this is shown in Fig. 9.

Algorithm 2 gives an overview of the dynamics filter. The key requirement is that the input trajectory has bounded

Algorithm 2: Dynamics Filter
Input: Initial Guess $\sigma_{initial}(t)$, Constraints $g(.)$
Output: Dynamically feasible trajectory $\sigma(t)$

1. Based on constraints $g(.)$, construct a model $\dot{\Omega} = \mathcal{F}(\Omega,u)$ where u is the set of highest order terms which have constraints. For example in high speed mode, $u = \begin{bmatrix} a_{long} & \dot{\beta} & \ddot{z} & \dot{\phi} \end{bmatrix}$. Saturation is applied to Ω and u based on $g(.)$. The trajectory of $\Omega(t)$ is $\sigma_\Omega(t)$.
2. Construct a policy $u = K(\Omega,\sigma)$ and the control lyapunov function $V(\Omega,u,\sigma)$ using backstepping on the cascaded system Ω.
3. Let $\sigma_{initial} = \bigcup_{i=0}^{n} \sigma_{i,i'} \cup \sigma_{i',i+1}$ where $\sigma_{i,i'}$ is infeasible and $\sigma_{i',i+1}$ is feasible.
4. In the infeasible section, $|V_{i'} - V_i| < M \left\|\sigma_{i,i'}\right\|_{BV}$ where $\|\sigma\|_{BV}$ is the bounded variation of a trajectory segment
5. In the feasible section, $V_{i+1} < V_{i'}$
6. Hence the total variation of the feasible around the infeasible is bounded by $\Sigma_{i=0}^{n} M \left\|\sigma_{i,i'}\right\|_{BV}$
7. The trajectory returned $\sigma_\Omega(t)$ is dynamically feasible by construction.



(a) (b)

Fig. 10: Simple benchmark scenario for planning from start to goal (a) Two poles (b) Four walls

lengths of constraint violation which limit $\Sigma_{i=0}^{n} M \left\|\sigma_{i,i'}\right\|_{BV}$. The Control Lyapunov function of the fictitous model then converges on the feasible sections and diverges by a bounded amount in the infeasible sections.

SIMULATION EXPERIMENTS

In this section, we discuss the set of experiments performed in simulation to design and verify the planning system. We will start with a generic experiment with planners and then move on to simulations with the actual system that is to be flown.

Simplified scenarios

The purpose of this experiment is to get a set of empirical results in an artificially created benchmark scenario to gain an understanding of performance of planning algorithms and the role of an executive. In Fig. 10. two benchmark scenarios are shown. The objective is to plan from start to

Table 3: Planner selection statistics averaged for 100 trials

Algorithm	Two poles	Four walls
Optimizer	87.25%	13.61%
RRT*-AR	12.75%	76.19%
AD*	0.00%	3.40%

Table 4: Executive statistics averaged for 100 trials

Criteria	Two poles	Four walls
Success rate with single planner	83.00%	5.00%
Success rate with executive	100.0%	100.0%
Average switches to alternate	0.7	3.8
Maximum alternate routes in a run	3	7

goal given the dynamics and sensor constraints of the system. The vehicle has a maximum speed $v_{max} = 25m/s$, roll limit $\phi_{max} = 45\,\text{deg}$, roll rate limit $\dot{\phi}_{max} = 22.5\,\text{deg}/s$ and vertical speed limit $v_{vert} = 2.5m/s$. It has to reach the goal 500m away at the illustrated heading without slowing down. The mounted laser sensor has a range of $150m$, field of view 100 deg nodding passively at a speed of $180\,\text{deg}/s$. These set of constraints created a difficult scenario where planning time had a large effect on system performance. The two scenarios differ in the obstacle distribution. In one of them, there are two poles between the start and the goal. In the other, there is a sequence of four walls which will eventually force the planner to circle around and find the only opening.

Table 3 shows the statistics for plans being selected. As expected the smooth gradients in the case of pole scenario allows the optimizer to dominate. In the wall scenario, the large policy change that is required is computed by the RRT*-AR which becomes the dominant planner. The case of AD* can be explained as purely a computational drawback. As the implementation involved a simple state lattice and a simple heuristic, it could neither produce a path quick enough in the 4 dimensional space, nor did it have the quality as the other planners due to discretization oscillations. We would like to refer to implementations such as MacAllister et al. (Ref. 14) which address such problems. However, based on these statistics we chose to stop development on the AD* planner and focus our investigation of the other planners.

Table 4 shows the executive usage statistics. Because the executive ensures safety it always succeeds in keeping the vehicle safe and reaching the goal. In contrast the single planner suffers because it is not keeping the vehicle safe while planning. The scenarios are also indicative of the alternate route usage and diversity. Because the walls require diverse policy changes, upto 7 alternates are maintained. The average switching indicates how many times alternates had to be invoked because none of the current planners had a plan. Without the executive in such a situation maintaining alternates, a

Fig. 11: A mission to land at a helipad in the Grand Canyon. The planner is not given a prior map and has to follow the canyon as it senses it.

Table 5: Planner selection statistics for grand canyon

Algorithm	Selection
Optimizer	65.82%
RRT*-AR	34.18%

mission failure would occur. For example in the wall case, on an average 3.8 times an alternate route was used.

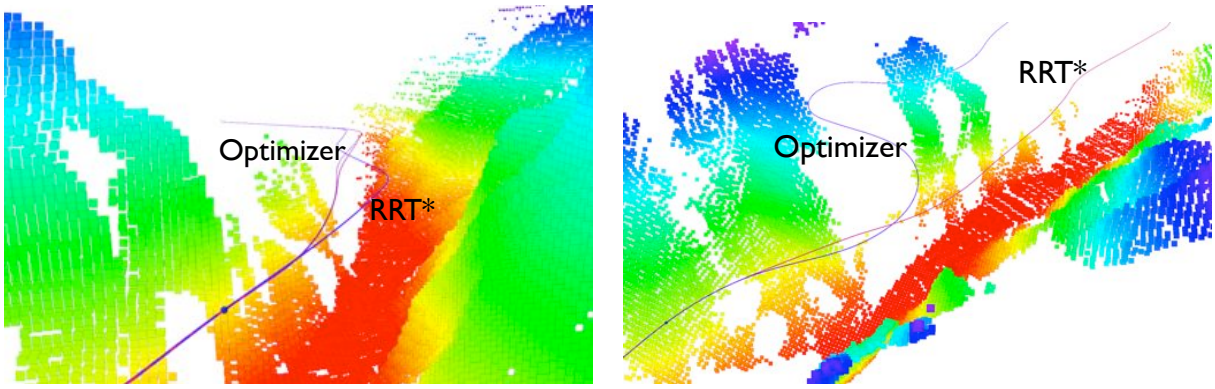
Landing in the Grand Canyon

We move on to simulation experiments with the actual system to be flown. To stress test the obstacle avoidance and characterize performance, we ran mission in the grand canyon as shown in Fig. 11. The mission speed is set at $30m/s$. Table 5 shows that RRT*-AR was selected more often than for nominal cases (34.18%). This is reflective of the fact that even though the grand canyon has a clear optimal path, the sensor limitations often encourage planners to "shortcut" across the canyon only to later detect that there is an obstruction.

Fig. 12(a). shows the standard scenario of the optimizer computing an optimal trajectory while the RRT*-AR does an oscillation as an artifact of the random sampling. However as the optimizer tries to cut across unknown space it gets trapped in a local minima on discovering obstacles Fig. 12(b). The RRT*-AR on the other hand is immune to such minima and plans around the obstacle.

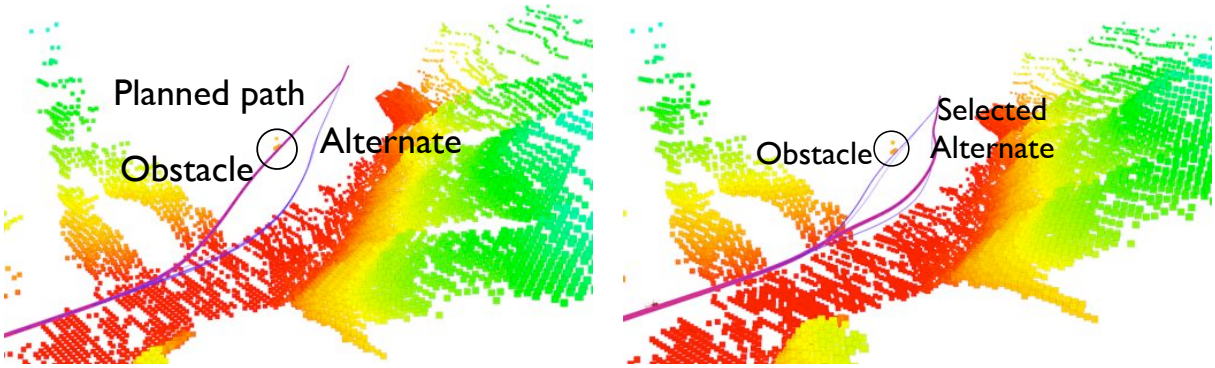
The grand canyon is an environment where many obstacles lie in occlusion of others. It provides situations where obstacles appear late in the field of view of the laser. Fig. 13(a) is one such example. The obstacle invalidates the current plan. We can see that the executive is maintaining diverse alternates and thus has one that can avoid the obstacle. In the next time step Fig. 13(b), the alternate is chosen. This feature is critical to performance as otherwise the delay could trigger the emergency maneuver behaviour.

Fig. 14. shows an application of the emergency library. In this situation, RRT*-AR is not being used as a result of which the chosen path is the optimizer path. Since it is guaranteed safe, the executive proceeds along it while retaining trajectories from the library. As soon as the future point is no longer



(a) (b)

Fig. 12: Both RRT*-AR and Optimizer dominate interchangeably in the Grand Canyon mission (a) Optimizer dominates here because the gradients from the edges push it into a global minima (b) Optimizer gets trapped in a bad local minima in a space between two obstacles. RRT*-AR plans around the obstacle.



(a) (b)

Fig. 13: Executive switches to alternate routes when an obstacle enters the field of view at a very late stage (a) The instant when the obstacle appears and invalidates the planned path. There are already alternates in the executive avoiding the obstacle (b) In the next time step (0.1s) later, the executive switches to the alternate.

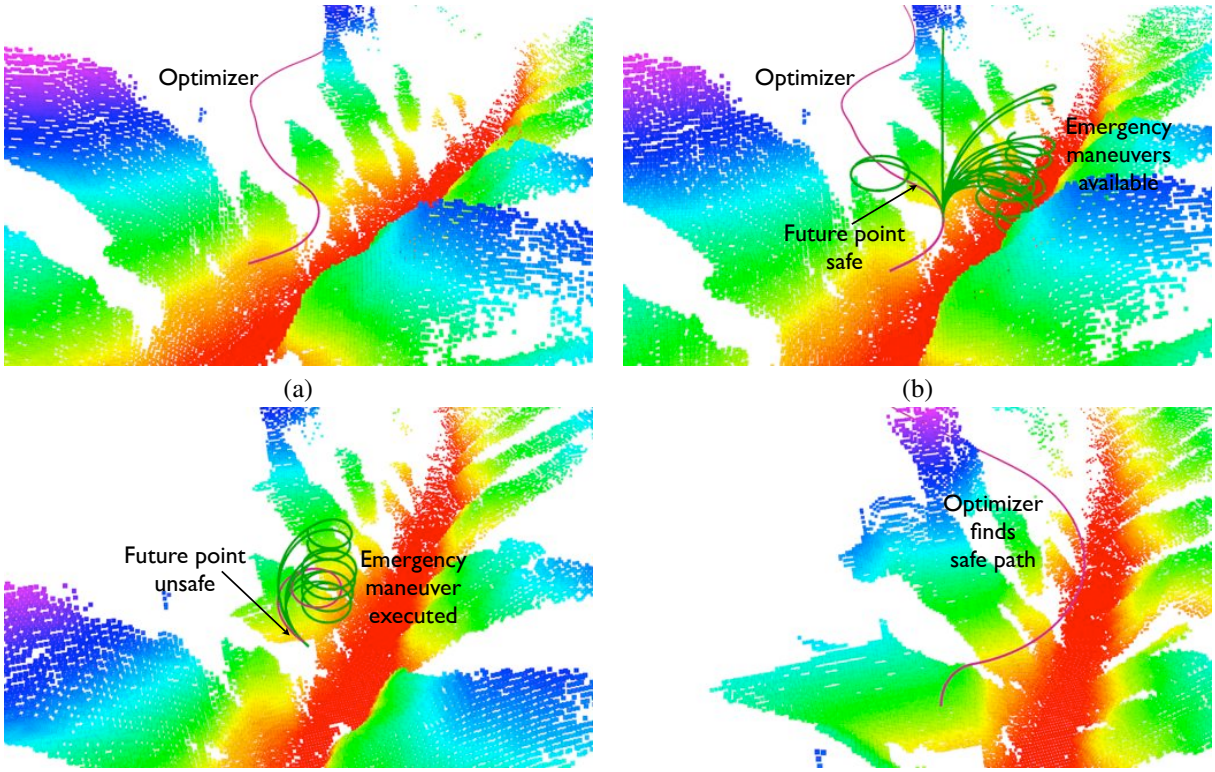


Fig. 14: Emergency maneuver takes the system out of an undesired configuration. (a) Since we run the experiment without an RRT*-AR, the selected path is the optimizer path. This lies in a local minima between two walls of the grand canyon (b) The system follows this path since the future point is safe (c) The future point is no longer safe and an emergency maneuver is selected (d) The optimizer is now in a configuration where it can find a safe path again.

Table 6: Planner selection statistics for SanDiego

Algorithm	Selection
Optimizer	29.17%
RRT*-AR	70.83%

safe, the executive overrides the current plan with the maneuver. As the maneuver is being executed, the optimizer finds it in a state from where the initial guess can lie in a safe area. Thus the executive restores normal status.

Benchmark

The obstacle field navigation benchmark (Mettler et al. (Ref. 15)) is a popularized standard for evaluating systems in different standardized scenarios. The benchmark aims to evaluate speed, energy consumption and safety of the system in these scenarios. However since we had no control over the controller and dynamics which are that of a large helicopter, we evaluated the benchmark as follows. Each mission was scaled up in size by 10 times. The helicopter then follows the mission at a speed of $30m/s$ subject to its original dynamics. Following this the metrics are scaled down by 10 and compared to the baseline. All 6 scenarios are evaluated along with some missions from sandiego. Since the helicopter acceleration limits are very conservative ($0.981m/s^2$) the missions were started with the helicopter in a trim state at 30m/s and was commanded to maintain that speed and loiter at the end. This only increased the difficulty of the system.

Fig. 15. shows the results for the Cube Baffle mission. The conservative dynamics are apparent from the angular rate difference. Given that limitation, the system stays safer and reaches the goal in a comparable time. For the other 5 scenarios, we found similar results that the optimizer alone was enough to solve the problem.

The San Diego missions were completely contrary to this as seen from the statistics in Table 6 where the RRT*-AR dominates for more than 70% of the time. Given the limited range of the sensor, situations would often arise where a building is observed to obstruct the optimizers initial guess. In easier cases, as shown in Fig. 16(a), the path is clipping a corner of a building. The gradient in this case is well behaved and pushes out the trajectory. The RRT*-AR computes trajectories around the cost valley but not as precise as the optimum. However the more common case is Fig. 16(b) where the optimizer is trapped in a local minima when it passes through the middle of the building. Notice that as the evidence grid develops, it is likely to leave "holes" acting as perfect local minima traps for the optimizer. The RRT*-AR avoids the building and takes the system to a state from where the optimizer becomes dominant. This interplay between the planners is observed throughout the SanDiego mission.

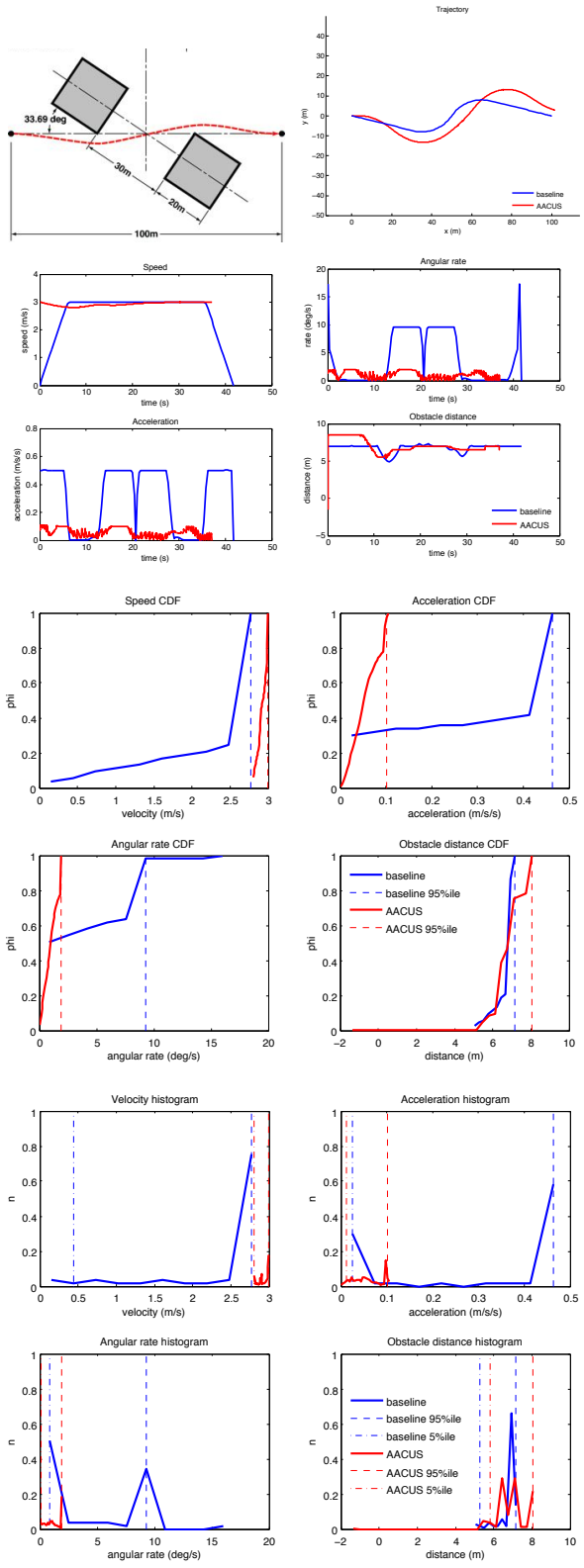
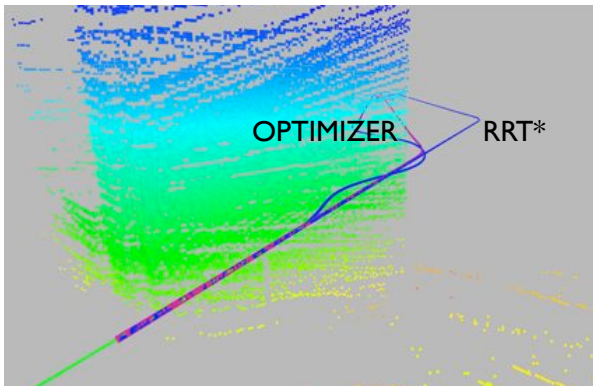


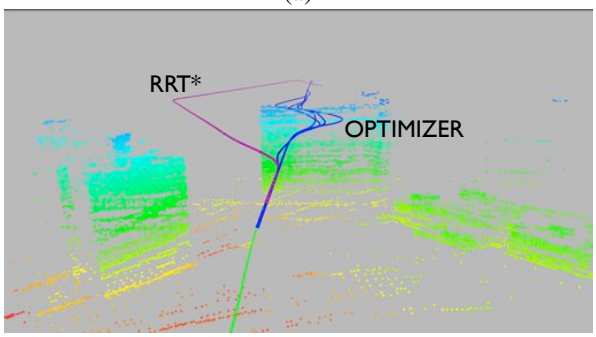
Fig. 15: The Cube Baffle benchmark from the obstacle field navigation set. The original helicopter dynamics and sensor range are scaled down by 10 preserving relative ratios. As a result the system is more conservative but can solve the problem in realtime with only the optimizer.

FLLIGHT TEST EXPERIMENTS

The planning system has been stress tested on an autonomous helicopter over a period of months. Missions have ranged



(a)



(b)

Fig. 16: Planners selected interchangeably in the SanDiego mission scenario. (a) The optimizer follows a nice gradient around the building while the RRT*-AR computes paths that miss the optimum valley (b) The more familiar scenario of the optimizer getting trapped in a local minima inside a building. The RRT*-AR circumvents the obstacle.

from avoiding no fly zones, terrain obstacles and landing straight to ground while avoiding enroute trees, facing the wind direction and dealing with clutter at the landing zone. In this paper, we focus only on the trajectory planning results that show how the system deals with unmapped obstacles.

Fig. 17 and Fig. 18 shows a collection of missions flown at Quantico, VA and Mesa, AZ. The two locations had very different terrains and the system adapted accordingly. Table 7 shows statistics for the Quantico mission. Overall the mission involves very little interaction with obstacles and hence

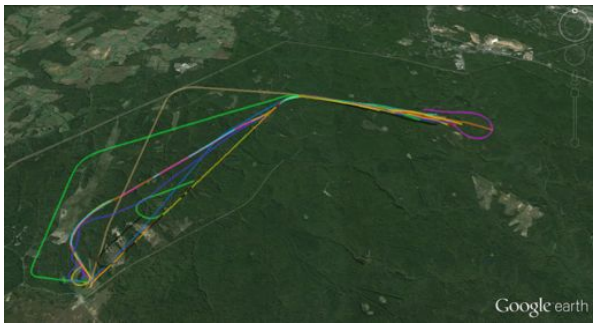


Fig. 17: Collections of missions flown in a day at Quantico, VA. The missions were permutations of landing, wave off, different wind directions, different popup NFZs.

Table 7: Planner statistics averaged over 8 flights in Quantico

Statistics	Values
Maximum speed	52.2 m/s
Speed at tree line	10 m/s
Min time to collision to tree	10 s
Max RRT*-AR usage	0
Max alternate usage	3

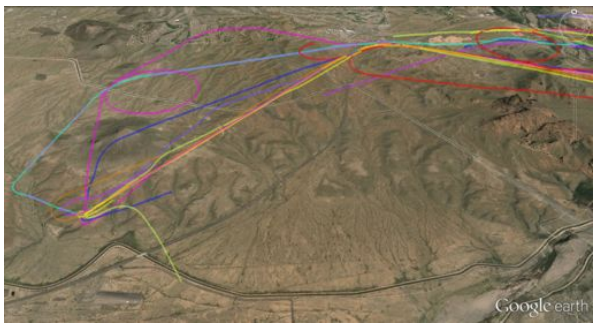


Fig. 18: Collection of missions flown at Mesa, AZ over a few flight days. The terrain is very different from Quantico. Similar permutations were flown here, the difference being that terrain avoidance was required in several cases.

the optimizer alone is good enough. The only interesting section is the intersection with trees. The original mission is always intersecting with a tree which is occluded and hence is seen late. However, since the executive maintains alternates it seamlessly switches trajectories. Table 8 shows statistics for the Mesa mission. This mission involved more interaction with terrain leading to situations where RRT*-AR and alternates were used.

Flight around a mountain

Fig. 19. shows obstacle avoidance of a mountain while flying at 30 m/s in Mesa, AZ. In this scenario the helicopter was following a mission to loiter behind the mountain. The obstacle is detected and a trajectory is planned such that the minimum time to collision is 5.0*s*.

Fig. 19 (c) shows the time to collision heat map and resultant gradient. Given the configuration of the obstacle, this gradient direction is well defined. Fig. 19 (d) shows the covariant gradient descent undertaken by the optimizer. This can be seen by the large perturbations in the trajectory while maintaining a smooth profile. This is similar to the illustration

Table 8: Planner statistics averaged over 5 flights at Mesa

Statistics	Values
Maximum speed	56 m/s
Max speed near mountain	30 m/s
Min time to collision to mountain	5.0 s
Max RRT*-AR usage	2
Max alternate usage	1

shown in Fig. 5. Fig. 19 (e) shows the optimizer reducing the cost with iterations, while the RRT*-AR lies on either side of the cost valley. The jump to the minimum by the optimizer indicates the rapid effect of covariant descent. The RRT*-AR in this case cannot compete with the optimizer as shown in Fig. 19(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 time optimal. There are rare cases where the optimal path in the tree is on the other side of the cost valley and closer to the obstacle. The RRT*-AR however does give a nice variance to the pool of alternates in the executive if the optimized plan is invalidated later on.

Flight between NFZ and a mountain

Fig. 20 shows a scenario where the planner has to plan between a known NFZ and an unmapped mountain at 30 m/s in Mesa, AZ. In this scenario the planner is going to land at a point behind the NFZ.

This case is very different from the previous experiment around the mountain. The NFZ is treated here as a constraint and thus has no gradients outside of it. This is because it is difficult to associate a cost function with being outside the NFZ as it will be a fictitious cost function and may dominate the real time to collision cost. The gradient from the mountain, even though it has a well defined direction, pushes the trajectory against the NFZ. If the gradient descent step results in an iteration where the trajectory is inside the NFZ, it is rejected. As a result, after 4 iterations, the optimizer gets stuck in a local minima. Ideally the optimal path would require the trajectory to contort far before the mountain NFZ junction. Since the gradients do not percolate far back enough, this cannot be achieved. The optimizer computes a path with a critically low time to collision.

The RRT*-AR on the other hand samples enough vertices to pass safely through the gap. As seen from Fig. 20 (g). the cost has distinct stepwise decrements. This is indicative of the RRT*-AR repairing the branch passing through the junction. It finally finds a good enough solution that is safer than the optimizer.

Flying between trees in approach

Fig. 21 shows a flight in between trees at Quantico, VA. In this scenario, the planner is going to land beyond the trees.

This case is different due to how late the obstacle is discovered by the sensor. Fig 21 (b) shows the point cloud accumulated by the time the planner has already avoided the trees. The sensor planner first ensures safety of a trajectory by checking if emergency maneuvers are possible all the way to landing and then focuses its efforts on the landing zone. Hence the tree *below* the path in 21 (a) is not detected as shown by how little points landed in the encircled area in Fig 21 (b).

The executive has in its collection alternates collected from RRT*-AR and optimizer which vary around the optimal. As soon as the tree appears in the obstacle map, it invalidates the current plan shown in 21 (c). The executive switches to the alternate even before the optimizer has had a chance to react. The executive has a time budget 20 times faster than the optimizer making this an example where it played a very critical role.

Flying above trees in approach

As seen previously, trees are the most realistic challenge for a helicopter. They occlude the landing zone and intersect the mission glide slope. Climbing above the tree line is not always an option due to glide slope constraints. Sometimes a lateral avoidance is required.

Fig. 22. shows a scenario of climbing above the tree line at Flying Circus, VA. In this case, the sensor sees the tree line as it banks to approach the landing zone against the wind direction. On detecting the trees, it climbs in height. The climb is limited by the allowable glide slope. There is no lateral solution as the tree line exists on both sides.

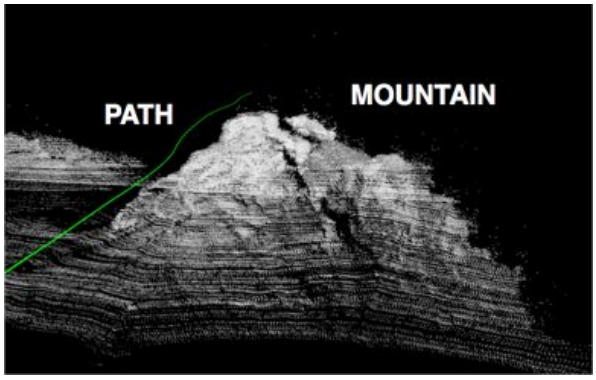
Fig. 23. shows a scenario where a single tree appears on the mission glide slope. This mission was repeated several times and the planner choses an avoidance direction dictated by the gradient. Like the previous situation, this problem is constrained by the glide slope and the landing direction and location.

It is important to note that the tree is a near to earth obstacle. From the point cloud itself, it is as much an obstacle as the ground to which the trajectory is being planned to. What distinguishes the cases is the time to collision because of different velocities. If the trajectory was passing through the ground at a similar speed of 10m/s, the trajectory would be optimized to rise above the ground to kill off the speed and then land.

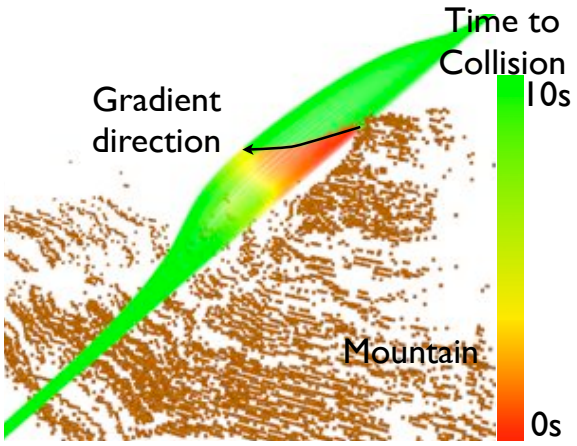
CONCLUSIONS

We have developed a planning system for autonomous helicopters that performs a wide range of near to earth missions dealing with a diverse set of environments. In our experiments, the autonomous helicopter is given a mission consisting of waypoints several kilometers apart with no prior map. The system avoided terrain such as mountains and trees as well as human defined popup NFZ at speeds ranging up to 56 m/s. To accomplish these results, our system uses an ensemble of planners running in parallel exploring the state space for a feasible solution. The heart of the system is a low latency verifiable module known as the trajectory executive which selects the plan from the ensemble, verifies it and maintains safety. Having a diverse collection of planners bidding plans solves the problem of which planning algorithm favours a given environment. The executive plays a crucial role of having redundant verifiable safety in the system. This system architecture allowed the system to be commanded to land at unprepared locations and allowed the system to correct human errors while designing the mission as a side result. Flight

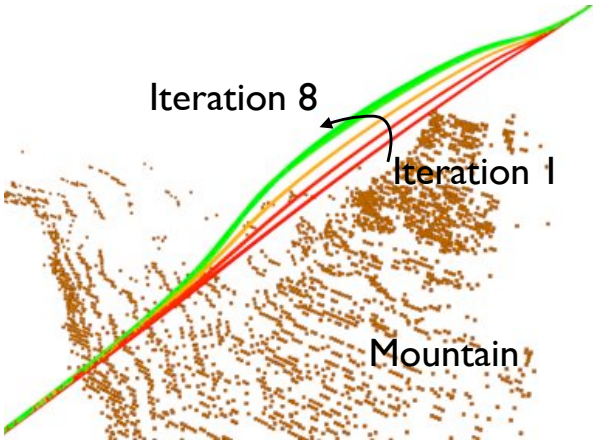
(a)



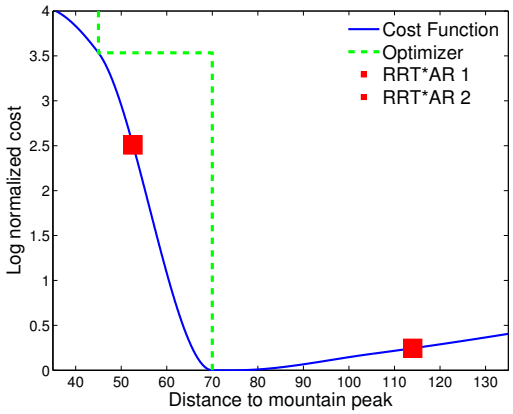
(b)



(c)



(d)



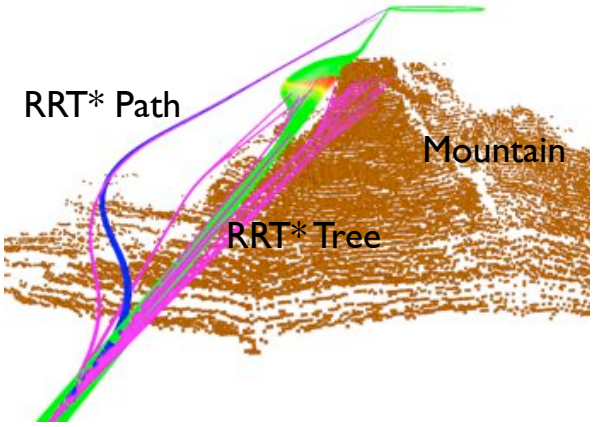


Fig. 19: Flying around an unmapped mountain in Mesa, AZ. (a) Safety pilot's view of the mountain (b) The sensor's view of the mountain (c) The gradient due to time to collision takes the trajectory away from the obstacle (d) Path for every one of the 8 iterations that leads to convergence to the optimal (e) The log normalized cost function valley with distance from the mountain peak. The optimizer descends into the valley within 8 iterations. The RRT*-AR computes alternate routes lying on either side of the valley. (f) The RRT*-AR optimal path avoids the mountain by a much larger distance than required. The RRT*-AR tree contains branches on either side of the cost valley.

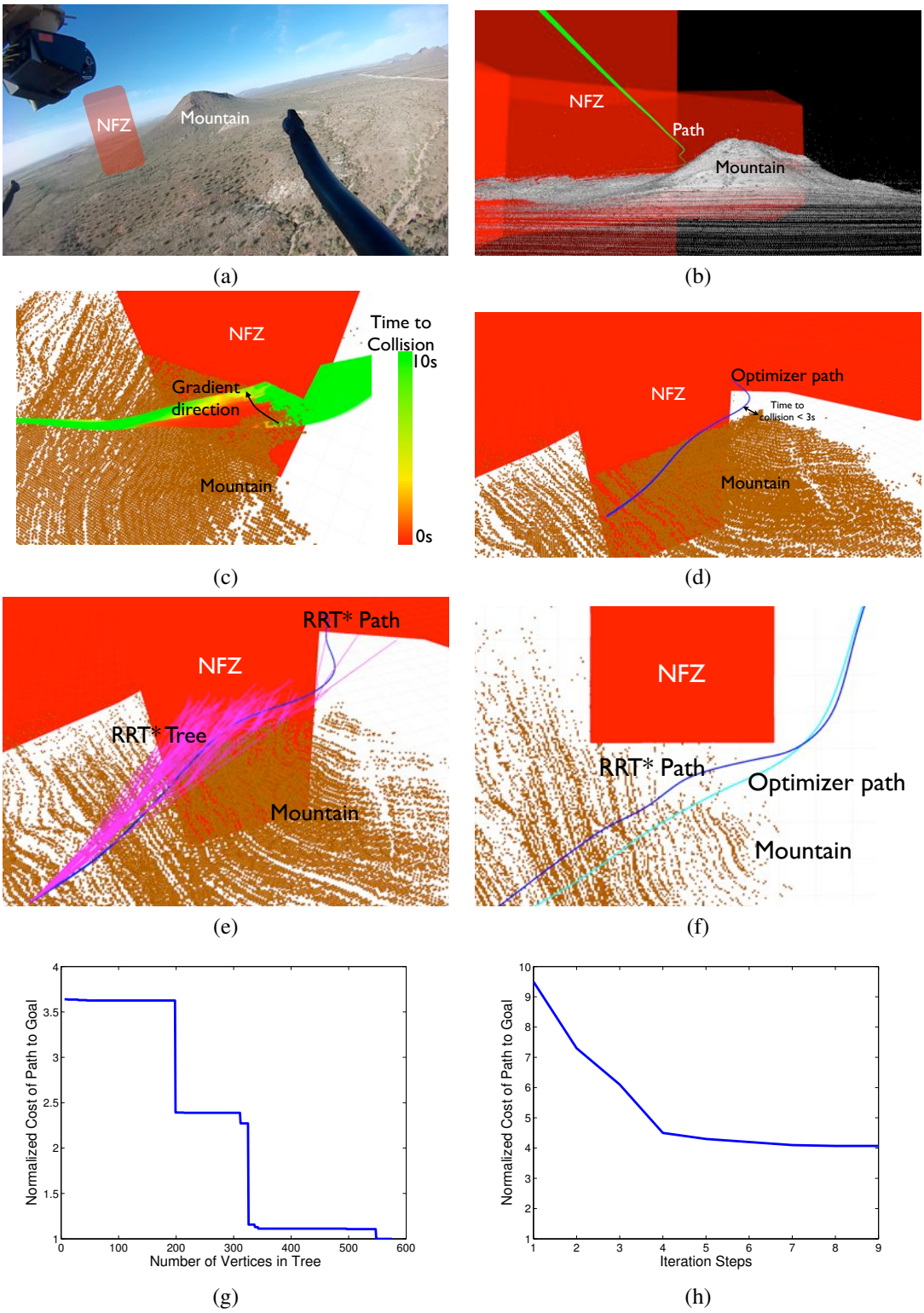


Fig. 20: Flying between a NFZ and an unmapped mountain in Mesa, AZ (a) The skid camera view of the scenario (b) The sensor's view of the situation (c) The gradient due to the time to collision pushes the trajectory into the forbidden NFZ. (d) The optimizer gets stuck in a local minima and has a critically low time to collision (e) The RRT*-AR tree is very diverse and contorts to find a near optimal trajectory (f) Comparison of the RRT*-AR trajectory to optimizer shows that RRT*-AR is safer (g) The best path in the RRT*-AR tree converges near optimal after sampling around 320 vertices. (h) The local optimizer cannot lower cost below a certain limit because perturbations at this point enter the NFZ.

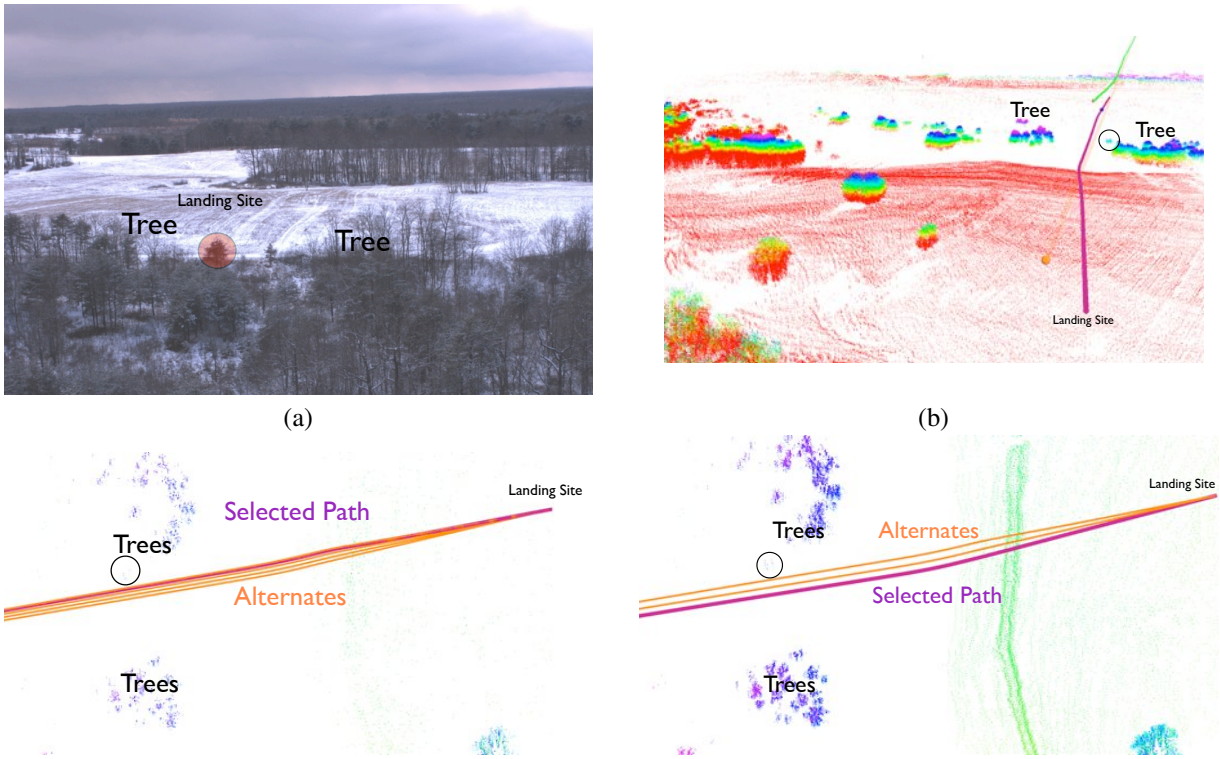
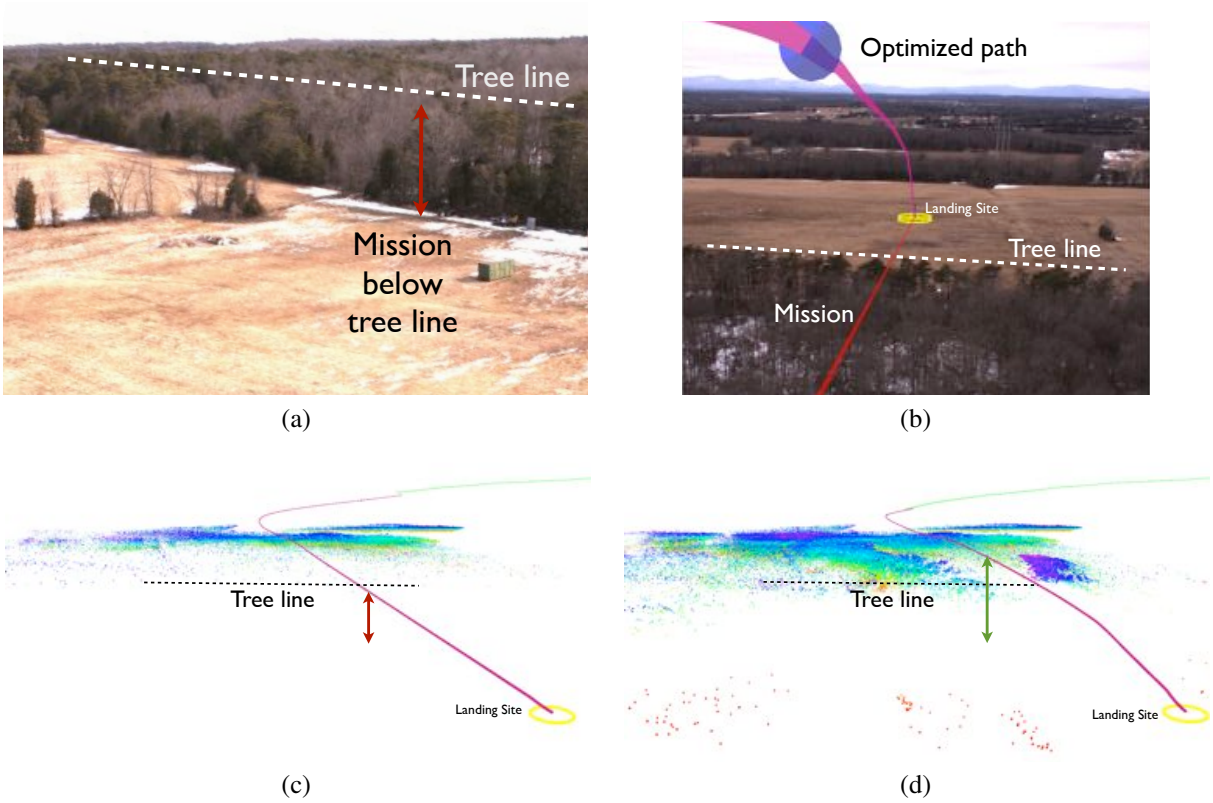


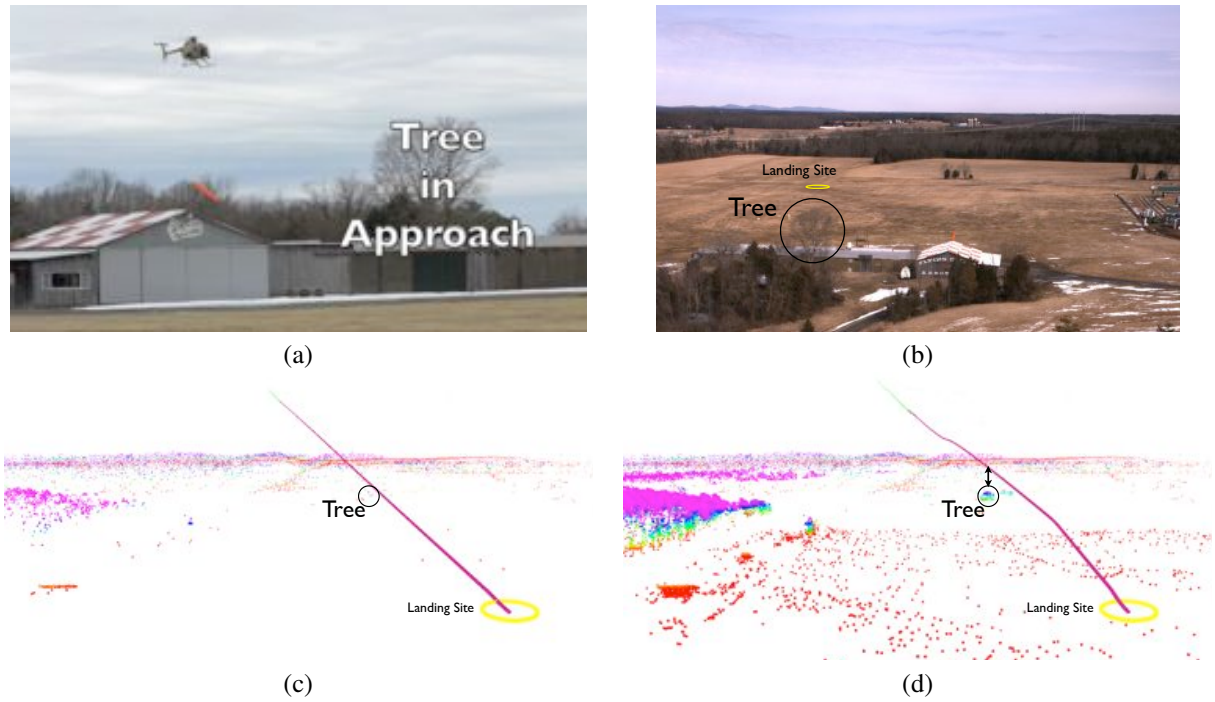
Fig. 21: Flight in between trees near the landing zone (a) The point of view from the onboard camera. The original mission intersects with a tree (highlighted) that is detected very late by the sensor. (b) The point cloud of the scene after the planner has already adjusted its path. The encircled obstacle is detected late but is critically near the mission path. (c) The tree is partially occluded and discovered very late in the approach. It invalidates the current trajectory. The executive has maintained a list of alternates (d) The executive switches to an alternate in the next time step.



(a) (b)

(c) (d)

Fig. 22: Flying over a high tree line near the landing zone at Flying Circus, VA (a) The mission glide slope is such that it is well below the tree line. (b) The point of view from onboard sensor. The mission goes through the trees while the optimizes path climbs over. (c) The original planned path when the tree line had not been discovered by the sensor (d) On discovering the tree line, the optimized path goes over.



(a) (b) (c) (d)

Fig. 23: Flying over a tree in approach at at Flying Circus, VA (a) The helicopter avoids a tree on its way to land (b) The point of view from onboard sensor. The tree invalidates the mission to land (c) The tree enters range but has not yet appeared in the obstacle map. The planned path goes through the tree (d) The planned path flies above the tree as soon as its detected. There is also a lateral deviation.

tests in Manassas, Mesa and Quantico extending a period of months validates the architecture.

We intend to address certain issues in future work. Running planner is parallel leads to redundant work done by each node. Even though the computational budget of the system was high, it can be more effectively used if planners do no repeat work done by each other. So a cooperative layer between algorithms will lead to a large improvement and allow the planners to work together to deal with more difficult cases. Secondly, the executive uses safety as a very reactive module. If safety was used with a larger horizon, the executive can select plans which have more tendency to be guaranteed safe in the future. This direction takes the system from being safe to pushing performance limits while still being safe.

ACKNOWLEDGEMENT

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.

REFERENCES

[1]S. Arora, S. Choudhury, and S. Scherer. A principled approach to enable safe and high performance maneuvers for autonomous rotorcraft. In *AHS Forum 70*, 2014.

[2]Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. Rrt*-ar: Sampling-based alternate routes planning with applications to autonomous emergency landing of a helicopter. Technical report, May 2013.

[3]D. Ferguson, M. Likhachev, and A. Stentz. A guide to heuristic-based path planning. In *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, 2005.

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

[5]Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *Robotics, IEEE Transactions on*, 21(6):1077–1091, 2005.

[6]C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent & Robotic Systems*, 57(1):65–100, 2010.

[7]C Goerzen and M Whalley. Minimal risk motion planning: a new planner for autonomous uavs in uncertain environment. In *AHS International Specialists? Meeting on Unmmaned Rotorcraft, Tempe, Arizona*, 2011.

[8]Sertac Karaman and Emilio Frazolli. Incremental sampling-based algorithms for optimal motion planning. In *Proc. Robotics: Science and Systems*, 2010.

[9]F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 2012.

[10]Marin Kobilarov. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871, 2012.

[11]Steve M. LaValle. *Planning algorithms*. Cambridge Univ Press, 2006.

[12]Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[13]Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *Proceedings of the international conference on automated planning and scheduling (ICAPS)*, pages 262–271, 2005.

[14]Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev. Path planning for non-circular micro aerial vehicles in constrained environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3933–3940. IEEE, 2013.

[15]B Mettler, Z Kong, C Goerzen, and M Whalley. Benchmarking of obstacle field navigation algorithms for autonomous helicopters. In *Proceedings of the American Helicopter Society 66th Annual Forum. Phoenix, AZ*, 2010.

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

[17]Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. In *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pages 802–807. IEEE, 1993.

[18]Nathan Ratliff, Matthew Zucker, J. Andrew (Drew) Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[19]Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *European control conference*, volume 1, pages 2603–2608. Citeseer, 2001.

[20]P. Tsenkov, J.K. Howlett, M. Whalley, G. Schulein, M. Takahashi, M.H. Rhinehart, and B. Mettler. A system for 3d autonomous rotorcraft navigation in urban environments. In *AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, HI*, 2008.

[21]M. Whalley, P. Tsenkov, M. Takahashi, G. Schulein, and G. Goerzen. Field-testing of a helicopter uav obstacle field navigation and landing system. In *65th Annual Forum of the American Helicopter Society, Grapevine, TX*, 2009.