Autonomous Emergency Landing of a Helicopter: Motion Planning with Hard Time-Constraints

Sanjiban Choudhury
sanjiban@cmu.edu
Graduate Research Assistant
Carnegie Mellon University
Pittsburgh, PA

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

Sanjiv Singh
ssingh@cmu.edu
Research Professor
Carnegie Mellon University
Pittsburgh, PA

ABSTRACT

Engine malfunctions during helicopter flight poses a large risk to pilot and crew. Without a quick and coordinated reaction, such situations lead to a complete loss of control. An autonomous landing system is capable of reacting quickly to regain control, however current emergency landing methods focus only on the offline generation of dynamically feasible trajectories while ignoring the more severe constraints faced while autonomously landing a real helicopter during an unplanned engine failure. We address the problem of autonomously landing a helicopter while considering a realistic context: hard time-constraints, challenging terrain, sensor limitations and availability of pilot contextual knowledge. We designed a planning system that deals with all these factors by being able to compute alternate routes (AR) in a rapid fashion. This paper presents an algorithm, RRT*-AR, building upon the optimal sampling-based algorithm RRT* to generate AR in realtime while maintaining optimality guarantees and examines its performance for simulated failures occurring in mountainous terrain. After over 4500 trials, RRT*-AR outperformed RRT* by providing the human 280% more options 67% faster on average. As a result, it provides a much wider safety margin for unaccounted disturbances, and a more secure environment for a pilot.

INTRODUCTION

Helicopters are in use for a variety of missions such as continuous surveillance, delivering goods in constrained environments and emergency rescue operations. Like all aerial vehicles, failures at any level can result in serious consequences finally leading to a crash. The loss of torque from the main rotor is one such common failure, occurring due to mechanical issues or fuel deficiency.

In such situations the pilot has to quickly execute a maneuver, known as autorotation, to land safely. For an autonomous system, precise control as well as choose a minimum risk landing zone within these time constraints becomes a very difficult task . As the vehicle descends at a fixed rate, the size of reachable landing zone exponentially decays. Since the vehicle cannot slow down to hover and is limited to a finite perception range, the planning time directly limits the number of options available to avoid an unmapped obstacle.

Research on this topic mainly focuses on the dynamical solution to autorotation. Optimal trajectories minimizing touchdown speed have been derived by Johnson (Ref. 1) and Lee et al. (Ref. 2), used to calculate control inputs by Aponso et al. (Ref. 3), and as pilot assistance during emergencies by Aponso et al. (Ref. 4) and Bachelder et al. (Ref. 5). The dynamics of flare have been a special focus for determining its feasibility by Tierney (Ref. 6) as well as in imitation learning by Abbeel et al. (Ref. 7) and model predictive control by Dalamagkidis et al. (Ref. 8). Recently, a complete feasible solution from glide to touchdown has been demonstrated (Yomchinda et al. (Ref. 9)). However, these approaches have not focussed on the issues of being real time in a planning problem with hard time-constraints, dealing with obstacles, limited sensor range and the effects of having a human in the loop. In this paper, we design a planning system that computes alternate routes (AR) as shown in Fig. 2 to address these issues thus maximizing the possibilities of a safe landing.

In summary, the main contributions are

- derivation of the need for alternate routes to increase the safety of the system,
- an algorithm to rapidly compute alternate routes and
- a multi-objective planning system to autonomously land a vehicle after engine failure.

PROBLEM FRAMEWORK

In this section, we take a look at the emergency landing planning problem that we wish to solve and identify the key features that make it challenging.

In Fig. 2, the different stages of emergency landing after engine failure at a nominal height is shown. Within 4 seconds of engine failure, the helicopter establishes autorotation - a trim state where the combination of forward and downward

Presented at the AHS 69th Annual Forum, Phoenix, Arizona, May 21–23, 2013. Copyright © 2013 by the American Helicopter Society International, Inc. All rights reserved.

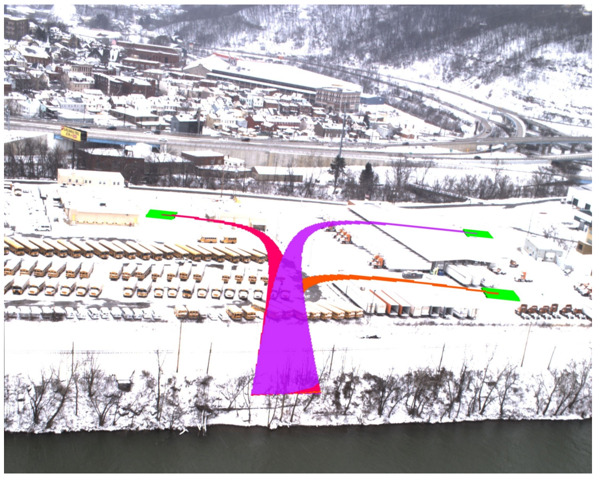


Fig. 1. Sample route set for safe landing after engine failure. This example shows a set of potential engine-out trajectories for a Bell 206 using the proposed RRT*-AR algorithm. The paths were calculated from a registered dataset from prior elevation maps, pointcloud, and image data collected onboard the aircraft.

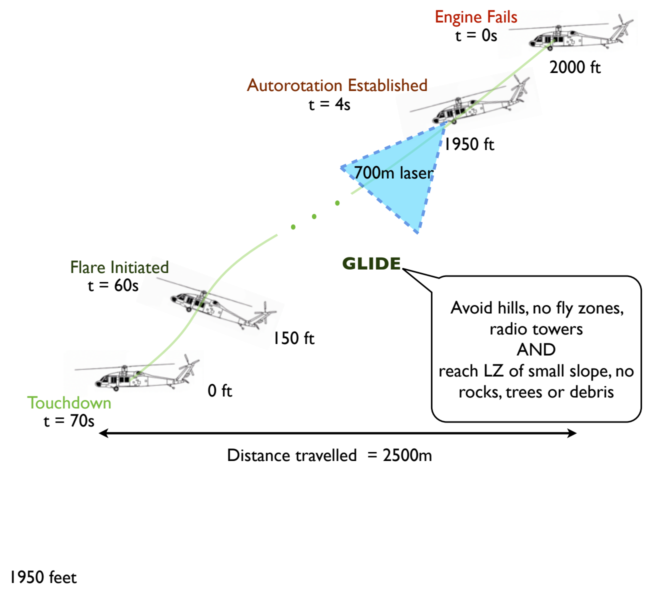


Fig. 2. Hard time-constraints of the emergency landing planning problem. On one hand, the vehicle descends almost 2000 feet in 60 seconds while flying over difficult terrain and aiming for a feasible landing zone (LZ). On the other hand, the perception limitations allow almost no time for re-planning in case of debris on the LZ or unmapped terrain obstacles.

velocity ensures that rotor energy is preserved. The helicopter glides to avoid terrain and man-made obstacles while aiming for a feasible landing zone. On reaching a certain height above the landing zone, the helicopter flares - it trades off the energy stored in the rotor to come to a smooth touchdown. The entire procedure lasts roughly 70 seconds.

The problem we wish to solve is to have a framework for an autonomous system that safely lands a helicopter after engine failure. We specifically target the glide stage of this procedure - the stage where the vehicle covers the most ground and spends the most time in. The following factors make the planning for this stage challenging:

Hard Time-Constraints

Autorotation landing is a very fast, one-shot process. Engine failure at a height of 2000 feet would leave around 60 seconds for the vehicle to touchdown. The vehicle does not have an option to hover as it waits for a plan to guide it to safety. Within these time constraints, it has to plan trajectories that are over 3 km long. The terrain can be arbitrarily complex with only patches of feasible landing areas.

In such situations, the planner needs to be able to find a path quickly and improve it as rapidly as possible. Since the feasible landing area of the helicopter reduces exponentially with time, it is important to guide the vehicle in the general direction of the optimum while continuing to refine the path. Information has to be retained and re-used across planning cycles, which is made possible by having a “life-long planner”.

Limited Perception

The perception system of the vehicle is a 3-D scanning LADAR with a range of up to 700 metres. Even though it is possible to acquire a detailed terrain and landing zone map a priori, a perception system is required for detecting unmapped obstacles and clutter on landing zones. Helicopters will have 17 seconds to clear an unmapped obstacle, and coupled with the dynamical constraints of this phase and the sensing period, the reaction time is less than 1.25 seconds.

In such cases, trajectories should make the fullest use of the perception system by flying over the landing zone before coming in to land such that any clutter in the landing zone can be detected well before-hand. When unmapped obstacles are detected, the planner must be able to switch to another safe plan instantaneously. This requires the maintenance of back up or alternative plans. The nature of the backups should be such that they are good candidates when the optimum is no longer safe.

Pilot Experience

The pilot can offer valuable experience on which trajectories are inherently safer to follow. Moreover, he/she is aware of the contextual information about which places are safe to fly over and thus is in a better position to make a choice. Given the severe time limitations, this mode of human computer interaction must be carefully designed. The options given to the pilot must be limited, distinct and helpful and contain candidates from interesting solution classes.

In summary, the desired attributes are

- A backup set of alternative routes,
- A limited set of route options which are easy to choose from, thus being helpful for pilots and
- A rapid, “life-long" planner.

ALTERNATE ROUTES AND REALTIME PLANNING

In the previous section, we introduced the key features of the emergency landing planning problem and how they motivate the nature of the planning approach. In this section, we advocate the use of alternate routes (AR) - a set of spatially different, locally optimal paths, as a powerful tool to address several of the afore-mentioned issues.

What is the relevance of Alternate Routes?

As we previously concluded, computing a set of alternate routes can lead to two main advantages - allowing the pilot-in-loop to make a contextual decision and having a safe alternate plan to switch to when the original plan is infeasible. We will now show that both these apparently different properties can be captured by the same optimization objective function.

Interesting set of path options for pilot - Consider the case of an engine failure occurring near the mountains as shown in Fig. 3. In such a scenario, it is not immediately obvious what the correct plan is. The first path is a risky path that goes through a mountainous region but reaches a safe flat area to land in by the coast. The second path stays clear of the mountain but aims for a risky landing in a clearing among the trees. There is very little room for error if the landing zone has clutter or rubbles. The third path lands in a similar clearing among the trees as the second path, however it has a good visibility of the landing site before it touches down. This can allow the sensor to detect clutter on the landing zone and have plenty of time to react if needed.

In such a situation, using the experience and contextual knowledge of the pilot is essential. By utilizing the pilot-in-loop, the system is able to offload the task of deciding “what is the right plan for the vehicle to follow in this context?” However, since the pilot has to make a decision very quickly, he can only be offered a very limited set of choices. Within this limited set, the planner must try to capture all classes of solutions the pilot might be interested in. Having two similar looking solutions in the set is ineffective. For example, in Fig. 3 these solution classes are “risky but good lz”, “safe but risky lz” and “good lz visibility”. Since each option is effectively

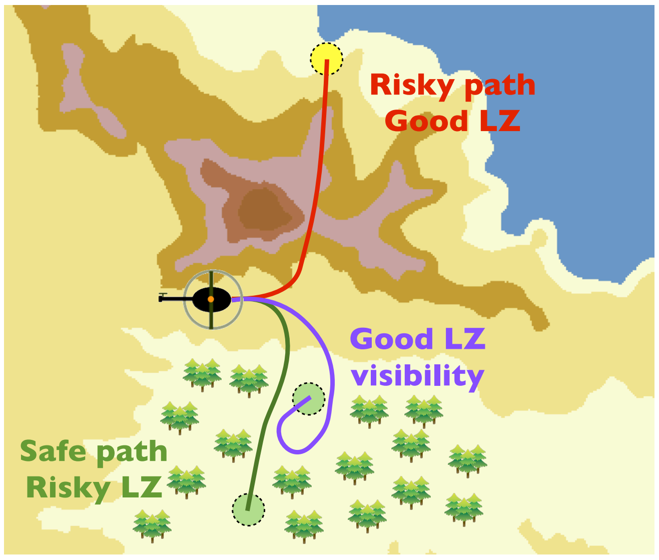


Fig. 3. A set of interesting options for the pilot consist of spatially separated, locally optimal solutions. Each path must be distinct and have a defining feature. “Risky path / Good LZ” comes close to the mountain but lands near the sea shore. “Safe path / Risky LZ” has a safe descent but lands near trees. “Good LZ visibility” has a safe descent while flying over the LZ and getting a good laser scan to check for debris.

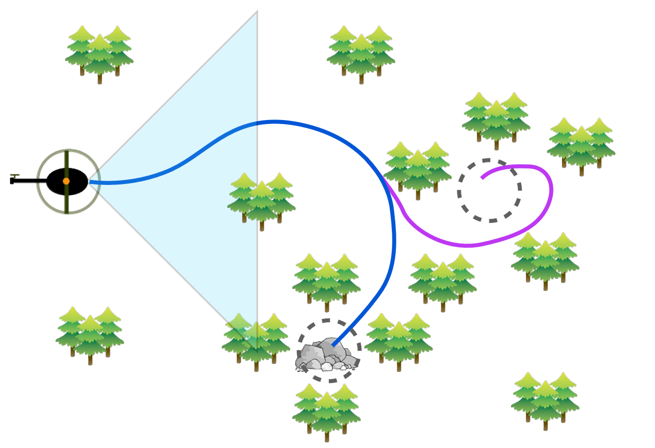
representing a solution class, it must be the optimal candidate for the solution class.

Thus the desired properties are - spatially separated and locally optimal solutions.

A set of emergency backup plans - Consider the case of the selected landing zone being covered in rubble, which is outside the range of the perception system as shown in Fig. 4 (a). If there were no backup plans, when the vehicle approaches the landing zone and senses the rubble, as shown in Fig. 4 (b), there isn’t enough time to re-plan a new path. However, if the planner had computed a set of backup paths, it would have an alternate path going to another landing site. It can switch to this path instantaneously as the rubble is detected.

A set of backup plans has some key characteristic features. Firstly, they should all be low cost feasible paths. Secondly, the set should be limited in size. The larger the set of backups, the more computationally intensive it is to go through each of the paths and verify which one to switch to. Thirdly, the different backups should be spatially different. If they are passing through the same volume, when one path becomes infeasible, they all become infeasible. For example, in Fig. 4, if the backups went to the same LZ, then the detection of the rubble would render all the backups ineffective.

Thus the desired properties are - cost of each path should be close to optimal and the paths should be spatially separated.


(a)

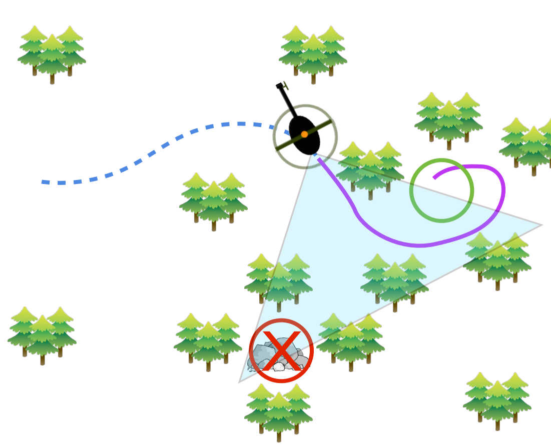

(b)

Fig. 4. Having a backup plan allows the helicopter to instantly switch when the current path is infeasible. (a) Helicopter hasn't yet detected the rubble at the LZ, but already has an alternative plan pre-computed. (b) Helicopter instantaneously switches plan when the rubble is detected.

How to compute AR?

To generate a set of alternate routes that satisfy the specified criteria, an appropriate optimization problem is required.

One such way to do it is to define multiple cost functions and solve each one separately. A typical greedy approach would be to solve for the original cost function. Then a new cost function can be created that penalizes proximity to the existing solution. The solution to this problem becomes the second member in the set. This process is iterated for the total number of members required in the set. This approach has many drawbacks. Firstly, it is very computationally expensive as redundant collision checks have to be repeated. Given the hard time-constraints, this is particularly unsuitable for our case. Secondly, since it is effectively planning afresh for every member of the set, it doesn't qualify to serve the role of a backup set generator. A backup set required computations of many plans in the same planning time as a single plan - this method offers no such facilities.

A very different approach to solving this problem is to sample from a distribution of paths. The parent path set is a pre-calculated set of paths up to the sensor horizon, and sampling is done by selecting a manageable sub set. Green Kelly et al. (Ref. 10) proposed a way of sampling paths to maximize "dispersion", i.e., iteratively add paths that are most distinct from the paths already added. Erickson et al. (Ref. 11) had a similar metric for sampling paths where the paths were selected according to "survivability", i.e, selecting a set that maximizes the expected survivability of a path considering a uniform distribution of obstacles. However these methods do not exactly address the same problem that we are proposing. Firstly, the methods reason about paths up to a finite horizon. Because we have a very detailed prior of the environment, our reasoning must extend all the way to the goal. At such large horizons, the methods are no longer tractable. Secondly, the methods only reason about collision avoidance and are agnostic to a cost function. In our case, the reasoning is based on optimizing a cost function which is dependent on factors other than just obstacles.

We want to frame an optimization problem that returns as a solution a set of answers, ensures all the solutions in the set have costs close to the optimal cost and are spatially distinct. Abraham et al. (Ref. 12) frame the same problem on a discrete graph to solve for alternate routes in road networks. Generalizing the problem to a continuous domain for emergency landing, we express it as

$$\begin{array}{rc} find: & \sigma_i = (x(t), u(t)) \quad \forall i = 1 \cdots m \\ minimize: & J(x(t), u(x,t), t_f) \\ constraints: & \dot{x} = f(x(t), u(x,t), t) \\ & x(0) = x_0, x(t_f) \in X_{LZ} \\ & g(x(t), u(t), t_f) \leq 0 \\ & \sigma_i \in \Sigma^* \end{array} \tag{1}$$

where $x(t)$ is the state of the vehicle and $u(t)$ the action, x_0 is the state at which engine failure occurs, X_{LZ} is the set of allowable landing zones, J is the cost function representing the desire to minimize risk, g is the set of environmental and vehicle constraints, σ_i is a path and Σ^* is the set of AR.

Alternate routes should have the following salient properties

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

2. σ_i is T-locally optimal (Local Optimality)

3. $J(\sigma_i)$ is $(1+\varepsilon)J(\sigma^*)$ (Uniform Bound Stretch)

where $S(.)$ is a swath around the path. Each of the above properties are expanded on below:

Limited sharing of new alternative and previous alternative path swath - A swath is define as the volume of space up to a radius r around the path. Limited sharing of swath implies that we want to minimize the amount of common sharing of volume by the alternate routes. This enforces them to be spatially separated. This is enforced by constraining only a fraction of the swath of any route in the set to be in common with the swath of any other route.

Local optimality - Local optimality, termed as T-locally optimal, ensures that every subpath is optimal. This checks for an obvious shortcut along the path which can lower the cost of the route.

Uniform Bound Stretch - This ensures that the cost of the path is within a bounded variation from the optimal cost. Thus the route has a bounded sub-optimal nature.

Does the optimization meet the desired characteristics?

Now we will examine if this optimization satisfies the original design criteria. We make the following observations about the optimization.

When solved at the moment of engine failure, the optimization creates an interesting set of path options for pilots The cost function is a concatenation of many smaller cost functions - avoiding obstacles, ensuring good landing zone visibility, penalizing curvature and landing zone feasibility. When engine failure occurs, the vehicle is at a considerable height above ground. At this height, the variety of possible solutions is large. Criteria 3 of the optimization problem ensures that all paths being considered are bounded in their sub-optimality. This means a particular routes excels in at least one of the criteria - either it is very safe, or has good LZ visibility and so on. Criteria 1 ensures that two paths near each other do not co-exist in the route set. This results in paths of different solution class in the final set. Finally, criteria 2 ensures that each candidate is locally optimal. Combining these criteria ensures that the pilot has an interesting set of representative options to choose from.

When solved at distances closer to the LZ, the optimization creates a reliable set of backup plans As the vehicle approaches the LZ, it has already committed to one of the solution classes that we mentioned before. The variety of paths from start to goal is not as large. Criteria 3 of the optimization problem does not play as interesting a role as before. It still ensures that all paths in the route have bounded sub-optimality. Criteria 1 plays the most critical role - ensuring that paths are spatially separated. This maximizes the survivability of a path - if the optimal path is found to pass through a "pop-up" obstacle, the chances that the obstacle affects other paths as well is minimized. This is a defining criteria for being a backup path. Criteria 2 enforces local optimality which is important as well in removing paths which have too many unnecessary diversions.

At this point, it is important to note that we had previously summarized the need for a rapid "life-long" planner. Now that this planner has to compute not only one route, but a set of routes, the emphasis on hard time-constraints is even more emphasized. In the coming sections, we will introduce our approach to solving the optimization in a computationally efficient way.

EMERGENCY LANDING SYSTEM

Previously we motivated the need for alternate routes and framed an optimization problem to solve for them. We now present the entire architecture to control and land a helicopter which will use the computation of alternate routes as one of its key components. We start by describing the specific planning problem and then introduce our approach to solve it.

Planning Problem

The planning problem we are looking to solve is to safely land a helicopter whose engine has failed mid-air. The helicopter is equipped with a GPS/INS system providing accurate pose, a device to measure rotor speed, and a 3-D scanning LADAR with range of up to 700 metres. The system is provided with a terrain map of the area and a detailed landing map with a continuous score of landing feasibility.

Formally the problem is framed as follows

$$\begin{array}{rc} \textit{minimize}: & J = \int_0^{t_f} c(x(t))dt + c(x(t_f)) \\ \textit{constraints}: & \dot{x} = f(x(t),u,t) \\ & x(0) = x_0 \\ & x(t_f) \in X_{LZ} \\ & g \leq 0, J < \infty \end{array} \tag{2}$$

where x is the 13 dimensional state space of positions, euler angles, the corresponding derivatives and the main rotor speed as shown below:

$$x = \{x^E, y^E, z^E, \dot{x}^E, \dot{y}^E, \dot{z}^E, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}, \Omega\}$$

where the E superscript designates a North East Down (NED) coordinate frame. The command u is the collective, cyclic and pedal inputs as shown below:

$$u = \{\delta_{col}, \delta_{lon}, \delta_{lat}, \delta_{pedal}\}$$

x_0 is the state of the vehicle at the time of engine failure, t_f is the terminal time, X_{LZ} is the set of allowable landing zones. J is the optimization objective function that encodes the criteria for safety of the vehicle. g is a set of environmental and vehicle constraints. f represents the autorotation dynamics of the helicopter. We use the autorotation model from Tierney (Ref. 6).

An interesting thing to note is that the dynamics of the auto-rotating model has a continuum of trim states where no rotor energy is lost. Every trim state has a pair of corresponding airspeeds and descent velocities. The two critical points of concern are minimum descent speed and maximum glide slope. They reflect the two different operating points we want to plan around - either optimize for minimum descent rate but restrict reachability or maximize glide slope but run the risk of having a larger descent rate.

Approach

Given the difficult dynamic constraints and the fact that the trajectories span several kilometres, we do not attempt to solve the generic autorotation problem in real-time. Instead we make a set of reasonable assumptions to more clearly define the problem we wish to solve.

Assumption 1 The engine failure occurs at a high enough altitude such that the 3 distinct stages of autorotation can be achieved - Entry into glide, Glide and Flare. When the engine fails, the idea is to transition to the trim state, stay in it so as to prevent the loss of rotor energy, reach a height from which flare can be achieved and execute the flare maneuver. This puts the minimum AGL at which engine failure can occur at 500 feet. At heights below this limit, the decoupled 3 stage approach will not work - a more coupled approach is required.

Assumption 2 The vehicle has a minimum airspeed when the engine fails such that it can be transitioned into a desired trim state within 5 seconds. This limits the speed between 60 knots and 140 knots. If the airspeed is beyond these limits, the maneuver required to bring it to trim state might take so much time that Assumption 1 is violated. At speeds beyond the limit, the vehicle can potentially enter a dangerous vortex state, which requires a much more specific planner.

Assumption 3 When the engine fails, the vehicle isn't immediately surrounded by obstacles till it is in trim state. The maneuver which transitions to trim state deals with the full 13 dimensional state, is pre-computed and cannot path plan around obstacles. Essentially the vehicle needs a known safety volume of maximum 250 m which is well within perception range and can easily be ensured.

The above assumptions essentially define a set of reasonable initial conditions, which we leverage to simplify the problem and in turn make a robust real-time approach. We decompose the problem into various sub-problems, link them via a state machine and frame it as a multiple objective planning problem (Scherer et al. (Ref. 13)). This is motivated primarily by the fact that the planning problem during the flare phase being different from the glide phase in terms of scope, precision and difficulty. The overview of the system architecture is shown in Fig. 5.

Originally the vehicle is in cruise mode when the engine has not failed yet. The planning system runs in the background computing plans at a fixed interval. On engine failure, a state machine is activated.

The state machine of objectives are

- Entry glide: After engine failure, the helicopter transitions to a trim state to arrest rotor energy loss. A fixed time, typically around 5 seconds, is assigned to this state, during which it continuously provides a set of glide paths. The pilot browses through the paths and selects one.
- Glide: The vehicle attempts to follow the command trajectory, while continuing to compute AR. The pilot or the system can switch paths any time during the phase.
- Entry flare: At a height above the LZ, the helicopter transitions to a safe flare initiation state.
- Flare: The vehicle engages in a precise move to use its rotor energy to come to a halt and touchdown.

Among these sub-problems, computing optimal glide paths is an interesting problem from a pure planning perspective. The solution space of glide paths that satisfy constraints is very large as the plans originate at altitudes well above terrain. Our approach is to translate these constraints into costs, thus searching for solutions of good quality. We adopt the unconstrained planning problem approach from Scherer et al. (Ref. 13) which only imposes the condition that cost incurred must be finite. The cost function used in all subproblems are summarized below.

- $J_{state}(x_{state})$: Weighted distance from a desired x_{state}.
- J_{obs} : Cost decays with squared distance from occupied cells in the evidence grid. The grid uses terrain information as a prior and updates from sensor data.
- J_{curv} : The curvature cost increases exponentially as curvature reaches constraint limits. This ensures paths do not demand large accelerations.
- J_{lzd} : (X,Y) distance from predicted LZ. This attempts to maximize the sensor's viewing time of the LZ so a cluttered LZ can be evaluated before hand.
- J_{rpm} : Deviation from 100% RPM. This retains control authority incase unmapped obstacles appear.
- J_{lz} : Depending on potential risk of LZ (slope or clutter), a terminal cost is assigned.
- J_u : Energy expended by control actions

The planning problem structure is represented as:

$$\begin{aligned} \mathit{minimize}: \quad & J_i = \int_0^{t_f} c(x(t))dt + c(x(t_f)) \\ \mathit{constraints}: \quad & \dot{x} = f(x(t), u(x,t,P_c), t) \\ & x(0) = x_0 \\ & x(t_f) \in X_{end,i} \\ & g_i \leq 0, J_i < \infty \end{aligned} \tag{3}$$

where $i = 1,2,3,4$ is the index of the states in the state machine. The approach is to generate an initial command guess

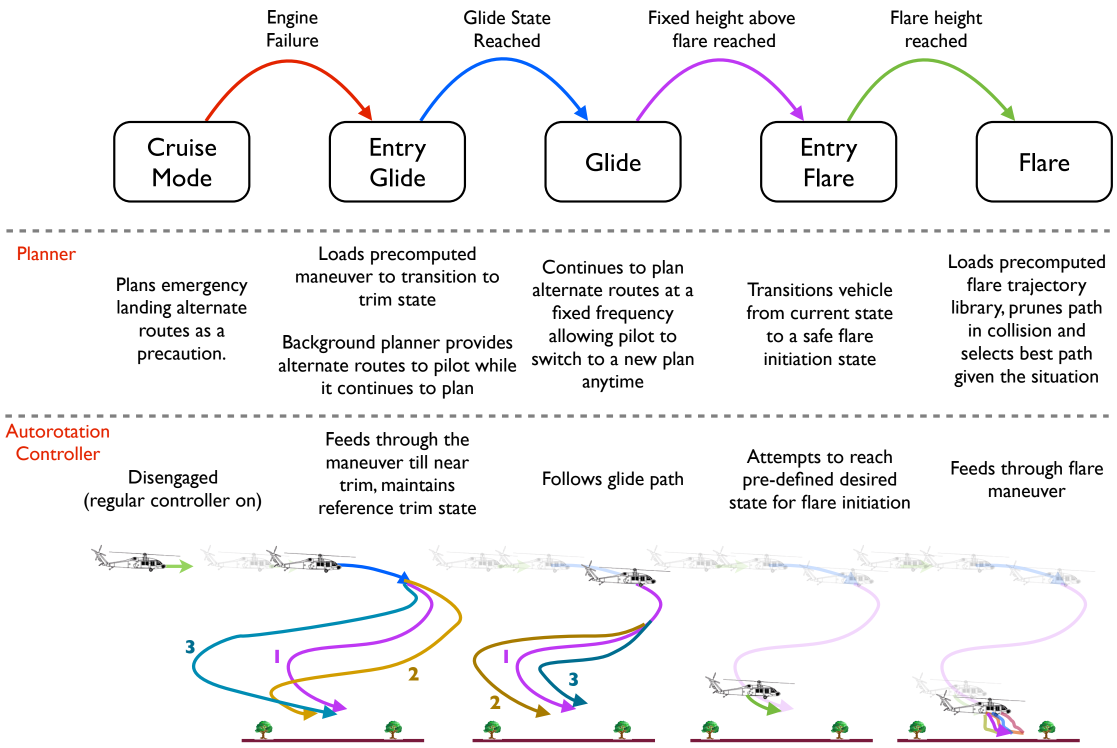

Engine Failure
Glide State Reached
Fixed height above flare reached
Flare height reached
Cruise Mode
Entry Glide
Glide
Entry Flare
Flare
Planner
Plans emergency landing alternate routes as a precaution.
Loads precomputed maneuver to transition to trim state
Background planner provides alternate routes to pilot while it continues to plan
Continues to plan alternate routes at a fixed frequency allowing pilot to switch to a new plan anytime
Transitions vehicle from current state to a safe flare initiation state
Loads precomputed flare trajectory library, prunes path in collision and selects best path given the situation
Autorotation Controller
Disengaged (regular controller on)
Feeds through the maneuver till near trim, maintains reference trim state
Follows glide path
Attempts to reach pre-defined desired state for flare initiation
Feeds through flare maneuver
3
1
2
1
2
3

Fig. 5. Multi-objective state machine for emergency landing. For each stage, the activities of the planner and controller is stated and a schematic of vehicle progress is shown.

Table 1. Planning Problem Definitions

i	State	Cost Function (J_i)	$X_{end,i}$	g_i
1	Entry glide	$J_{state}(x_{trim}) + J_u$	-	-
2	Glide	$J_{obs} + J_{curv} + J_{lzd}$ $+ J_{rpm} + J_{lz}$	X_{fs}	-
3	Entry flare	$J_{state}(x_{flareinit}) + J_u$	-	-
4	Flare	$J_{state}(x_{td}) + J_u$	x_{td}	$x \in X_{safe}$

P_c, perform trajectory optimization based on predicted result, and a low-level tracking controller tracks the guess. Table 1 describe the problem parameters where X_{fs} is a valid flare initiation state, x_{td} is a touchdown state and X_{safe} is a set of states where the vehicle is not in collision. The initial guesses are defined as follows:

Entry glide The desired trim state is chosen to be either minimizing descent speed or maximizing glide slope. The vehicle is transitioned to this state by a precomputed maneuver. Dependent on the current airspeed, a particular maneuver is loaded. This maneuver transitions the vehicle to a basin around the trim state. The controller subsequently stabilizes the vehicle around the trim state. The cost function $J_{state}(x_{trim}) + J_u$ is used to derive LQR gains for the collective and cyclic.

Glide The cost function reflects the desire to minimize risk. Ensuring the pilot has options and there exists feasible backup trajectories, makes alternate routes very critical. The planning algorithm used to generate the initial guess will be covered in detail in the next section. The planner generates a set of routes, in addition to ensuring that the route chosen by the pilot is still feasible. In case the cost of the current path increases, the system will choose to switch to a back up path. The controller in this phase attempts to track the path as well as it can. All the plans terminate at X_{fs}, a valid flare initiation state.

Entry Flare The cost function is similar to the entry glide state. However, no precomputed maneuver is required in this stage since the planner is near trim state when it enters this stage. Only the LQR control is enough to ensure stability around the flare initiation trim state.

Flare This planning problem poses difficulties in creating real-time solutions. It is not trivial to adapt aforementioned real-time flare controllers to avoid obstacles. The state and end constraints makes the optimal control problem hard to solve, and the extreme non-linear nature of the dynamics made it difficult to design a very high fidelity tracking controller. Thus the problem was solved by constructing an “end game” trajectory library (Ref. 14) offline by solving the problem for a large number of initial conditions. While running online, the trajectory library was invoked, collision trajectories pruned out, and the command corresponding to nearest neighbor in this library executed.

Algorithm 1 G = RRT*-AR$((V,E),N)$
Input: V = vertices, E = edges, N=number of iterations
Output: G is the tree returned

```
for $i = 1,\dots,N$ do
    $x_{rand} \leftarrow$Sample
    $X_{near} \leftarrow$Near$(V, x_{rand}, r_{near})$
    $(x_{min}, \sigma_{min}) \leftarrow$ChooseParent$(X_{near}, x_{rand})$
    if CollisionFree$(\sigma_{min})$ then
        $V \leftarrow V \cup \{x_{rand}\}$
        $E \leftarrow E \cup \{(x_{min}, x_{rand})\}$
        $(V,E) \leftarrow$Rewire$((V,E), X_{near}, x_{rand})$
return $G = (V,E)$
```

Algorithm 2 (x_{min}, σ_{min}) = ChooseParent $(X_{near}, x_{rand}, r_{near}, c_{lb})$
Input: X_{near} = set of near vertices, x_{rand} = a sampled point, r_{near} = radius of near neighbours, c_{lb} = lower bound cost
Output: x_{min} = parent resulting in lowest cost, σ_{min} = trajectory from parent

```
$pSet \leftarrow \{\}$ , $c_{min} \leftarrow \infty$ , $x_{min} \leftarrow$NULL , $\sigma_{min} \leftarrow$NULL
for $x_{near} \in X_{near}$ do
    $d_{eq} \leftarrow$min$(D_{eq}, \rho r_{near})$
    $c_{eval} \leftarrow$SteerCostApproxOptim$(x_{near}, x_{rand})$ +Cost$(x_{near})$
    if $c_{eval} < (1+\varepsilon)c_{lb}$ then
        if $\exists x_{eq}$ s.t $\|x_{eq} - x_{rand}\| < d_{eq}$ and $x_{near}$ =Parent$(x_{eq})$ then
            $c_{eval} \leftarrow c_{eval} + \varepsilon c_{lb}$
        $pSet \leftarrow pSet \cup \{c_{eval}, x_{near}\}$
$pSet \leftarrow$sort$(pSet)$
for $x_{parent} \in pSet$ do
    $\sigma \leftarrow$Steer$(x_{parent}, x_{rand})$
    if CollisionFree$(\sigma)$ then
        $c_{min} \leftarrow$Cost$(x_{near})$+Cost$(\sigma)$
        $x_{min} \leftarrow x_{near}$ , $\sigma_{min} \leftarrow \sigma$
        break
return $(x_{min}, \sigma_{min})$
```

Autonomous Control

The control system block consists of an inner loop attitude control and an outer loop path following control. The inner loop is a PID controller with airspeed based gain scheduling. The outer loop is a trajectory tracking control, which computes desired accelerations to track the path and feeds desired attitudes to the inner loop control.

The inner attitude control loop is similar to Yomchinda et al. (Ref. 9), where the pitch and roll commands are filtered by a second order response. The difference between current and desired response is tracked by a feedback PID controller to compute the cyclic and pedal commands. The gains are pre-computed and scheduled according to the airspeed of the vehicle.

The outer path follower tries to minimize cross track error between the vehicle and the path. Similar to an approach in Park et al. (Ref. 15), a point on the trajectory is selected and used to calculate desired lateral acceleration, desired forward and vertical velocity. This is then used to compute bank and pitch angles. The collective is directly computed by desired vertical acceleration required to match the glide slope.

In the case of entry glide and entry flare, since a trim state has to be tracked, the controller is a much simpler LQR controller whose gains are obtained by using the cost function mentioned in Table 1.

RRT*-AR

Previously, we enlisted as a requirement the need for a rapid, "life-long" planner. We then introduced the need for AR and proposed an optimization problem for generating them. However, it remained to be seen what kind of a planner can generate AR within the hard time-constraints. Specifically, we require a planner that

- Can solve the optimization problem to generate AR.
- Is rapid enough to meet the hard time-constraints.
- Carries information across planning cycles to improve plan over time.

There has been a significant amount of research on the planning problem of optimizing a cost function under differential constraints summarized by LaValle (Ref. 16). A significant contribution in this area has been the RRT* algorithm proposed by Karaman et al. (Ref. 17) unifying the speed of sampling based planners with asymptotic optimality guarantees. A state is sampled from the configuration space and neighbours within a ball of radius r_{near} are selected the parent which would result in least cost from root is selected. Then attempts to "rewire" the nodes in X_{near} using this state as a parent is done. RRT* can plan across large spaces by producing a feasible solution quickly and improving it with time.

To solve the optimization problem presented, the RRT* algorithm must be able to match our requirements and generate AR in real time. To achieve this, we propose a way to partially trade off exploitation and precision in exchange for exploration and speed. We call this the *RRT*-Accelerated Alternate Routes with Replanning (RRT*-AR).*

Alternate Routes

Solving the optimization problem proposed in Eq 1 is not trivial. A simple approach would be to apply the RRT* algorithm on the problem, but assuming a goal region instead of a goal, thus allowing multiple solutions. At the end of the planning cycle, we obtain multiple leaf vertices corresponding to the set of candidate solutions. A subset of these candidates, which satisfy the definition of AR, is selected and returned. For the algorithm to be effective, we require two main characteristics. Firstly, the solution set must have enough variation to ensure

Algorithm 3 $G = \text{Rewire}((V,E), X_{near}, x_{rand}, r_{near}, c_{lb})$
Input: V = vertices, E = edges, X_{near} = set of near vertices, x_{rand} = a sampled point, r_{near} = radius of near neighbours, c_{lb} = lower bound cost
Output: G is the tree returned

```
1  for $x_{near} \in X_{near}$ do
2      $c_{eval} \leftarrow \text{SteerCostApproxPessim}(x_{rand}, x_{near}) + \text{Cost}(x_{rand})$
3      if $\exists x_{eq}$ s.t $\|x_{eq} - x_{near}\| < d_{eq}$ and $x_{rand} = \text{Parent}(x_{eq})$ then
4          $c_{eval} \leftarrow c_{eval} + \varepsilon c_{lb}$
5      if $c_{eval} < (1+\varepsilon)c_{lb}$ and $c_{eval} < \text{Cost}(x_{near})$ then
6          $\sigma \leftarrow \text{Steer}(x_{rand}, x_{near})$, $d_{eq} \leftarrow \min(D_{eq}, \rho r_{near})$
7          if $\text{Cost}(x_{rand}) + \text{Cost}(\sigma) < \text{Cost}(x_{near})$ then
8              if CollisionFree($\sigma$) then
10                 $x_{parent} \leftarrow \text{Parent}(x_{near})$
11                 $E \leftarrow E \setminus \{x_{parent}, x_{near}\}$
12                 $E \leftarrow E \cup \{x_{rand}, x_{near}\}$
13 return $G = (V,E)$
```

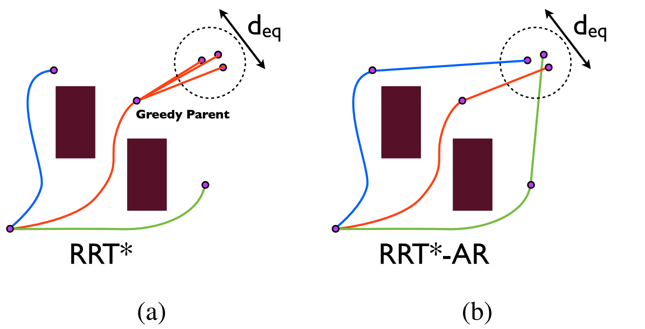


Fig. 6. RRT* exploits the cost function while RRT*-AR allows for more variation. (a) RRT* joins all the vertices in the ball of diameter d_{eq} to the optimal parent which seems redundant. (b) RRT*-AR treats the vertices to belong in the same equivalence class, penalized the greedy parent and allows other connections.

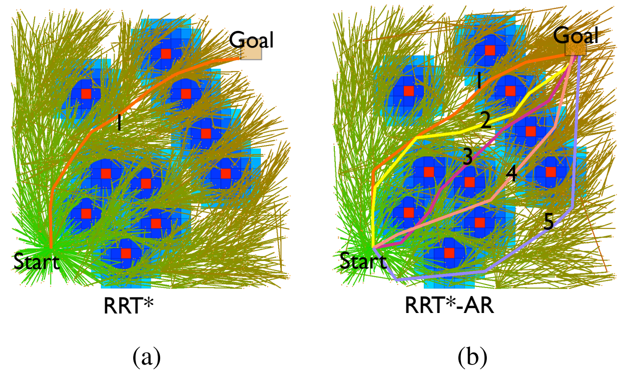


Fig. 7. RRT*-AR computes 5 routes while RRT* computes just 1. (a) RRT* tree is dense around optimal which connects to all vertices reaching the goal (b) RRT*-AR seems to have multiple dense clusters corresponding to different routes.

that it contains a set of potential AR. Secondly, the process of selecting the final set should be computationally cheap.

When the RRT* samples a point, it tends to join it with its best parent. This implies points nearby to each other would all have the same parent. This exploitation property tends to create dense trees, with the density being centred around the optimal path. For alternate routes to exist, nearby vertices should consider alternate parents which have similar cost to the best possible parent. For example, in a 2D case, this allows the tree to have a segment in every homotopy class. One such way would be, if the best parent to a vertex already has children nearby, the second best parent gets a chance. We formally try to capture this in Algorithm 2 and 3.

We first define an *equivalence class* of vertices which are within d_{eq} distance of each other. Formally, samples x_1 and x_2 are said to belong to the same equivalence class if $\|x_1 - x_2\| < d_{eq}$. With the assumption of a continuous cost function, vertices belonging to an equivalence class have a bounded variation in the cost it takes to reach them from the root. Given two equivalent vertices v_1 and v_2, it is likely that both will have the same parent v_{p1} and in that case the existence of both these vertices will be redundant. However, when v_2 is being created, if we add a phantom cost to the edge joining v_{p1} and v_2, we allow for v_2 to search for another parent. This phantom cost then corresponds to how much of exploration we allow for discovery of other routes. If x_{parent} is the parent of x_1 then it is penalized by a phantom cost εc_{lb} (the maximum cost variation allowed among alternate routes) while being evaluated as a parent of x_2. This is shown in Fig 6 where a penalization is applied due to equivalence class presence.

An important question is - what is the effect of penalizing the greedy parent in terms of the asymptotic optimality guarantees of the algorithm? To restore the guarantees, the value d_{eq} shrinks as a constant fraction of the RRT* radius r_{near}. Thus, in the limit the algorithm converges to the same structure as RRT*. To prevent the loss of the alternate routes via rewiring, a periodic "latching" procedure is applied. In this procedure, alternate routes are computed, then locked from rewiring till the next procedure comes up. This preserves the alternate routes while continuing to improve their local optimality. The proofs and analysis of this result can be found in Choudhury et al. (Ref. 18). As shown in Fig 7, the RRT*-AR has more variety in solution while still having the optimal solution in its set.

Finding a combination of m AR from a set of M candidate solutions requires considering up to $\binom{M}{m}$ combinations. For efficiency purposes, we settle for the greedy suboptimal solution of finding the first m lowest cost trajectories which satisfy the AR constraints. For every candidate in the cost sorted list of trajectories, a similarity metric is computed with respect to a running buffer set of accepted AR, and if it satisfies all constraints, it is appended to buffer set.

We now review whether RRT*-AR matches the requirements for AR.

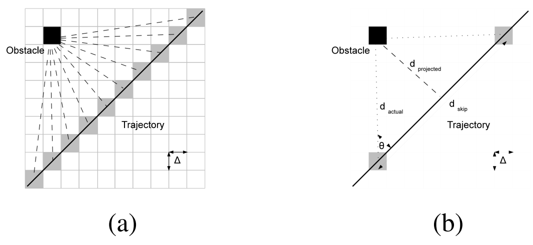


Fig. 8. RRT*-AR has a 10 times speed up on obstacle cost checks. (a) RRT* evaluates path by calculating obstacle cost in a nominal way - step along every grid cell on the path and accumulate the cost (b) RRT*-AR uses optimistic / pessimistic bounds on the cost - the pessimistic bound is to find nearest obstacle distance and assume it remains the same for the entire line.

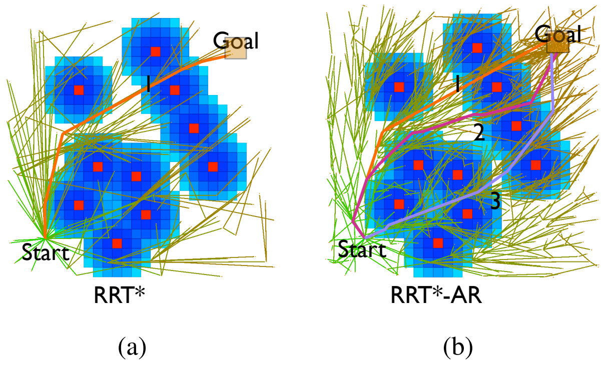


Fig. 9. After 0.5 seconds, RRT*-AR has a tree 6 times bigger, and has already computed 3 routes. (a) RRT* does a correct but expensive evaluation process and has very few paths reaching goal (b) RRT*-AR does a fast evaluation process with asymptotic correctness, thus growing many paths to goal.

- Limited sharing of a path swath - The similarity metric mentioned above applies this constraint. The variation of parent children combinations in RRT*-AR ensures that such paths exist if possible.
- Local Optimality - RRT*-AR initially explores by trying different parent child combination. As the radius of the disk shrinks, the exploitation nature dominates. The latching procedure ensures that the routes themselves are preserved, but at the same time are locally improved.
- Uniform Bound Stretch - The penalization factor in RRT*-AR is the specified allowable cost variation. This ensures that paths beyond the cost bound is never considered.

Accelerated Planning

Our key emphasis has been on generating AR in real time. RRT*-AR achieves this by trading off precision in cost for a boost in speed. A bottleneck for speed in RRT* is when it makes a connection between a parent and a child, it ranks the parents by the true cost of the trajectory joining the two. But for the best parent to get the lowest rank, the cost need not be true, it just needs to have the same order. So making an approximation on evaluation can boost iteration speed many times. In Fig. 8, while the RRT* evaluates the obstacle cost by stepping over every grid cell that the path passes through, RRT*-AR evaluates a bound on the cost. While choosing a parent the cost is optimistic (Algorithm 2) such that a viable parent isnt suppressed due to cost approximation errors. During rewiring (Algorithm 3) the cost is pessimistic. In this approach, there is a probability of being wrong, i.e, choosing the wrong parent. Still by bounding the probabbility, asymptotic optimality can be guaranteed (Ref. 18).

Reuse Tree

To retain the effort of creating the search tree and cost evaluation of trajectory segments across iterations, we implement a method to reuse the tree by finding a way to "latch" the vehicle state onto an existent tree as shown in Fig 10. In the first step, the cost of all vertices from the root is made invalid (-1), even though the cost from their parents is retained. Next, the current state of the vehicle is taken and a set of near vertices from the tree is obtained. This near vertices have to be sorted by their depth in the existing tree. The current state is now rewired to the tree. Since all vertices have cost from root as -1 before rewiring and the current state has no parent, after rewiring it becomes the root of the valid tree. By depth sorting the near vertices, the current state attempts to rewire by giving vertices with lower depth more priority. This is because on rewiring to these vertices first, their subtrees are also made valid, making the rewiring process efficient. The most common example of reusing a tree would be that the current state would rewire to most of the children of the previous root. This would result in most of the tree remaining static, while vertices near the root would change parents. The next step involves executing the RRT* in the usual way and returning the new tree.

RESULTS

To test the system in a realistic setting we setup an experiment in simulation with a UH-60L Black Hawk experiencing failures at different locations over the mountains of San Juan National Forest, Colorado (37° 40' 28.00",-107° 34' 4.5264") at an average height of 900m AGL. Grid obstacle cost in this case was purely derived from the digital elevation map. Fig. 11 shows a typical glide planning scenario over a land which has 2 minimas in landing cost. The planners were given the time period of entry glide state, i.e. 5 seconds.

RRT* converges to having 2 routes, 1 going to each of the peaks. RRT*-AR comes up with 6 routes (the total number asked for) which include the solutions arrived at by RRT*. These solutions are interesting because they are placed on either side of mountain peaks, have different amounts of LZ visibility and curvature and thus provide a comprehensive set to the pilot.

The alternate routes also show the expected safety behaviour when an unmapped obstacle is sensed. In Fig. 12,

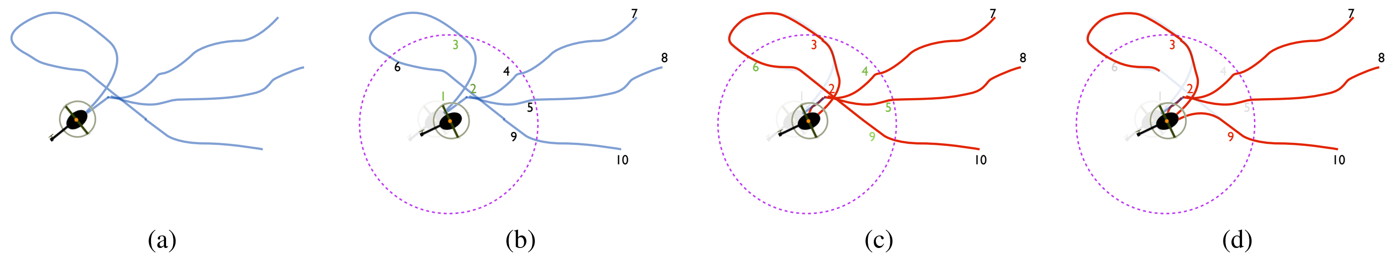


Fig. 10. Life-long planning by re-using search tree across planning cycles. (a) Search tree in previous iteration. (b) Root moves forward and wishes to latch onto the existing tree, which is de-activated for the moment. It does a near neighbour lookup and retrieves vertices which are sorted by their depth in the search tree. (c) The new root attempts to rewire to vertices in the sorted order. On successful rewiring, the sub-branch is re-activated and costs to the vertices is restored (branches in red). (d) If the current root is a better parent to vertices of greater depth, the vertices are rewired (vertex 9).

Table 2. Comparison between RRT* and RRT*-AR

	RRT*		RRT*-AR	
	Cost (var)	Count	Cost (var)	Count
Best	1.0($\pm$0.0)	100%	1.18($\pm$0.25)	100%
Path1	1.89($\pm$1.21)	87.36%	1.51($\pm$0.47)	99.75%
Path2	2.98($\pm$1.96)	57.3%	2.07($\pm$1.07)	95.95%
Path3	3.05($\pm$0.97)	15.42%	2.92($\pm$1.77)	85.71%
Path4	3.65($\pm$1.39)	4.05%	3.77($\pm$2.25)	66.12%
Path5	3.87($\pm$0.65)	1.14%	4.51($\pm$2.83)	43.74%

Table 3. Cost comparisons between RRT* and after speedups

	Cost $\tau = 0.6s$	Cost $\tau = 1.0s$	Time to cost = 1.1
RRT*	1.97($\pm$1.28)	1.60($\pm$0.86)	4.40($\pm$1.28)
RRT*-AR	1.65($\pm$1.03)	1.37($\pm$0.61)	2.62($\pm$1.07)

the helicopter maintains back up plans going to another landing site. Interestingly, it chooses to fly over the landing zone (enhanced LZ visibility reduces cost of path), while maintaining backups which reach the other landing site. This allows it to instantly switch routes.

The results in Table 2, for 791 runs of 5s each with AR parameters ($\varepsilon = 4, \gamma = 0.7, d_{nn} = 500, d_{eq} = 500, \rho = 0.2$) show how many times each planner could compute up to 5 routes, and the cost of the routes computed. RRT*-AR comes up with a much higher number of routes (2.82 times) while having an allowable best cost variation (18%). The RRT* shows the typical aggressively optimal nature with a sharp distribution in AR frequency and cost. RRT*-AR has a smoother distribution of cost and frequency.

After 3318 runs, each run continuing till 1000 vertex additions, RRT*-AR achieves 15.4% in terms of cost reduction than RRT* as shown in Table 3. Fig. 14 shows that the key performance enhancement is because RRT*-AR produces an acceptable path (0.29s) faster than RRT* (0.54s), and maintains a similar rate of cost reduction initially.

The reuse of trees made the system more reliable as shown in Fig. 15 after 450 runs. It quickly converts most paths to near optimal while creating a new tree every time would take several iterations till arriving at the right answer. The histogram's decay shape shows how effective a method is, which in case of reusing peaks at the optimal while for planning afresh is more level.

Fig. 13 shows the testing of the system in a virtual reality platform. A video of demonstration of the system can be found in http://youtu.be/o1J3qIG4R4c .

CONCLUSIONS

We presented a complete autonomous landing system for a helicopter after engine failure. In the context of emergency landing of helicopters, the planning system makes a significant stride forward - it can compute safe paths in real time, deal with unmodelled obstacles or cluttered landing zones and takes a human decision into account. The RRT*-AR algorithm is effective in maximizing the likelihood of safe alternative routes by returning 2.8 times the expected number of AR as does RRT* with speedups of 67%. With the local optimality and spatial separation features of AR, the RRT*-AR increases the probability of finding a risk free path if the sensor detects an obstacle on descent or if the pilot wishes to over-ride the current path that is being followed.

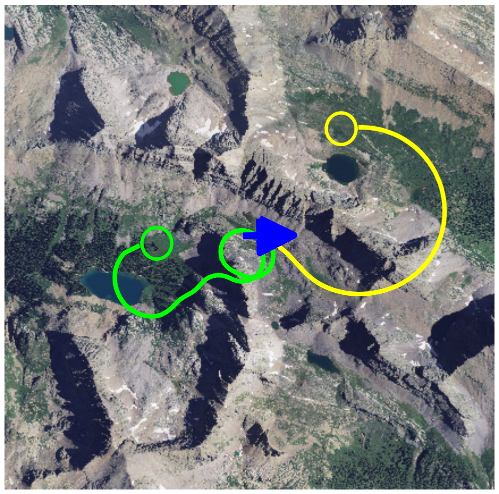
In future work, we intend to explore the benefits AR gives towards system robustness. For example, ARs can be computed ahead of time at regions of high risk. We also intend to address issues such as safety metrics of AR for a more comprehensive landing system.

REFERENCES

[1]Johnson, W., "Helicopter optimal descent and landing after power loss," , 1977.

[2]Lee, A., Bryson, A., and Hindson, W., "Optimal landing of a helicopter in autorotation," , 1986.

[3]Aponso, B., Lee, D., and Bachelder, E., "Evaluation of a Rotorcraft Autorotation Training Display on a Commercial Flight Training Device," *Journal of the American Helicopter Society*, Vol. 52, (2), 2007, pp. 123–133.


(a)

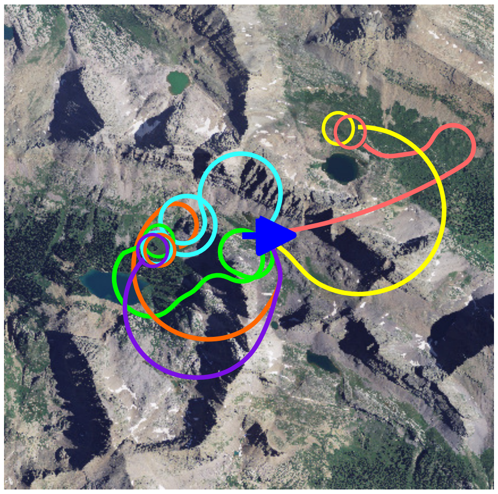
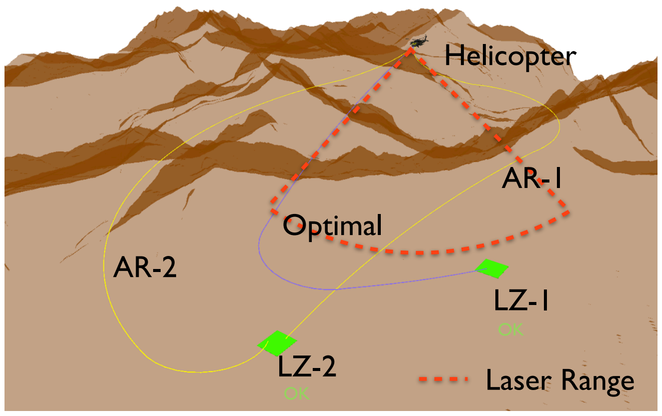
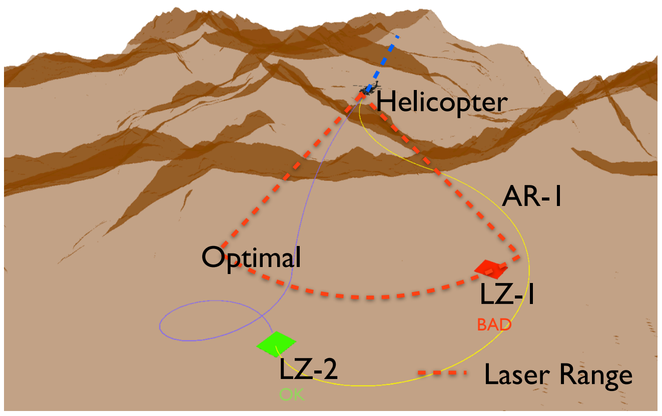

(b)

Fig. 11. AR for a failure over the mountains. (a) RRT* comes up with 2 routes - each going to a maximal feasible LZ (b) RRT*-AR coming with 6 routes including the routes of RRT* - each path having its own merit.



(a)

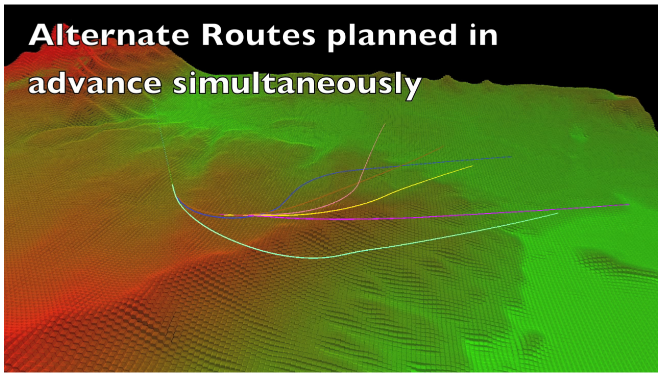


(b)

Fig. 12. Alternate Routes allow quick reaction to clutter in LZ during a landing scenario in the valley. (a) The optimal path (blue) is followed by the helicopter while simultaneously planning alternate (yellow) routes (b) LZ-1 comes within the sensor range and is observed to be infeasible. The helicopter instantaneously switches to an alternate route.

(a)

(b)



(c)

(d)

Fig. 13. Successful landing of a helicopter in a virtual reality platform. (a) Helicopter is in cruise mode (b) Engine failure occurs and autonomous system takes over (c) While glide established, system has already come up with alternate routes which a pilot can choose from (d) Helicopter flares and touches down.

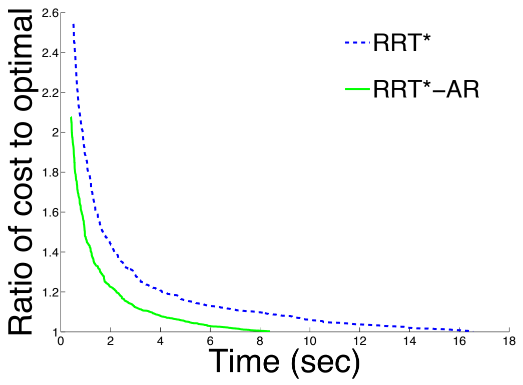


Fig. 14. RRT*-AR costs decay rapidly in the beginning indicating fast exploration of search space and an increased rate of vertex addition.

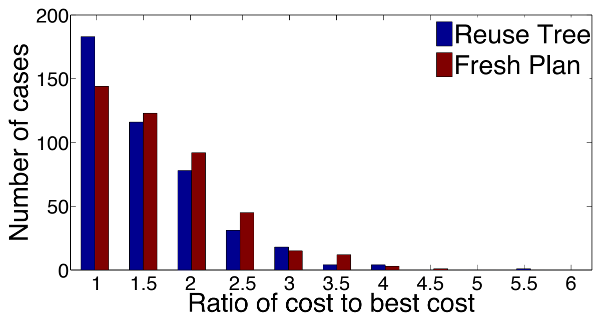


Fig. 15. If the search tree is reused across iterations, more paths are closer to optimal than if a new tree is created at every iteration.

[4]Aponso, B., Bachelder, E., and Lee, D., “Automated autorotation for unmanned rotorcraft recovery,” AHS international specialists meeting on unmanned rotorcraft, 2005.

[5]Bachelder, E. and Bimal, L., “Using Optimal Control for Rotorcraft Autorotation Training,” Proceedings of the American Helicopter Society 59th Annual Forum, Phoenix, Ariz, 2003.

[6]Tierney, S., *Autorotation Path Planning Using Backwards Reachable Set and Optimal Control*, Master’s thesis, The Pennsylvania State University, 2010.

[7]Abbeel, P., Coates, A., Hunter, T., and Ng, A., “Autonomous autorotation of an RC helicopter,” Experimental Robotics, 2009.

[8]Dalamagkidis, K., Valavanis, K., and Piegl, L., “Autonomous autorotation of unmanned rotorcraft using nonlinear model predictive control,” Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009, 2010.

[9]Yomchinda, T., Horn, J., and Langelaan, J., “Autonomous Control and Path Planning for Autorotation of Unmanned Helicopters,” Proceedings of the American Helicopter Society 68th Annual Forum, Fort Worth, Texas, May 1-3, 2011.

[10]Green, C. J. and Kelly, A., “Toward Optimal Sampling in the Space of Paths,” ISRR, 2007.

[11]Erickson, L. H. and LaValle, S. M., “Survivability: Measuring and ensuring path diversity,” ICRA, 2009.

[12]Abraham, I., Delling, D., Goldberg, A., and Werneck, R., “Alternative routes in road networks,” *Experimental Algorithms*, 2010, pp. 23–34.

[13]Scherer, S. and Singh, S., “Multiple-objective motion planning for unmanned aerial vehicles,” Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, 2011.

[14]Stolle, M. and Atkeson, C., “Policies based on trajectory libraries,” Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, 2006.

[15]Park, S., Deyst, J., and How, J. P., “A new nonlinear guidance logic for trajectory tracking,” AIAA Guidance, Navigation, and Control Conference (GNC), August 2004 (AIAA 2004-4900).

[16]LaValle, S., *Planning algorithms*, Cambridge Univ Press, 2006.

[17]Karaman, S. and Frazzoli, E., “Incremental sampling-based algorithms for optimal motion planning,” Proc. Robotics: Science and Systems, 2010.

[18]Choudhury, S., Scherer, S., and Singh, S., “Realtime Alternate Routes Planning:The RRT*-AR Algorithm,” Technical Report CMU-RI-TR-12-27, Robotics Institute, Pittsburgh, PA, December 2012.