RRT*-AR: Sampling-Based Alternate Routes Planning with Applications to Autonomous Emergency Landing of a Helicopter

Sanjiban Choudhury, Sebastian Scherer and Sanjiv Singh

***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 could react quicker to regain control, however current emergency landing methods only generate dynamically feasible trajectories without considering obstacles. We address the problem of autonomously landing a helicopter while considering a realistic context: multiple potential landing zones, geographical terrain, sensor limitations and pilot contextual knowledge. We designed a planning system to generate alternate routes (AR) that respect these factors till touchdown exploiting the human-in-loop to make a choice. This paper presents an algorithm, RRT*-AR, building upon the optimal sampling-based algorithm RRT* to generate AR in realtime and examines its performance for simulated failures occurring in mountainous terrain, while maintaining optimality guarantees. 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. Using AR, the focus can now shift on delivering safety guarantees and handling uncertainties in these situations.**

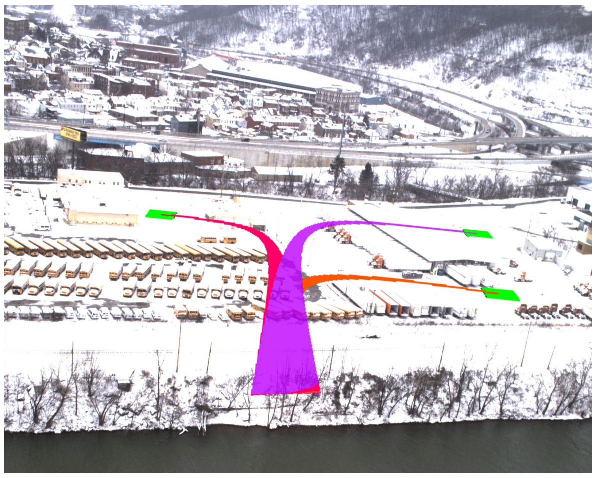


Fig. 1. Sample route set for safe landing after engine failure. This example shows a set of potential engine-out trajectories for a Bell 206 using the proposed RRT*-AR algorithm. The paths were calculated from a registered dataset from prior elevation maps, pointcloud, and image data collected onboard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

I. 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 executes a maneuver, known as autorotation, to land safely. The rotor drag is overcome by the flow of air through the blades, allowing the gravitational potential energy of the helicopter to be transferred to the rotor. The pilot controls the helicopter in a glide path and at the end trades off the energy in a move known as flare, to slow down and come to a safe touchdown. However, to do both precise control as well as choose a minimum risk landing zone, becomes a very difficult task for a human. Similarly, a single optimum path computed by an autonomous system cannot always reflect human preference or be quickly modified when obstacles are detected.

Research on this topic mainly focuses on the dynamical solution to autorotation. Optimal trajectories minimizing touchdown speed have been derived by Johnson [1] and Lee et al. [2], used to calculate control inputs by Aponso et al. [3], and as pilot assistance during emergencies by Aponso et al. [4] and Bachelder et al. [5]. The dynamics of flare have been a special focus for determining its feasibility by Tierney [6] as well as in imitation learning by Abbeel et al. [7] and model predictive control by Dalamagkidis et al. [8]. Recently, a complete feasible solution from glide to touchdown has been demonstrated (Yomchinda et al.[9]). However, these approaches have not focussed on the issues of being real time, 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) that addresses these issues thus maximizing the possibilities of a safe landing.

In summary, the main contributions are

- a definition of an algorithm to provide alternate routes as a solution to an optimization problem,
- acceleration of the algorithm by leveraging allowable variation in optimum cost and by leveraging differential constraints of the vehicle and
- a multi-objective planning system to autonomously land a vehicle after engine failure.

In Section II we formulate the optimization problem and define AR. In Section III, we present the RRT*-AR algorithm as components building upon the RRT* algorithm to generate AR. In Section IV we present a system architecture using a multiple objective planning method to control the helicopter through all stages post-failure. In Section V we present

S. Choudhury, S. Scherer and S. Singh are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A., {sanjiban,basti,ssingh}@cmu.edu

results on both quality and speed of AR for simulations in mountainous regions and conclude in Section VI summarizing the improvements.

II. Problem Formulation

We express the planning problem as an optimization minimizing a risk, and solve for AR. This can be expressed as

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

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.

Abraham et al. [10] describe AR as a set satisfying the following conditions:

1) $J(\sigma_i \cap \{\sigma_0 \cup \cdots \cup \sigma_{i-1}\}) \leq \gamma J \|\sigma^*\|$ (Limited Sharing of new alternative and previous alternative paths)
2) σ_i is T-locally optimal for $T = \alpha J(\sigma^*)$ (Local Optimality)
3) $J(\sigma_i)$ is $(1+\varepsilon)J(\sigma^*)$ (Uniform Bound Stretch)

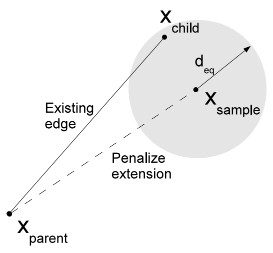
III. RRT*-AR

There has been a significant amount of research on the planning problem of optimizing a cost function under differential constraints summarized by LaValle[11]. A significant contribution in this area has been the RRT* algorithm proposed by Karaman et al.[12] unifying the speed of sampling based planners with asymptotic optimality guarantees. RRT* can plan across large spaces by producing a feasible solution quickly and improving it with time. A state is sampled from the configuration space and neigbours 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. To solve the optimization problem presented, the RRT* algorithm must be able to 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).*

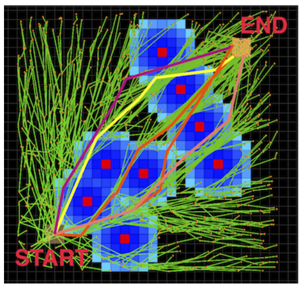
A. Alternate Routes

Solving the optimization problem proposed in Section II, subject to the constraints in the definition of AR, is not trivial. A simple approach would be to apply the RRT* algorithm on the multiple goal region problem. 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 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



(a)



(b)

Fig. 2. (a) Presence of a sibling vertex (x_{child}) in the equivalence class (shaded area of radius d_{eq}) penalizes x_{parent} for contesting to be parent of x_{sample} (b) Applying equivalence class creates more variation in the otherwise dense RRT* tree, creating 5 routes instead of just one.

Algorithm 1 (x_{min}, σ_{min}) = ChooseParentAlternate($X_{near}, x_{rand}, r_{near}, c_{lb}$)
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

```
1  $c_{min} \leftarrow \infty$ , $x_{min} \leftarrow$NULL , $\sigma_{min} \leftarrow$NULL
2  for $x_{near} \in X_{near}$ do
3      $\sigma \leftarrow$Steer($x_{near}, x_{rand}$), $d_{eq} \leftarrow$min($D_{eq}, \rho r_{near}$)
4      $c \leftarrow$Cost($x_{near}$)+Cost($\sigma$)
5      if $\exists x_{eq}$ s.t $\|x_{eq} - x_{rand}\| < d_{eq}$ and $x_{near}$ =Parent($x_{eq}$) then
6          $c \leftarrow c + \varepsilon c_{lb}$
7      if $c < c_{min}$ then
8          $c_{min} \leftarrow c$
9          $x_{min} \leftarrow x_{near}$ , $\sigma_{min} \leftarrow \sigma$
10 return $(x_{min}, \sigma_{min})$
```

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

has children nearby, the second best parent gets a chance. We formally try to capture this in Algorithm 1 and 2.

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 2(a) where a penalization is applied due to equivalence class presence.

The intuition behind this approach is that the RRT* tree looks a lot more like a graph, without carrying the burden of a graph like representation as we can see in Fig 2(b). This algorithm is asymptotically optimal, by including the term ρ and ensuring:

$$\gamma_{RRT^*AR} \geq \frac{2}{(1-\rho)}((1+\frac{1}{d})(\frac{\mu(X_{free})}{\zeta_d}))^{\frac{1}{d}}, \quad \rho < 1 \qquad (2)$$

where ρ is the ratio of d_{eq} to the radius of near neighbours r_n, d is the dimension of the space, X_{free} is the volume of free space and ζ_d is the volume of an unit d-dimensional ball. The proof of this result can be found in Choudhury et al.[13].

Finding a combination of m AR from a set of M candidate solutions requires considering upto $\binom{M}{m}$ 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.

B. Accelerated Planning

Our key emphasis has been on generating AR in real time. To achieve this speed up we introduce two algorithms - Algorithm 3 uses cost approximation and Algorithm 4 exploits the unique reachability volume of the system.

Algorithm 3 demonstrates how we trade 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 evaluating the true cost. 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 but there is a probability of being wrong, i.e, choosing the wrong parent. For asymptotic optimality, this probability $p(n)$ has to be bounded [13] as follows:

Algorithm 3 (x_{min}, σ_{min}) = ChooseParentApprox$(X_{near}, x_{rand}, c_{lb})$
Input: X_{near} = set of near vertices, x_{rand} = a sampled point, c_{lb} = lower bound cost
Output: x_{min} = parent resulting in lowest cost, σ_{min} = trajectory from parent

```
1  $pSet \leftarrow \{\}$ , $x_{min} \leftarrow$ NULL , $\sigma_{min} \leftarrow$ NULL
2  for $x_{near} \in X_{near}$ do
3      $c_{eval} \leftarrow$ SteerCostApproxOptim$(x_{near}, x_{rand})$+Cost$(x_{near})$
4      if $c_{eval} < (1+\varepsilon)c_{lb}$ then
5          $pSet \leftarrow pSet \cup \{c_{eval}, x_{near}\}$
6  $pSet \leftarrow$ sort$(pSet)$
7  for $x_{parent} \in pSet$ do
8      $\sigma \leftarrow$ Steer$(x_{parent}, x_{rand})$
9      if CollisionFree$(\sigma)$ then
10         $c_{min} \leftarrow$ Cost$(x_{near})$+Cost$(\sigma)$
11         $x_{min} \leftarrow x_{near}$ , $\sigma_{min} \leftarrow \sigma$
12         break
13 return $(x_{min}, \sigma_{min})$
```

Algorithm 4 G = ConstrainedRRT*$((V,E), N)$
Input: V = vertices, E = edges, N=number of iterations
Output: G is the tree returned

```
1  for $i = 1, \ldots, N$ do
2      $x_{rand} \leftarrow$ Sample
3      $(x_p, r_p) \leftarrow \left\{ \underset{x \in \mathbb{R}^m, r \in \mathbb{R}}{\text{argmin}}\ r \;\middle|\; Ball(x,r) \cap \Sigma_{back}(x_{rand}) \supseteq Ball(x_{rand}, r_{near}) \cap \Sigma_{back}(x_{rand}) \right\}$
4      $X_{nearp} \leftarrow$ Near$(V, x_p, r_p)$
5      $(x_{min}, \sigma_{min}) \leftarrow$ ChooseParent$(X_{nearp}, x_{rand})$
6      if CollisionFree$(\sigma_{min})$ then
7          $V \leftarrow V \cup \{x_{rand}\}$
8          $E \leftarrow E \cup \{(x_{min}, x_{rand})\}$
9          $(x_c, r_c) \leftarrow \left\{ \underset{x \in \mathbb{R}^m, r \in \mathbb{R}}{\text{argmin}}\ r \;\middle|\; Ball(x,r) \cap \Sigma_{front}(x_{rand}) \supseteq Ball(x_{rand}, r_{near}) \cap \Sigma_{front}(x_{rand}) \right\}$
10         $X_{nearc} \leftarrow$ Near$(V, x_c, r_c)$
12         $(V,E) \leftarrow$ Rewire$((V,E), X_{nearc}, x_{rand})$
13 return $G = (V,E)$
```

$$p(n) < n^{-(1+1/d)}, \quad \forall n \geq M \qquad (3)$$

where d is the dimension of the space and M is a very large integer.

Algorithm 3 shows how cost approximation while choosing a parent for a sample has to be optimistic. We note that an optimistic cost can often return finite cost for segments passing through obstacles, and in such cases we move to the second best parent and so on. On the other hand, the rewiring process considers the pessimistic cost approximation.

Secondly, we also address the case of a vehicle with dynamic constraints represented as reachability volume, the volume that it can reach being Σ_{front} and the volume that can reach in Σ_{back}. We consider the special case of $\Sigma_{front}(x) \cap \Sigma_{back}(x) = \{x\}$, i.e., the vehicle cannot loop. In such a case, symmetric nearest neighbor lookups in RRT* will neither have a lot of parents nor children. We solve this issue by breaking symmetry in this lookup and creating two smaller lookups - one for finding parents and one for children. Algorithm 6 shows this, and the radius of the lookup ensures asymptotic optimality by definition. Practically, the speedup will not work in all cases unless γ is adjusted. (See Choudhury et al. for details[13])

C. 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. In the first step, the cost of all vertex from 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.

IV. Emergency Landing of a Helicopter

In Section III, we described the general planning algorithm. We now present the entire architecture to control and land a helicopter which will use RRT*-AR as one of its key components. We start by framing the planning problem and then propose our approach to solve it.

A. Planning Problem

The planning problem discussed in Section II is now formulated for landing a helicopter after engine failure. We use the autorotation model from Tierney [6]. To provide an idea of the time constraints, the average time for a helicopter to descend from 600 m after engine failure is 60 seconds. During this time safe plans spanning distances of over 2.4 km must be computed. For a safe touchdown, the terminal state of the helicopter has to be on the ground at rest. A continuous landing zone (LZ) feasibility/risk prior is given to the planner to always encourage a path to ground.

B. Approach

Solving the full optimal control problem is computationally expensive and not always numerically stable. Instead, we decompose it into various sub-problems, link them via a state machine and frame it as a multiple objective planning problem (Scherer et al. [14]). This is motivated primarily by the assumption that failure occurs at high altitudes, which results in the planning problem during the flare phase being different from the glide phase in terms of scope, precision and difficulty.

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. [14] 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:

$$\begin{aligned} minimize: \quad & J_i = \int_0^{t_f} c(x(t))dt + c(x(t_f)) \\ constraints: \quad & \dot{x} = f(x(t), u(x,t,P_c), t) \\ & x(0) = x_0 \\ & x(t_f) \in X_{end,i} \\ & g_i \leq 0, J_i < \infty \end{aligned} \tag{4}$$

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 P_c, perform trajectory optimization based on predicted result, and a low-level tracking controller tracks the guess. Table I 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:

1) Entry glide and Entry Flare: Both these states have a simple straight line guess to the desired trim state. The trim tracking controller has to optimize $J_{state}(x_{trim}) + J_u$, thus an LQR controller is used while trajectory optimization optimizes J_{obs}.

TABLE I

PLANNING PROBLEM DEFINITIONS

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

2) Glide: The cost function reflects the desire to minimize risk. Ensuring the pilot has options and there exists feasible backup trajectories, make AR very critical. The RRT*-AR is used to generate initial guesses for this state. In addition to its original definitions, it is given some special attributes. Firstly, the sampling is restricted to the reachability volume Σ_{front} of the vehicle, and biased towards the LZ with a distribution proportional to J_{lz}. Secondly, the command plan is generated in a 3D task space for this 13 degree of freedom system. A variable arc primitive is used to cover a more general spectra of solutions.

3) 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 made 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 [15] 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.

V. 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 3 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 results in Table II, for 791 runs of 5s each with AR parameters ($\varepsilon = 4, \gamma = 0.7, d_{nn} = 500, d_{eq} = 500, \rho = 0.2$) show that 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.

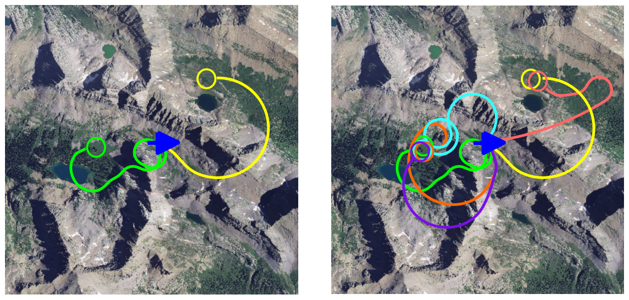

(a) (b)

Fig. 3. 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.

TABLE II

COMPARISON BETWEEN RRT* AND RRT*-AR

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

TABLE III

COST COMPARISONS BETWEEN RRT* AND AFTER SPEEDUPS

	Cost $\tau = 0.6s$	Cost $\tau = 1.0s$	Time to cost = 1.1
RRT*	1.97(±1.28)	1.60(±0.86)	4.40(±1.28)
Approx	1.65(±1.03)	1.37(±0.61)	2.62(±1.07)
Constrained	1.63(±0.84)	1.38(±0.57)	2.89(±0.77)

The acceleration due to both improvements, for 3318 runs till 1000 vertex additions, shown in Table III show the Approximation and Constraint based approaches achieve similar rates of cost reduction (15.4% and 17.3%). Fig 4 shows that the key performance enhancement is because RRT*-AR produces an acceptable path (0.29s and 0.40s) faster than RRT* (0.54s), and maintains a similar rate of cost reduction initially. As per definition, both approaches slow down with increasing vertices and approach the original RRT*. Fig 5 shows that the variance of RRT*-AR reduces sharply before flattening out while RRT* variance goes to zero. This is because RRT*-AR adds vertex to a tree much more rapidly than RRT* at the expense of the approximation error.

The reuse of trees made the system more reliable as shown in Fig 6 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 7 shows the testing of the system in a virtual reality platform with the pilot interface. A video of demonstration of the system can be found in http://y2u.be/AwRhaQtpVxU .

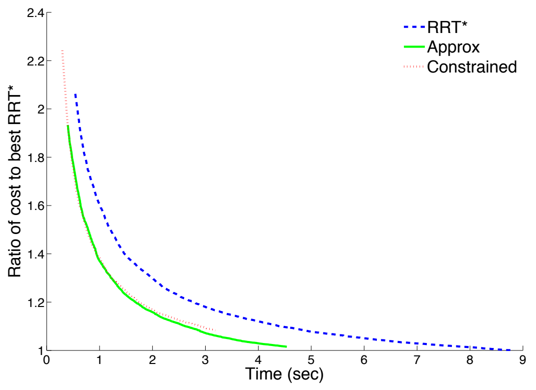


Fig. 4. Accelerated planner costs decay rapidly in the beginning indicating fast exploration of search space and an increased rate of vertex addition.

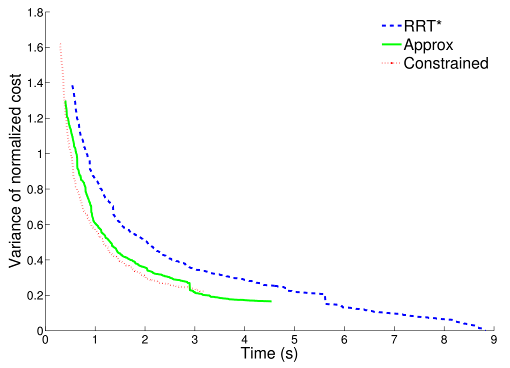


Fig. 5. RRT*-AR rapidly reduces variance before flattening out. This rapidness is credited to rate of vertex addition. The flattening out is due to the approximation error that has been accumulated. RRT* continues to decrease variance to zero.

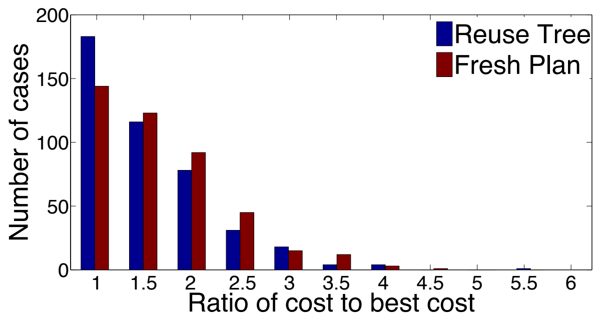
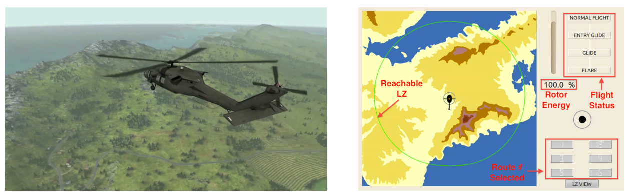


Fig. 6. If the search tree is reused across iterations, more paths are closer to optimal than if a new tree is created at every iteration.



(a) (b)

Fig. 7. Example of the planner in action. (a) Engine failure occurs over a realistic terrain (b) GUI information provided to the pilot

VI. Conclusion

We presented an algorithm for generating alternate routes in real-time. 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] W. Johnson, "Helicopter optimal descent and landing after power loss," 1977.

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

[3] B. Aponso, D. Lee, and E. Bachelder, "Evaluation of a rotorcraft autorotation training display on a commercial flight training device," *Journal of the American Helicopter Society*, vol. 52, no. 2, pp. 123–133, 2007.

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

[5] E. Bachelder and L. Bimal, "Using optimal control for rotorcraft autorotation training," in *Proceedings of the American Helicopter Society 59th Annual Forum, Phoenix, Ariz*, 2003.

[6] S. Tierney, "Autorotation path planning using backwards reachable set and optimal control," Master's thesis, The Pennsylvania State University, 2010.

[7] P. Abbeel, A. Coates, T. Hunter, and A. Ng, "Autonomous autorotation of an rc helicopter," in *Experimental Robotics*. Springer, 2009, pp. 385–394.

[8] K. Dalamagkidis, K. Valavanis, and L. Piegl, "Autonomous autorotation of unmanned rotorcraft using nonlinear model predictive control," in *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*. Springer, 2010, pp. 351–369.

[9] T. Yomchinda, J. Horn, and J. Langelaan, "Autonomous control and path planning for autorotation of unmanned helicopters," in *Proceedings of the American Helicopter Society 68th Annual Forum, Fort Worth, Texas, May 1-3*, 2011.

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

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

[12] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proc. Robotics: Science and Systems*, 2010.

[13] S. Choudhury, S. Scherer, and S. Singh, "Realtime alternate routes planning:the rrt*-ar algorithm," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-12-27, December 2012.

[14] S. Scherer and S. Singh, "Multiple-objective motion planning for unmanned aerial vehicles," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 2207–2214.

[15] M. Stolle and C. Atkeson, "Policies based on trajectory libraries," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 3344–3349.