Multiple-Objective Motion Planning for Unmanned Aerial Vehicles

Sebastian Scherer and Sanjiv Singh

***Abstract*— Here we consider the problem of low-flying rotorcraft that must perform various missions such as navigating to specific goal points while avoiding obstacles, looking for acceptable landing sites or performing continuous surveillance. Not all of such missions can be expressed as safe, goal seeking, partly because in many cases there isn't an obvious goal. Rather than developing singular solutions to each mission, we seek a generalized formulation that enables us to express a wider range of missions. Here we propose a framework that allows for multiple objectives to be considered simultaneously and discuss corresponding planning algorithms that are capable of running in realtime on autonomous air vehicles. The algorithms create a set of initial hypotheses that are then optimized by a sub-gradient based trajectory algorithm that optimizes the multiple objectives, producing dynamically feasible trajectories. We have demonstrated the feasibility of our approach with changing cost functions based on newly discovered information. We report on results in simulation of a system that is tasked with navigating safely between obstacles while searching for an acceptable landing site.**

I. Introduction

Unmanned rotorcraft are being considered for a variety of missions such as continuous surveillance, cargo transport and casualty evacuation in battlefields as well as at sites of natural disaster. As opposed to fixed wing aircraft which typically fly high enough to safely operate blind, low-flying rotorcraft, especially those that must land on unimproved terrain, must fly with onboard perception to enable safe operation. However, since safe landing sites are not readily apparent a priori, they must be actively found by the unmanned rotorcraft. In such cases, the missions requires motion planning that cannot naturally be expressed as safe navigation to a goal point. Instead we need a framework that can combine multiple objectives that might change over the course of a single mission.

Here we propose a framework (Fig. 1) that generalizes from point goals to a more general form of mission specification as objective functions. Examples of objectives are specifications such as the minimum distance to sensed obstacles, smoothness of the path and deviation from desired altitude. The trouble with consideration of a large number of potentially changing "preferences", some of which have hard constraints in addition, is that they are difficult to optimize in realtime.

We present a formalism where elementary objective functions are augmented with additional attributes that enable a compact and consistent specification of planning problems.

Sebastian Scherer and Sanjiv Singh are with the Robotics Institute, Carnegie Mellon University, Pittsburgh. email: [basti, ssingh]@cmu.edu

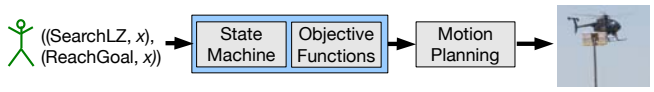


Fig. 1: A high-level overview of the planning framework. The input to the motion planner is a objective function that is set through a state machine based on user commands. The particular missions we address in this paper are "search for landing zones close to x" and "reach goal x."

These attributes are exploited by planning algorithms to increase the planning horizon and plan in real-time.

Below we show some related work, describe the general problem we are solving and then show our approach. Next we show an example of applying the framework to an unmanned helicopter and show results for two different missions.

II. Related Work

Current approaches to unmanned aerial vehicle autonomy either address the problem of obstacle avoidance while reaching a goal or optimize a specifc objective such as search & track. In our own work we have previously addressed reaching a goal point while avoiding obstacles on an RMax helicopter and extend to a more general mission here (Scherer et al.[1]). Search & track methods for UAVs optimize a similar objective as the landing site search application however approaches ignore obstacles and are optimal for a certain task such as the methods by Frew & Elston[2] and Tisdale et al. [3]. Tompkins et al. combined sun exposure and energy to plan a mission plan that reaches a goal point[4]. Cyrill & Grisetti propose a method to explore an environment while also enabling improved localization and mapping [5]. While these approaches optimize one particular problem they do not generalize to several missions.

Real-time motion planning for robotics is an important problem that addresses the problem of which trajectory to select based on partial information. Koenig & Likhachev [6] developed an efficient incremental algorithm to replan kinematic routes once new sensor information is received.

Since routes planned only based on geometric constraints are not necessarily feasible to execute, planning approaches that search a graph over feasible motions have also been developed by Frazzoli et al. [7] using maneuver automata. Hwangbo et al. [8] developed a method to set a kinematically feasible grid that we use in our approach to create a set of initial guesses for trajectory optimization. An A*-like search algorithm that optimizes for multiple objectives has been presented by Wu et al. [9].

Trajectory optimization is used as a step after planning and sometimes sufficient on its own. The goal is to improve

an initial guess to optimize an objective function. It has been incorporated into a planning framework by Brock et al. [10]. Richards & How[11] used mixed-integer linear programming to optimize trajectories as a receding horizon controller for air and spacecraft to avoid obstacles. One method that inspired our trajectory optimization algorithm is the CHOMP algorithm by Ratliff et al. [12]. However, we modified their method to enable distributing the gradient in a more flexible way and we incorporate the dynamics during the optimization.

III. Problem

The problem of planning with changing objective functions is formally defined here. While the formulation below is general, in Sec. IV-D we give the specific details of the problem instance for different missions of an autonomous helicopter.

a) Continuous Definitions: The vehicle state is defined as $\mathbf{x} \in \mathcal{X}$ with $\mathbf{x_0}$ as the initial state and $\mathbf{x_t}$ is the state at time $t \in \mathcal{T} = \mathbb{R}_0^+$. The path points $\mathcal{P} = \mathbb{R}^3 \times \mathbb{R}_0^+$ are a subset of $\mathbf{X}$ that contains the position $\mathbf{p}$ of $\mathcal{X}$ and the magnitude of the velocity s. The trajectory through a set of n points is denoted as $\mathbf{P} = \{\mathbf{p_0}, \ldots, \mathbf{p_n}\}$. The command vector is denoted as $\mathbf{P}_c$ and the resulting vector denoted as $\mathbf{P}_r$. $\mathsf{dist}(\mathbf{p}_1, \mathbf{p}_2)$ is the Euclidean distance between the two points.

b) Discrete Mapping: Finding global solutions in continous problems with many minima is difficult and therefore it is necessary to discretize the problem to plan on a discrete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ representing the state instead. $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges. We can convert between the continous and discrete states with the following functions: $v = \mathsf{node}(\mathbf{x})$, $\mathbf{x} = \mathsf{state}(v)$.

Since we are not guaranteed to have a single goal we define the search of the graph for a path with a set of start vertices $V_s \subseteq \mathcal{V}$ and a set of goal vertices $V_g \subseteq \mathcal{V}$. The transition from one vertex to the next is given by the successor function $V_{succ} = \mathsf{succ}(v)$. The successors are determined by the direction which can be calculated from the state by $\mathbf{d} = \mathsf{dir}(\mathbf{x})$.

The cost of an edge is given by $J(e_i)$ and the cost of a path $\mathbf{B} = \{v_0, e_0, \ldots, v_{n-1}, e_{n-1}, v_n\}$ is given similarly as $J(\mathbf{B})$.

c) Motion Model: The forward motion model is defined as a differential equation as $\dot{\mathbf{x}}(\mathbf{x}, u, t) = f(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t)$ with the command input $\mathbf{u}(\mathbf{x}, t, \mathbf{P}_c)$ defined by the state, time, and command vector $\mathbf{P}_c$. We explicitly specify $\mathbf{P}_c$ since it is the command vector we are optimizing. The initial state is given by $\mathbf{x}(0) = \mathbf{x_0}$ and the equation of motion is integrated as follows:

$$\mathbf{x}(t_f) = \mathbf{x}(0) + \int_0^{t_f} \dot{\mathbf{x}}(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t)dt$$

d) Optimization Problem: A frequently addressed motion planning problem is the two-sided boundary value problem with an initial and final condition. However, we do not necessarily have an end point constraint and consequently the problem is a one-sided optimization problem. The functional is the integrated cost over the length of the trajectory:

$$J(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t_f) = \int_0^{t_f} c(\mathbf{x}(t))dt$$

The general formulation of the optimization problem that needs to be addressed to allow flexible motion planning is to

$$\begin{aligned} \text{minimize:} \quad & J(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P_c}), t_f) \\ \text{subject to:} \quad & \dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & J(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t_f) < \infty \end{aligned}$$

Since there is a constraint that the functional is less than ∞, we have an implicit method of adding constraints from the objective function.

e) Mission State Machine: The operator (or higher level mission planning) can specify the mission through a string over an input dictonary depending on the state different cost functions are selected.

f) Objective Functions: Traditionally an objective function only represents the cost of being in a certain state that is optimized. This limited definition however makes optimizing changing objective functions directly an infeasible problem because we cannot directly find the minima of many cost functions. Since we allow non-continuous cost functions it is not possible to analytically determine the minimum, however we can exploit the augmented representation to directly return the minimum $\mathcal{M} = \{\mathbf{p}_{goal}\}$ without having to check all the values of $c(\mathbf{p})$ over the domain of the cost function. Additionally we define the subgradient of the objective function separate from the cost because a gradient can be defined even if the cost is infinite. This enables a separation of constraints from the gradient. The last method is a prediction action a that can be performed on the objective function which will enable us to increase the planning horizon beyond assuming that the cost functions remain static. More formally we can define the cost functions as a tuple $C = (f,\, g,\, \mathcal{M},\, a,\, \mathcal{D})$ where

- $f := \mathcal{P} \times \mathcal{T} \rightarrow \ \mathbb{R}_0^+$ is the cost of being at a certain position in a certain time.
- $g := \mathcal{P} \times \mathcal{T} \times \mathbb{N} \rightarrow \mathcal{P}$ is the subgradient of g at the position, time, and index of the path.
- $\mathcal{M} := \{\mathbf{p}_m \in \mathcal{D}\}$ is the set of global minima in $\mathcal{D}$.
- $\mathcal{D} := \mathcal{P} \times \mathbb{R}^3$ is the compact set of states with resolution r in each dimension.

For example, cost function addition can then be defined as follows $c' = c_1 + c_2$:

$$c' = (f_1 + f_2, g_1 + g_2, \mathcal{M}_1 \cup \mathcal{M}_2, \{a_1, a_2\}, \mathcal{D}_1 \cap \mathcal{D}_2)$$

IV. Approach

Our approach makes the problem solvable by using a hybrid approach with a discrete initial guess planner and a continuous optimization algorithm. The discrete algorithm searches an approximation and calculates a finite number of

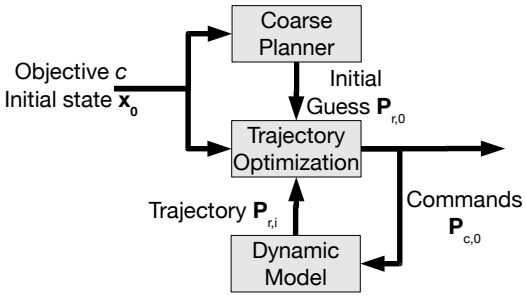


Fig. 2: Detailed planning architecture. The input to the planning algorithm is the objective and the initial state and the output is the command given to the vehicle. In our approach a coarse initial guess is determined based on the minima and cost of the objective and the coarse planner plans in a discretized state space.

distinct initial guesses while the continuous part optimizes within a local pertubation around the guess. Since a trajectory optimization algorithm can fail we always consider a set of initial guesses.

First a coarse-planner is used to create a kinematic initial guess that is resolution-optimal and in the second step a trajectory optimization algorithm optimizes the initial guess based on the cost function gradient and dynamics of the vehicle. The motiviation for using a coarse planner in the first step is that the resulting trajectory will be modified anyway and there is no advantage in incorporating the dynamics at this planning stage.

Minima of a cost function defined over a fixed domain are found easily and are good places to reach because the cost incurred by being at them is minimal since the planning horizon is infinite with non-negative costs. Therefore, an initial guess has to make progress toward a minimum, while the trajectory optimization has to minimize the cost incurred along the way.

The input to the planning algorithm is an initial state $\mathbf{x}(0)$ and an objective c which is a combined expression of objectives. The result after planning is a command trajectory $\mathbf{P}_c$ in the planning domain $\mathcal{D}$ that is given to the vehicle. The trajectory is optimized based on the forward dynamics and the cost function. The overall planning architecture is illustrated in Fig. 2.

A. Problem Approximation

An intuitive method to express the problem is in terms of an optimal control policy which minimzes the cost integral over the planning horizon. However since the cost function c can have many minima and the set of functions $\mathbf{u}$ is infinite we look at a reduced set of parameters $\mathbf{P}_c$. The set of parameters makes the problem feasible since only a finite set of values need to be found. Unfortunately $\mathbf{P}_c$ cannot be precomputed since it is changing in length and a lookup table would be too large for the number of degrees of freedom. We make the problem of finding an initial guess feasible with the following assumptions:

- The minima $\mathcal{M}$ in the cost function c are limited by the domain $\mathcal{D}$ in size and resolution and therefore it becomes feasible to calculate global minima over the finite grid domain.
- Based on the finite domain $\mathcal{D}$ we can define a finite planning graph G to find a best initial guess to start our optimization algorithm.
- Since the planning space of the graph is still large we assume that we always want to reach a minimum in $\mathcal{M}$ because it has the lowest cost if we stay at it for infinite time.
- During planning on the graph G we assume the costs stay fixed to improve planning performance. After reaching the minimum the cost function is updated with the effect of traversing the path $\mathbf{P}_c$.

These approximations of the original problem allow us to find the minima of the cost function in real-time.

Algorithm 1 $\mathbf{P}_c$=TrajectoryOptimization($\dot{\mathbf{f}}, \mathbf{x_0}, c, \mathbf{P}_{r,0}$).
Input: $\dot{\mathbf{f}}$= dynamics, $\mathbf{x}_0$=initial state, c= cost function,$\mathbf{P_{r,0}}$= intial reference trajectory, $\mathbf{P_{c,0}}$= intial command trajectory.
Output: $\mathbf{P}_c$ is the command trajectory

for $i = \{1, \ldots, m\}$ **do**
  $\mathbf{P}_{c,i} = map(\lambda \cdot g_c(\mathbf{P}_{r,i-1})) + \mathbf{P}_{c,i-1}$
  $\mathbf{P}_{r,i} = \int_0^{t_f} \dot{\mathbf{f}}(\mathbf{x}(t), \mathbf{u}(\mathbf{x}(t), t, \mathbf{P}_{c,i})dt$
  $t_i = c(\mathbf{P_{r,i}})$
end for
$b = \text{argmin}_{\forall i \in \{0,\ldots,m\}} t_i$
$\mathbf{P}_c = \mathbf{P}_{c,b}$

B. Trajectory Optimization

Improving a trajectory based on an initial guess has been used in prior work to optimize paths to be dynamically feasible, short, and clear of obstacles. However in our framework the trajectory optimization algorithm actually optimizes over an arbitrary smooth cost objective that is defined as a cost function expression. The trajectory optimization algorithm has to create a dynamically feasible trajectory based on the initial state, the objective, and an initial guess and has to be able to handle a wide variety of objective gradients.

The algorithm for trajectory optimization is shown in Alg. 1. The trajectory optimization starts from an initial guess $\mathbf{P}_{r,0}$ and performs gradient descent until the trajectory does not improve or the maximum number of optimization steps has been exhausted. The command trajectory is updated with the gradient $g_c(\mathbf{P}_{r,i})$ of the cost function via the map function. The map function depends on the application and is a mapping between the changes to the resulting path $\mathbf{P}_r$ and the commands $\mathbf{P}_c$. This mapping could be the Jacobian between the two parameterizations for example. The updated trajectory is predicted with the forward motion model $\dot{\mathbf{f}}$ and the cost of the resulting trajectory is stored. At the end of the optimization the best trajectory $\mathbf{P}_c$ is returned since optimizing the trajectory with respect to the gradient does not necessarily optimize the overall cost and smoothness of the trajectory.

C. Initial Guess Generation

Algorithm 2 $\mathbf{Q}_c$=InitialGuess($G, \mathbf{x}_0$,c)

Input: G=a discrete set of valid motions in $\mathcal{D}$, $\mathbf{x}_0$=initial state, c= cost function

Output: $\mathbf{Q_c} = \{\mathbf{P}_{c,0}, \ldots, \mathbf{P}_{c,m}\}$ is the set of command initial command trajectories.

$\mathbf{Q}_c = \mathsf{recGuess}(0, G, \mathbf{x_0}, c)$

Algorithm 3 $\mathbf{Q}_c$=recGuess($l, G, \mathbf{x}_0$,c)

Input: $l \in \mathbb{N}$ is the level of the recursion,G=a discrete of valid motions in $\mathcal{D}$,$\mathbf{x}_0$=initial state, c= cost function

Output: $\mathbf{Q_c} = \{\mathbf{P}_{c,0}, \ldots, \mathbf{P}_{c,m}\}$ is the set of command initial command trajectories.

$\mathbf{Q}_c = \emptyset$
if $l < l_{max}$ **then**
  $\mathbf{Q}_{t1} = \mathsf{PlanInitialGuess}(G, \mathbf{x}, c) \cup \mathsf{SimpleInitialGuess}(\mathbf{x}, c)$
  for all $\mathbf{P}_{c,i} \in \mathbf{Q}_{t1}$ **do**
    if $J(\mathbf{P}_{c,i}) \neq \infty$ **then**
      if $|\mathbf{P}_{c,i}| > l_{min}$ **then**
        $\mathbf{Q}_c = \mathbf{Q}_c \cup \mathbf{P}_{c,i}$
      else
        $c_{new} = c$
        $a_{c_{new}}(\mathbf{P}_{c,i})$
        $\mathbf{Q}_{t2} = \mathsf{recGuess}(l+1, G, \mathbf{p}_{c,i}[n], c_{new})$
        $j = \text{argmax}_{\forall \mathbf{P}_{t2,i} \in \mathbf{Q}_{t2}} |\mathbf{P}_{t2,i}|$ {(}Reduce search space by greedily picking longest.)
        $\mathbf{Q}_c = \mathbf{Q}_c \cup \mathbf{P}_{t2,j}$
      end if
    end if
  end for
end if

A trajectory optimization algorithm can fail because it optimizes the trajectory in a continuous fashion locally. Especially hard constraints such as obstacles separate the solution space and create minima for the cost functional J. Since there is a possibility that the algorithm can fail we need to be able to recover by considering multiple hypothesis. In the following we describe two methods to initialize the trajectory optimization algorithm.

We want to optimize behavior in the limit and would like to eventually reach global minima. Therefore, a good initial guess should try to reach one of the minima $\mathcal{M} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$. After the plan has reached a minimum it should progress to the next minimum, if the action prediction removes the current minimum. The action prediction allows us to increase the planning horizon by predicting the outcome of each path segment. The next path segments are found recursively. The algorithm starts with a call to recGuess in Alg. 2. The recGuess recursively explores the outcome of predicting actions to increase the planning horizon in Alg. 3.

The first algorithm is the "simple" initial guess that just connects a start configurition $\mathbf{x}_0$ with the a minimum in $\mathbf{p}_i \in \mathcal{M}$ and is always available. The shape of this kind of initial guess depends on the motion model and in Sec. IV-D we show the initial guess function simple($\mathbf{x}$,$\mathbf{p}_i$) was implemented for the example. The guess does not consider the cost of the trajectory and therefore can intersect with obstacles.

Algorithm 4 $\mathbf{Q}_c$=SimpleInitalGuess($\mathbf{x_0}, c$).

Input: $\mathbf{x}_0$=initial state, c= cost function

Output: $\mathbf{Q_c} = \{\mathbf{P}_{c,0}, \ldots, \mathbf{P}_{c,m}\}$ is the set of command initial command trajectories.

$\mathbf{Q}_c = \{\}$
for all $\mathbf{P}_i \in \mathcal{M}_c$ **do**
  $\mathbf{Q}_c = \mathbf{Q} \cup \mathsf{simple}(\mathbf{x_0}, \mathbf{P_i})$
end for

Algorithm 5 $\mathbf{P}_c$=PlanInitialGuess($G, \mathbf{x}_0$,c)

Input: G=a discrete of valid motions in $\mathcal{D}$,$\mathbf{x}_0$=initial state, c= cost function

Output: $\mathbf{Q_c} = \{\mathbf{P}_{c,0}, \ldots, \mathbf{P}_{c,m}\}$ is the set of command initial command trajectories.

$v_0 = \mathsf{node}(\mathbf{x}_0), d = \mathsf{dir}(\mathbf{x}_0), \mathbf{Q} = \emptyset$
$(g_f, v_f) = \mathsf{astar}(\{v_0, d\}, \mathsf{node}(\mathcal{M}_c))$
$(g_b, v_b) = \mathsf{astar}(\mathsf{node}(\mathcal{M}_c), \{v_0, d\})$
$\mathbf{B}_0 = \mathsf{createPath}(v_b, g_f)$
$\mathbf{Q} = \mathbf{B}_0$
if $\mathbf{B}_0 \neq \emptyset \wedge J(\mathbf{B}_0) \neq \infty$ **then**
  $g = g_f + g_b$
  $c = 0, i = 0$
  while $|Q| < n_{desiredpaths}$ **do**
    $v = \mathsf{samplePoint}(c++)$
    if $g(v) \geq J(\mathbf{B}_0) \wedge g(v) \leq \alpha J(\mathbf{B}_0)$ **then**
      $B_i = \mathsf{createPath}(g_f, v) \cup \mathsf{reversechop}(\mathsf{createPath}(g_b, v))$
      if $\mathbf{B}_i \cap \mathbf{Q} < \gamma \wedge |\mathbf{B}_i| < (1+\epsilon)|\mathbf{B}_0|$ **then**
        if $\mathsf{locallyOptimal}((B)_i)$ **then**
          $i++$
          $\mathbf{Q}' = \mathbf{Q} \cup \mathbf{B}_i$
          $\mathbf{Q}'_c = \mathbf{B}_c \cup \mathsf{state}(\mathbf{B}_i)$
        end if
      end if
    end if
  end while
end if

The second algorithm searches the planning graph of valid trajectories. One could find the optimal using a simple A* search. However, since we want to calculate a *set* of initial guesses we formulate the problem as an alternative route planning problem. Our approach is similar to Abraham et al.[13] adapted to a grid planning scenario.

An admissible alternative path needs to share as little as possible with the prior paths so the length that the alternative path $\mathbf{B}_i$ shares with the optimal path $\mathbf{B}_0$ should be small compared to the length of the optimal path. We also want to share as little as possible with other alternative paths that might be added later to get a large set of alternative routes and therefore the sharing between prior alternative routes also has to be small.

Each subpath of the alternative route has to be locally-optimal path which we can call T-locally optimal. The third condition is defined as uniformly bounded stretch. This refers to the fact that every increase in the total path cost should also be only a small fraction of the optimal path cost to avoid unnecessary detours. The stretch can be defined with every

subpath having a stretch of at most $(1+\epsilon)$. Additionally since two parallel grid paths are essentially the same except offset by one grid paths we require at least one node to be at least β cells away. To summarize we define a path to be admissible if

1) $l((\mathbf{B}_0 \cup \ldots \cup \mathbf{B}_{i-1}) \cap \mathbf{B}_i) \leq \gamma \cdot |P_0|$ (Limited sharing of new alternative and previous alternative paths);
2) $\mathbf{B}_i$ is T-locally optimal for $T = \alpha \cdot J(P_0)$;
3) $\mathbf{B}_i$ is $(1+\epsilon)$-UBS (Uniformly bounded stretch);
4) $\max(\min(\mathsf{dist}(\mathbf{p}_k, \mathbf{p}_j)) \geq \beta, \forall k \in \mathbf{B}_i \wedge \forall j \in \mathbf{B}_0$ (Minimum distance separation)

The set of alternative admissible paths is still very large and to enable a real-time algorithm let us consider a subset of the admissible paths called *single via paths.* These are paths that go through an intermediate vertex v and are the concatenation of shortest paths from $s - v$ and $v - t$.

The algorithm (Fig. 5) first computes a bi-directional A* cost grid up to the closest goal from the current location to the minima and for the minima to the start. The result are the forward cost grid g_f and the backward cost grid g_b that give the cost from each location to the start and the minima respectively. If an optimal route exists we calculate the joint cost grid $g = g_f + g_b$. Now we approximate condition 4 by sampling points with a quasi-random sequence [14]. The probability of picking vertices that are adjacent to a previous path is low because the next sample will be as far away from the previous samples as possible.

If the picked sample cost is less than $g(v) \leq \alpha J(\mathbf{B}_0)$ a new path is created. The new path is then checked for sharing (condition 1) and if the uniformly bounded stretch is within $(1+\epsilon)$ (condition 3) . Since we need to check if the created path is T-locally optimal (condition 2) we check if the shortest path that starts halfway between $s - v$ and ends halfway between $v - t$ is at most T more expensive. It is a reasonable assumption to only check this subpath, because the only place the detour can occur is somewhere around v . If the path fulfils the conditions it is added to the set $\mathbf{Q}$ of paths that is checked for sharing.

The algorithm PlanInitialGuess will conduct a resolution-optimal and complete search between $\mathbf{x}_0$ and the minimum in c and at least the optimal path if it exists will be returned. Since the locally-optimal and minimum distance separation conditions have to be approximated it is possible that some non admissible paths might be admitted. However the likelihood is low and the only impact is an increase in computation time for the trajectory optimization.

D. Autonomous Helicopter Example

In this section we consider the example of emergency medical evacuation as an application of the planning algorithm. The problem of landing at an unknown location is difficult to execute efficiently because the cost functions such as obstacles and landing sites are discovered in real-time and are changing. Therefore, we have to replan continuously based on the current discovered state of the environment.

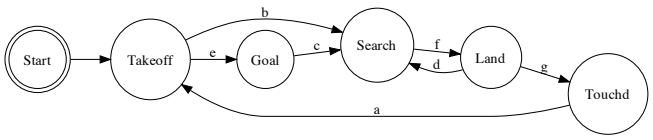


Fig. 3: The state machine for the example unmanned helicopter. The available missions are takeoff, search for landing sites, land, and reach a goal point. The state transitons depend on events and the current cost.

1) Defined Missions: The state machine that we consider in this problem switches between takeoff, reaching a goal, search for landing zones (LZs), landing, and touchdown and is shown in Fig. 3. The transition between the different states depends on the input string given by the user and robot internal strings. For each state a different set of objective functions will be used.

The cost function c_s that is given to the planning algorithm represents the overall mission, expressing the cost and trajectory energy. The function is rescaling the gradient up to a limit to equalize the weights of the smoothed combined cost $c_{combined}$ and the energy cost function c_{energy} :

$$c_{s,} = \text{limit}(\text{smooth}(c_{combined}, \mathbf{W}), w_1) + \text{limit}(c_{energy}, w_2)$$

The combined function contains the cost function selector based on the state machine H which selects from the sub objective functions described below. Each of these objective functions is enabled if the state machine is in a certain state S:

$$c_{combined} = c[H] = \{c_{to}, c_{td}, c_{app}, c_{lz}, c_{oa}\}[H]$$

For each mission state the cost function expression that is optimized is different. For example**,** for landing site search we have the following objective:

$$c_{lz} = \min(w_1 c_{obst} + w_2 c_{info} + w_3 c_{alt})$$

where c_{obst} is the obstacle objective, $c_{info}$is the information gain, c_{alt} is the desired altitude above ground. Each objective consists of a tuple such as $c_{obst} = (g_{obst}, f_{obst}, M_{obst}, a_{obst}, D_{obst})$:

- $obst(\mathbf{P}) = \max(0, d^2_{max} - d(\mathbf{P}))$ where $d(\mathbf{P})$ is the distance to the closest obstacle.
- $g_{obst} = \nabla obst(\mathbf{P})$ where $obst$ is the obstacle objective function,
- $f_{obst} = \begin{cases} obst(\mathbf{P}) \\ \infty \qquad \text{if } d^2(\mathbf{P}) < d^2_{obst} \end{cases}$,
- $M_{obst} = \emptyset, a_{obst} = \emptyset$, $\mathcal{D}_{obst}$ is the domain and resolution of the grid.

The c_{info} cost function represents the current knowledge about landing sites and changes as the vehicle gets feedback on the availability of landing sites. The algorithm used for landing site evaluation is based on our prior work [15]. If an area has been explored it gets a low information gain and

unknown areas receive a high information gain. The landing site evaluation has a ground goal it needs to find a path to which biases the search around this area. The prediction action a_{info} assumes that by traversing an area any landing sites in that area becomes known.

2) Vehicle Model: The command trajectory $\mathbf{P}_c$ we would like to output is smoothed during the trajectory optimization using two techniques. One technique adds a smoothness objective prior to the cost function. This energy$(\mathbf{P})$ objective tries to minimize deviation from a straight line.

The other technique is to use a smoothing matrix to distribute the update of the cost function gradient across the path. This is desirable because it equalizes quickly changing gradients, smoothes the trajectory and introduces a prior on how fast the trajectory should be updated. There are many possible matrices that could be considered for distributing the update of the gradient. One such matrix is the inverse of the gradient of the trajectory as shown in [12]. However the inverse affects all axes and does not decouple the update across coordinates. Instead we choose a distribution function that is similar in shape to a Gaussian in that it decays rapidly. The matrix $\mathbf{W}$ that distributes the gradient can be defined as follows for $i \in \{1, n\} \wedge j \in \{1, m\}$ with radius r:

$$\mathbf{W}(i,j) = \frac{1}{2} + \frac{r}{2} \begin{cases} \sqrt{d_{v1}} & d_{v1} \geq 0 \\ \sqrt{d_{v2}} & d_{v2} \geq 0 \wedge j \geq (i - 2r) \\ \sqrt{d_{v3}} & d_{v3} \geq 0 \wedge j \leq (i + 2r) \end{cases}$$

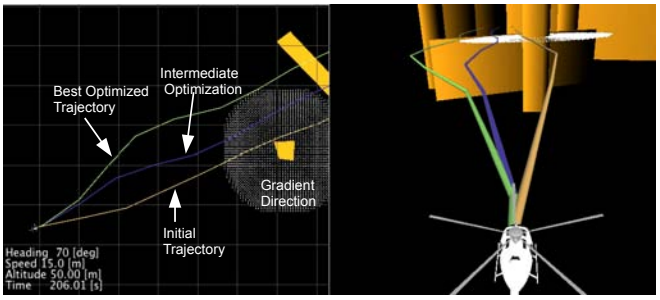
where

$$d_{v1} = r^2 - (j-i)^2,\ d_{v2} = r^2 - (j-i-2r)^2,\ d_{v1} = r^2 - (j-i+2r)^2$$

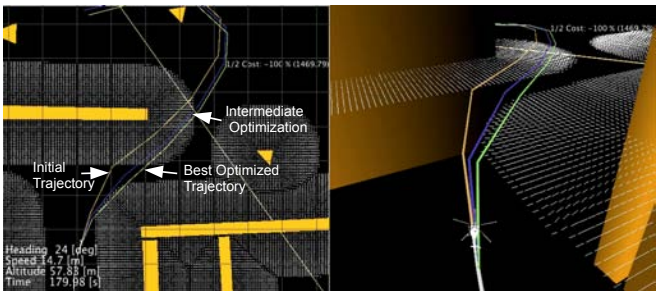
3) Trajectory Optimization: We discretize the trajectory by a set of n 3-D points $\{\mathbf{p}_0, \ldots, \mathbf{p}_n\}$ with associated speed s_i that can be concatenated into one vector $\mathbf{P_c}$ containing the trajectory. Initially the algorithm is given the simulated resulting trajectory $\mathbf{P}_r$ and the initial dynamic state $\mathbf{x}_0$. For the example considered here the command parameters correspond closely to the resulting trajectory and therefore the mapping from parameters to the path (map) is a direct passthrough. In our experience the non-linearity of the problem makes applying the Jacobian pseudo inverse or transpose the path mapping unstable in the optimization and we were able to achieve better results by directly updating the path.

4) Initial Guess Model: A good initial guess is a guess that can be followed closely with the path controller. We defined two algorithms to create the initial guess from a set of minima $\mathcal{M} = \{\mathbf{p}_{m,1}, \ldots, \mathbf{p}_{m,n}\}$. The simple initial guess connects the start to the goal location with a one-sided Dubin's curve which only consists of a circle segment and a straight line to the goal location.

Traditionally in 2D grid search an 8-connected grid is considered to plan the motion of a vehicle. However regular grid paths ignore kinematic constraints and are therefore not executable by a helicopter with significant forward velocity.

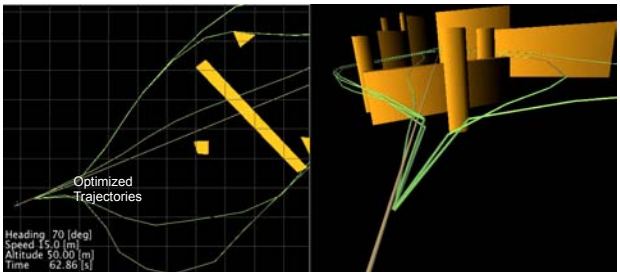


(a)

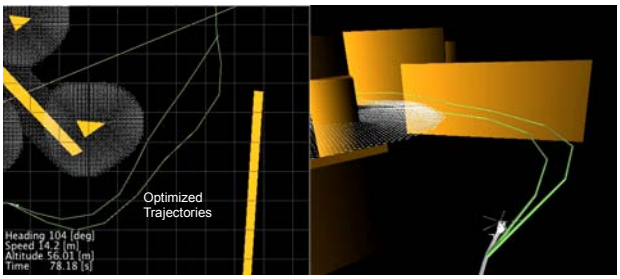


(b)

Fig. 4: Examples of the trajectory optimization algorithm avoiding obstacles. In (a) the simple initial guess trajectory does not use the coarse planner and therefore only the optimization moves the trajectory to avoid collision. In (b) the coarse planner is used to plan an initial guess, however the path gets close to some obstacles. After optimization the clearance is larger. (Orange = obstacles, green = best trajectory, orange = initial guess. purple = intermediate optimization step.)



(a)



(b)

Fig. 5: Using the planned initial guess for trajectory optimization. (a) shows a set of optimized trajectories based on the set of initial guess plans. If there are no other better initial guesses found by the planner such as in (b) only the simple and one planned initial guess are optimized.

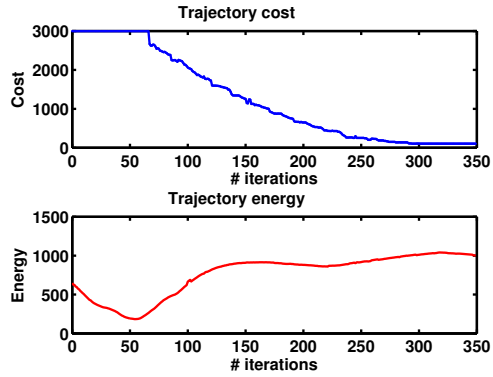


Fig. 6: Cost and energy per iteration for one planning cycle. The initial guess is moved outside the obstacle during the optimization since the simple initial guess path intersected an obstacle. The optimization could be interrupted after the obstacle is cleared. The setup is shown in Fig. 4a.

Instead we adopt the idea of [8] to plan in a kinematically-feasible search grid. Additionally to the grid position each cell also has a notion of the parent cell which allows us to plan in a discretized position and direction space (x, y, z, θ, ϕ). The angles are corasely discretized and the allowable node expansions are set in conjunction with the cell resolution.

V. Experiments

In the following we show results of using the multiple-objective motion planning algorithm to plan routes for an autonomous helicopter in simulation with two different cost functions. The first experiment shows avoiding an unknown obstacle, and the next experiment searches for landing sites.

The robot is assumed to be equipped with a downward looking range sensor for landing zone evaluation and a forward looking range sensor for obstacle avoidance with a range of 150 meters. All computation was performed with an Intel 2.6 GHz Core 2 Quad computer. Prior knowledge of a digital elevation model for altitude cost was given to the vehicle.

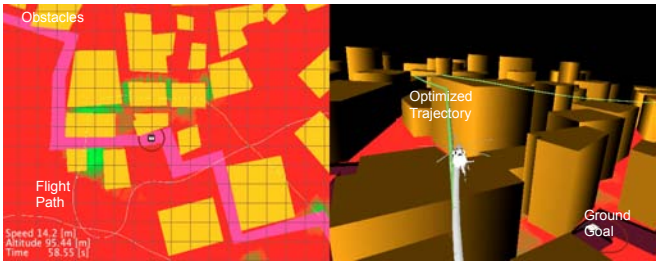
A. Reach a Goal Point

The first experiment uses the obstacle avoidance objective to demonstrate different aspects of the planning algorithm. Initially we consider the case of only using the simple initial guess to illustrate the trajectory optimization algorithm. In Fig. 4a one can see how the trajectory optimization algorithm moves the trajectory to minimize the cost of being close to an obstacle while keeping the desired altitude. The trajectory only climbs if necessary since there is an objective to keep a desired altitude.

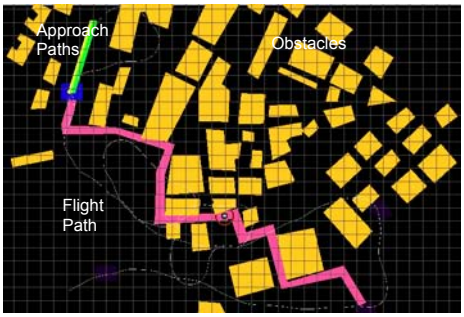
The initial guess is close to the obstacle and a large gradient pushes the trajectory to the left. Since the trajectory is distributed along the gradient with the smoothing function the whole trajectory is moved to avoid the obstacle. The best trajectory avoids all the obstacles and has no cost and a slightly higher energy cost because the trajectory is not a straight line anymore.

In Fig. 6 one can see how the obstacle cost and energy cost evolve as the algorithm optimizes the trajectory. Initially

(a)



(b)



(c)

Fig. 7: Manhattan environment landing site search. (a) shows the problem setup. A set of buildings similar to Manhattan presents obstacles and occludes the range sensor. (b) demonstrates the occlusions. (c) shows the overall search and approach path.

since the trajectory intersects with an obstacle the cost is maximal. As the shape of the trajectory changes the cost decreases towards the minimum. The energy of the trajectory decreases from the initial cost because the initial guess is only kinematically feasible. As the path moves out of the obstacle region the energy used in the trajectory increases because it has to bend.

Next we enable the coarse initial guess planner which will avoid obstacles. However as shown in Fig. 4b the resulting initial trajectory is not necessarily safe because it is only a kinematically feasible plan. After optimizing the trajectory the best path avoids high cost zones close to obstacles.

Since we cannot guarantee that the optimization will converge to a solution we want to optimize a set of initial guesses (Fig. 5a). One guess is always a simple guess that moves through an obstacle while the other initial guesses avoid the larger obstacles in the center either to the left or right. One can see that the alternative route planning algorithm picked a set of significantly different trajectories that can be optimized. If there are no better alternatives

Initial Guess		Optimization		Overall
.53(±.37)		.10(±.04)		.62(±.48)
Minima	Planner	Gradient	Dynamics	
.20(±.06)	.33(±.33)	.02(±.02)	.08(±.02)	

TABLE I: Median and standard deviation computation time for landing site search in seconds.

only one will be found as shown in Fig. 5b where only one planned and one simple initial guess were found.

B. Search for Landing Sites

A challenging example for landing site search is shown in Fig. 7. The vehicle has to find a good landing site and is only given the final ground goal shown in Fig. 7a and false prior knowledge of potential sites in dark purple. The obstacles, ground path and good landing zone are unknown and need to be discovered as the environment is explored. Any valid landing site needs to have a ground path to the goal and the only ground path is shown in magenta and the only valid landing zone is shown in blue. Additionally since the range sensor is occluded from the high buildings it is sometimes necessary to perform multiple passes over an area to explore. The false prior knowledge of landing sites could be due to the fact that it was a previously known to-be-good location to land and now a disaster has struck and no more good locations are known.

A snapshot during exploration is visible in Fig. 7b. One can see that there are holes in the exploration coverage because the range sensor was occluded by the buildings. The green spots indicate that there are frontiers that have not been sensed before. Consequently when we consider the complete search path the vehicle takes some loops to complete the coverage of the area (Fig. 7c).

The calculation times for landing zone search are presented in Tab. I. The data was taken over one landing site search run and shows that the worst case planning time is large (~3000ms) while the median planning time is about 620ms. The largest contribution of this variance is due to the coarse initial guess planner. Since it sometimes has to explore a larger region of the motion planning graph the planning time will increase. The trajectory optimization time on the other hand has a smaller variance with a median of about 100ms because it performs a fixed number of iterations. The current implementation first runs the coarse initial guess planner followed by the trajectory optimization algorithm. In future work the algorithm could be optimized to perform the optimization more frequently while the coarse initial guess planner runs less frequently to reduce reaction time.

VI. Discussion

We presented a framework for expressing higher-level missions with a set of changing objective functions and conveyed a planing algorithm that is suitable for to find routes in real-time. We were able to approximate the problem by exploiting the additional information in the augmented cost functions.

In summary the main contributions of this paper are:

- a formalization of the problem of motion planning with multiple changing objectives,
- a definition of augmented cost functions that enables planning with a longer horizon due to action prediction,
- a decomposition of the problem into a local and coarse planning architecture to enable real-time planning.

The simulation results demonstrated the effectiveness of our algorithm in a variety of environments for avoiding obstacles, and performing landing zone search. The motion planning algorithm is mission agnostic and switches between multiple different objectives using the mission state machine.

In future work we intend to test the algorithm on different dynamical models, and with different parametizations to test the generalization of the approach. We intend to formally define the set of admissable cost functions and increase the set of base cost functions to enable optimizing larger cost function expressions on more complicated missions. So far we have considered only orthogonal objectives which simplifies setting the weights. However, in future work we would like to use competing objectives and learn the set of optimal weights based on self-supervised learning with a meta-objective function.

References

[1] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying Fast and Low Among Obstacles: Methodology and Experiments," *The Int. J. of Robotics Research*, vol. 27, pp. 549–574, May 2008.

[2] E. Frew and J. Elston, "Target assignment for integrated search and tracking by active robot networks," in *Proc. ICRA*, pp. 2354–2359, 2008.

[3] J. Tisdale, Z. Kim, and J. Hedrick, "Autonomous UAV path planning and estimation," *Robotics & Automation Magazine, IEEE*, vol. 16, pp. 35–42, June 2009.

[4] P. Tompkins, T. Stentz, and W. Whittaker, "Mission planning for the Sun-Synchronous Navigation Field Experiment," in *Proc. ICRA*, pp. 3493–3500, IEEE, Aug. 2002.

[5] C. Stachniss and G. Grisetti, "Information gain-based exploration using rao-blackwellized particle filters," in *Proc. of Robotics: Science and Systems*, 2005.

[6] S. Koenig and M. Likhachev, "D* Lite," in *Eighteenth national conference on Artificial intelligence*, July 2002.

[7] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics and Automation*, vol. 21, pp. 1077–1091, Dec. 2005.

[8] M. Hwangbo, J. Kuffner, and T. Kanade, "Efficient Two-phase 3D Motion Planning for Small Fixed-wing UAVs," in *Proc. ICRA*, (Rome), pp. 1035–1041, Mar. 2007.

[9] P.-Y. Wu, D. Campbell, and T. Merz, "Multi-Objective Four-Dimensional Vehicle Motion Planning in Large Dynamic Environments," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, no. 3, pp. 621–634, 2011.

[10] O. Brock and O. Khatib, "Elastic strips: A framework for integrated planning and execution," *Experimental Robotics Vi*, vol. 250, pp. 329–338, Jan. 2000.

[11] A. Richards and J. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the American Control Conference (ACC)*, Jan. 2002.

[12] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient Optimization Techniques for Efficient Motion Planning," in *Proc. ICRA*, (Kobe), pp. 489–494, Mar. 2009.

[13] I. Abraham, D. Delling, A. Goldberg, and R. Werneck, "Alternative Routes in Road Networks," in *Proc. 9th International Symposium on Experimental Algorithms (SEA)*, Jan. 2010.

[14] H. Niederreiter, "Low-discrepancy and low-dispersion sequences," *Journal of Number Theory*, vol. 30, pp. 51–70, Sept. 1988.

[15] S. Scherer, L. Chamberlain, and S. Singh, "Online Assessment of Landing Sites," in *AIAA Infotech@Aerospace*, (Atlanta), Apr. 2010.