Low-Altitude Operation of Unmanned Rotorcraft

Sebastian Scherer

May 9, 2011

CMU-RI-TR-11-03

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee:
Sanjiv Singh (chair), Carnegie Mellon University
Takeo Kanade, Carnegie Mellon University
Alonzo Kelly, Carnegie Mellon University
Emilio Frazzoli, Massachusetts Institute of Technology

To my family

Abstract

Currently deployed unmanned rotorcraft rely on preplanned missions or teleoperation and do not actively incorporate information about obstacles, landing sites, wind, position uncertainty, and other aerial vehicles during online motion planning. Prior work has successfully addressed some tasks such as obstacle avoidance at slow speeds, or landing at known to be good locations. However, to enable autonomous missions in cluttered environments, the vehicle has to react quickly to previously unknown obstacles, respond to changing environmental conditions, and find unknown landing sites.

We consider the problem of enabling autonomous operation at low-altitude with contributions to four problems. First we address the problem of fast obstacle avoidance for a small aerial vehicle and present results from over a 1000 runs at speeds up to 10 m/s. Fast response is achieved through a reactive algorithm whose response is learned based on observing a pilot. Second, we show an algorithm to update the obstacle cost expansion for path planning quickly and demonstrate it on a micro aerial vehicle, and an autonomous helicopter avoiding obstacles.

Next, we examine the mission of finding a place to land near a ground goal. Good landing sites need to be detected and found and the final touch down goal is unknown. To detect the landing sites we convey a model based algorithm for landing sites that incorporates many helicopter relevant constraints such as landing sites, approach, abort, and ground paths in 3D range data. The landing site evaluation algorithm uses a patch-based coarse evaluation for slope and roughness, and a fine evaluation that fits a 3D model of the helicopter and landing gear to calculate a goodness measure. The data are evaluated in real-time to enable the helicopter to decide on a place to land. We show results from urban, vegetated, and desert environments, and demonstrate the first autonomous helicopter that selects its own landing sites.

We present a generalized planning framework that enables reaching a goal point, searching for unknown landing sites, and approaching a landing zone. In the framework, sub-objective functions, constraints, and a state machine define the mission and behavior of an UAV. As the vehicle gathers information by moving through the environment, the objective functions account for this new information. The operator in this framework can directly specify his intent as an objective function that defines the mission rather than giving a sequence of pre-specified goal points. This allows the robot to react to new information received and adjust its path accordingly. The objective is used in a combined coarse planning and trajectory optimization algorithm to determine the best path the robot should take. We show simulated results for several different missions and in particular focus on active landing zone search.

We presented several effective approaches for perception and action for low-altitude flight and demonstrated their effectiveness in field experiments on three autonomous aerial vehicles: a 1m quadrocopter, a 3.6m helicopter, and a full-size helicopter. These techniques permit rotorcraft to operate where they have their greatest advantage: In unstructured, unknown environments at low-altitude.

Acknowledgments

This thesis would not have been possible without the many people who have influenced, supported, and guided me along the way. They are numerous and it will be inevitable that I have forgotten somebody that deserves mention and I thank the forgotten in advance.

First I would like to thank Sanjiv Singh for believing in me and his support in my research, guidance in the process, and insight in Robotics. I would also like to thank my committee members: Takeo Kanade for his insight in technical as well as presentation matters, Alonzo Kelly for valuable discussions about planning and guiding my research in a good direction, and Emilio Frazzoli for being supportive of my work and valuable advice. Lyle Chamberlain has been invaluable in building the hardware that made this thesis possible, and especially for being a good friend. I would also like to thank Roy Maxion who brought me to Carnegie Mellon and gave me the freedom to explore Robotics. I would also like to thank Red Whittaker for his inspiration and support that got me into field robotics.

My research experience was enjoyable thanks to all the interactions with great people around me: Ben Grocholsky for his philosophical insights into robotics, valuable feedback and help with experimentation, Chris Geyer for explaining me the intrinsics of vision and the moment based-plane fitting code. Stephen Nuske for his feedback on the thesis. Bradley Hamner in working with Dodger, and useful comments on my presentations. Wenfan Shi for help with systems, linux, and driver development. Prasanna Velagapudi for help in simulation matters and feedback. Maggie Scholtz for the mechanical design on the airrobot mounting and great pictures of the vehicle. Sanae Minick for help with my many requests and Suzanne Lyons-Muth for guiding me along the way of the details of the PhD process and Reid Simmons for interesting discussions during the student lunches.

I would like to show my thanks to Fred Heger for being a good office-mate and friend. Thanks for your constructive comments and helping with odd requests: Debadeepta Dey, Joseph Djugash, Young-Woo Seo. I was finally drawn into robotics during the Grand Challenge and the great team motivated me. I would also like to thank the Urban Challenge team for their support.

I owe my deepest gratitude to my family that has enabled me to pursue my passion. I would like to thank my wife Hiroko for supporting me during long hours and for believing in me; my daughters Lara, and Amelie for being the best daughters in the world. Bine, and Papi I thank you for letting me pursue my dream and supporting me along the way. For his advice on graduate life I would like to thank Franz Scherer Jr. I would also like to thank Yoshie and Yoichiro Sugino for their support.

A lot of people enabled our experiments on the Yamaha RMax and I would like to thank: Samar Dajani-Brown, Mike Elgersma, Gary Stevenson, Henele Adams, Alan Touchberry, Tara Schesser, Robert Schley, Mark Delouis, and Brad Looney. For our experiments on the airrobot I would like to thank: Thomas Meyer, Burkhard Wiggerich, and Fabian Laasch from Airrobot, and Sup Premvuti from Hokuyo. I would also like to thank STAT Medevac for making their helicopter (EC 135) available. The experiments on the Unmanned Little Bird helicopter would not have been possible without the support of Piasecki Aircraft Corp: Fred and John Piasecki, Chris Chiechon, and Buzz Miller. I would also like to thank the team at the Boeing Company in Mesa, AZ that made the tests possible: Don Caldwell for his patience in integrating our software, and Lindsay Fry-Schallhorn for conducting the flight tests of our system, Christine Cameron, Tim Gleason, Jason Graham, Roger Hehr, David Guthrie, Jack Gray, Dino Cerchie, and Mark Hardesty for enabling the experiments.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Today the threat of low-altitude obstacles constrain the large number of fielded unmanned aerial vehicles (UAVs) to operate at high altitude, or under close human supervision at low altitude. Safe autonomous flight at low altitude is essential for widespread acceptance of aircraft that must complete missions close to the ground, and such capability is widely sought. For example, search and rescue operations in the setting of a natural disaster allow different vantage points at low altitude. Likewise, UAVs performing reconnaissance for the police, the news or the military must fly at low-altitude in presence of obstacles. Large rotorcraft can fly at high altitude, however have to come close to the ground during landing and takeoff. So far unmanned full-scale rotorcraft have to land at prepared land at prepared sites with prior knowledge of obstacle-free trajectories.

Teleoperation with camera feedback close to obstacles is currently used, however it is difficult because the operator needs a good situational awareness of obstacles and the vehicle state with high frequency. Situational awareness is difficult to achieve since camera feedback does not allow gauging the distance to obstacles and small obstacles such as wires are difficult to detect. In a rotorcraft the range of possible motion also requires a large field of view and requires disambiguating translational from rotational motion. The communication of this flight critical information requires low latency, high bandwidth, reliable communication. The methods developed in this thesis could be used to enable advanced teleoperation or help augment pilot operations.

In this work we assume that the vehicle is given a mission by the operator through partially known terrain that is executed completely autonomously with no further input and therefore after launch no more communication is required. Autonomous navigation requires the vehicle to be be aware and react to obstacles in the flight path, and land in unknown terrain. Since, it cannot be assumed that reliable communication is available the aircraft should behave according to the users intentions.

1.1 Problem

This thesis presents real-time algorithms that enable low-altitude missions of UAVs in previously unknown environments completely autonomously. The problem we are addressing is challenging because we have to plan and account in a complex environment for changing knowledge of obstacles, landing sites, and other information such as wind on the ground. In general, there is a large number of objectives and constraints that need to be considered during planning for an unmanned aerial vehicle. An overview of the problem is given in Fig. 1.1 and can be characterized as follows:

Unknown Obstacles The UAV operates close to the ground and needs to react to obstacles in realtime. Obstacles that are relevant for rotorcraft include wires, radio towers, vegetation, hills, buildings, and other man-made structure. It is difficult to operate close to the ground because it is necessary to sense and avoid very small obstacles like wires in a timely manner. At the same time, it is required to sense a large field of view to avoid myopic behavior. The environment is coarsely known or completely unknown and the vehicle needs to react to the terrain as it is discovered. We assume that the robot is equipped with a range sensor that provides geometric information and a pose sensor (GPS/INS) for state.

Discover Landing Sites An important advantage of rotary wing aerial vehicles is the ability to land at unimproved landing sites. However, finding these sites is difficult because helicopters have many constraints on landing sites. The slope must be small, the terrain smooth, there has to be an approach and abort path with appropriate glide slope, and the site has to be reachable from the location one is trying to reach to on the ground. Before the robot can approach a site, it is necessary

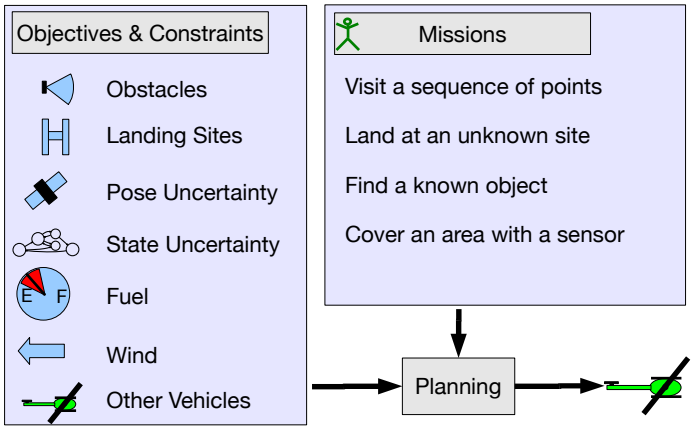


Figure 1.1: The general problem of navigating at low altitude. An aerial vehicle is expected to be capable of performing a variety of interesting missions while at the same time respecting a set of partially known objective functions and respect dynamic and obstacle constraints for successful mission completion.

to find the landing site in the shortest time possible. We also want to incorporate prior information about potential landing sites to shorten the search time. The challenging problem of searching and landing at previously unknown landing sites in real-time is a benchmark task for evaluating our framework and algorithms.

Rotary Wing Vehicles In this work, we will focus on rotorcraft of various sizes that we assume to be stable and can be controlled via higher level commands. Although, the dynamics of the uncontrolled vehicle are complicated we focus on respecting the dynamics on the commanded path for a controlled vehicle.

1.2 Thesis Statement

Enabling safe autonomous operation requires the development of methods that enable vehicles to operate at low altitude. In this thesis I examine several problems and demonstrate the approaches for small (<1m), medium, and full-scale (manned) rotorcraft flying at low-altitude. The methods are developed by considering different objectives. Initially the work focussed on only considering goal points and obstacles as the objectives and I developed two approaches: Fast obstacle avoidance with late detection of small obstacles, and efficient calculation of obstacle cost for faster global motion planning. To enable a larger self-awareness of the vehicle and to permit operation close to the ground I developed a method to evaluate unprepared landing sites, and generalize the planning problem to a set of objective functions. We test this generalized planning approach for landing site search. This thesis makes the following thesis statement:

> *In low-altitude environments safe and efficient trajectories for aerial vehicles can be constructed by combining and sequencing a set of simpler objective functions.*

The framework developed in this thesis has a wide range of applications and could be applied to other classes of aerial platforms such as fixed wing aircraft. The algorithms developed advance the field of real-time perception and motion planning in realistic applications of unmanned rotorcraft. In the future, I expect that this work will extend the envelope of missions where unmanned aerial rotorcraft can be used.

1.3 Overview

The work is put in context in Chapter 2. Small aerial vehicles have to react quickly to new obstacle and Chapter 3 presents a reactive approach coupled with higher level planning to allow operation among obstacles even if the dynamics of the aerial vehicle cannot be predicted with certainty.

Chapter 4 addresses a particular bottleneck of route planning in a 3D grid for aerial vehicles. Obstacles need to be expanded to achieve a high safety margin. Calculating the cost of obstacles depends on the distance to the closest obstacle and can be efficiently updated using the LIDT algorithm.

A UAV has to land eventually and finding landing sites is particularly relevant for unmanned rotorcraft because they are not constraint to land on runways. We address a slightly more complicated problem since we require to find not only feasible landing sites with approach paths but also landing sites that are reachable from a ground goal in Chapter 5.

The final chapter presents a generalized approach to planning that allows larger rotorcraft to perform missions such as searching for landing sites while at the same time avoiding obstacles (Chapter 6). The approach is more computationally intensive as the reactive algorithm presented earlier and relies on being able to accurately predict the dynamics. However, it is more suitable for problems where we would like to plan trajectories that can be guaranteed safe and deliberate. Additionally, the method can be quickly adapted to new dynamic models, sensors, and missions.

The concluding Chapter 7 contains a summary, conclusions, and perspectives for future work.

1.4 Publication Notes

Chapter 3 is edited from the previously published paper in [Scherer et al., 2008]. The efficient cost function algorithm of Chapter 4 was previously published in [Scherer et al., 2009] and has been updated with an omitted line in the original LIDT algorithm and results from using the algorithm on a helicopter. Parts of chapter 5 have been published in [Scherer et al., 2010]. The landing site algorithm has been enhanced to incorporate approach paths, and ground search paths. The results now also reflect results from new landing experiments on the Unmanned Little Bird helicopter and shows computation results for a more optimized algorithm.

2 Related Work

Guidance for unmanned aerial vehicles is challenging because it is necessary to solve many sub-problems to achieve an interesting level of autonomy. Initially, much research has focussed on keeping vehicles airborne. After good controllers had been demonstrated and were available, the boundaries of operation of aerial vehicles can be pushed lower and closer to obstacles. However, as one gets close to obstacles, GPS-based position estimation is not reliable enough and new methods have to be developed. Moreover, given that positioning information is available, it is still necessary to sense the surroundings to avoid collisions. Once that reactive capability is achieved motion planning can be incorporated.

In the following section we will present related work on control, obstacle avoidance, perception, and motion planning which is applicable to aerial vehicle systems.

2.1 Control

Computing hardware made it possible to control aerial vehicles autonomously, and vision enabled to control helicopters in the 90's. Using the limited computational resources available impressive vision-based control was achieved by Amidi et al. [Amidi, 1996, Amidi et al., 1998], Kanade et al. [Kanade et al., 2004], Lewis et al.[Lewis et al., 1993], and others. Many of these capabilities were partially motivated by the International Aerial Robotics Competition [Michelson, 2000]. Initially, while GPS accuracy was not sufficient for control, vision based approaches for control were developed by Proctor et al.[Proctor et al., 2006, Proctor and Johnson, 2004] and later while control using GPS to determine position became possible, focus shifted more towards visual servoing for applications on platforms as described by Saripalli et. al [Saripalli and Sukhatme, 2003]. As computers and batteries became lighter, control of smaller vehicles indoors as developed by Bouabdallah [Bouabdallah, 2007], and Tournier [Tournier, 2006] was possible. For small ornithopters control using vision enables stable flight. One example is the Delfly described by Deal and Huang[Deal and Huang, 2008].

In our work, we will assume that the low-level control problem has been solved and that we have a higher level waypoint or over-ground velocity input to command the vehicle.

2.2 Obstacle Avoidance

Obstacle avoidance is a necessary capability to operate close to structures. Optic flow is a popular approach to avoid obstacles since it is biomimetic and lightweight. Vision-based reactive methods have been popular because payload weight is a serious limitation for UAVs. Beyeler et al. [Beyeler et al., 2006, Beyeler et al., 2007] and Zufferey et al. [Zufferey et al., 2006] developed autonomous extremely lightweight indoor fliers that avoid obstacles and stay away from obstacles using a small 1-D camera. Oh et al. [Oh et al., 2004, Oh, 2004] and Green et al.[Green and Oh, 2008] developed autonomous optical flow based obstacle avoidance similar in idea to the design by Zufferey et al. A larger fixed-wing aerial vehicle was used by Grifiths et al. [Griffiths et al., 2006] and Merrel et al. [Merrell et al., 2004] to reactively avoid obstacles and follow a canyon with ladar and optical flow.

Hrabar et al. [Hrabar and Gaurav, 2009] also performed extensive experiments using optical flow for obstacle avoidance however additionally a stereo camera was used to prevent collisions from straight ahead zero-flow regions. Stereo image processing with evidence grid based filtering was also used by Andert et al.[Andert et al., 2010][Andert and Goormann, 2007] to create a map based on stereo imagery that avoided obstacles reactively in simulation. Viquerat et al. [Viquerat et al., 2007] presented an reactive approach to avoid obstacles using Doppler radar. Paths around obstacles

have been planned and maps have been built using ladar by Whalley et al. [Whalley et al., 2008, Whalley et al., 2005], and Tsenkov et al. [Tsenkov et al., 2008]. Recently Grzonka et al. presented a small quad-rotor that is capable of localization and SLAM in 2D[Grzonka et al., 2009].

Byrne et al. have demonstrated obstacle detection from a wide-baseline stereo system that uses color segmentation to group parts of the scene (even if lacking in optical texture) into spatially contiguous regions for which it is possible to assign range [Byrne et al., 2006].

Shim and Sastry have proposed an obstacle avoidance scheme that uses nonlinear model predictive control. Their system builds controllers for arbitrarily generated trajectories. They have demonstrated results on a RMax platform operating in a plane using a single-axis laser rangefinder at speeds of 2m/s [Shim et al., 2006].

Zapata and Lepinay have proposed a reactive algorithm[Zapata and Lepinay, 1999] similar to the reactive algorithm presented in Chapter 3.2.2, but we are aware only of simulated results [Zapata and Lepinay, 1998]. We contribute an integrated architecture in the next Chapter that combines path planning with a reactive obstacle avoidance algorithm based on a global map with a virtual sensor and show significant results from real experiments.

2.3 Obstacle Cost Calculation

Typically obstacle costs in a grid based representation for planning are calculated by searching for the closest obstacle within the desired radius. However aerial vehicles want to stay far away from obstacles if necessary and therefore want a large obstacle expansion. The obstacle expansion is related to the distance transform and Meijster et al.[Meijster et al., 2000] presented an efficient algorithm to globally calculate the distance transform. Kalra et al. [Kalra et al., 2006] showed an incremental algorithm to incrementally construct Vornonoi diagrams. We show an efficient algorithm similar to D* Lite [Koenig and Likhachev, 2002a] that updates the distance transform up to a limit incrementally.

2.4 Landing Site Evaluation

There has been some prior work on landing and landing site selection. From a control perspective the problem has been studied by Sprinkle[Sprinkle et al., 2005] to determine if a trajectory is still feasible to land. Saripalli et al. [Saripalli and Sukhatme, 2007] have landed on a moving target that was known to be a good landing site. Barber et al. [Barber et al., 2005] used optical flow based control to control a fixed wing vehicle to land.

Vision has been a more popular sensor modality for landing site evaluation because the sensor is lightweight. De Wagter and Mulder[de Wagter and Mulder, 2005] describe a system that uses vision for control, terrain reconstruction, and tracking. A camera is used to estimate the height above ground for landing in Yu et al. [Yu et al., 2007], and similarly in Meijas et al. [Mejias et al., 2006] a single camera is used to detect and avoid powerlines during landing.

Using a stereo camera, Hintze [Hintze, 2004] developed an algorithm to land in unknown environments. Bosch et al. propose an algorithm for monocular images that is based on detecting nonplanar regions to distinguish landing sites from non-landing sites[Bosch et al., 2006]. Another popular approach is to use structure from motion to reconstruct the motion of the camera and to track sparse points that allow reconstruction of the environment and plane fitting. This method as been applied to spacecraft by Johnson et al. [Johnson et al., 2005] and to rotor craft by Templeton et al. [Templeton et al., 2007].

Ladar based landing site evaluation has not been studied very deeply except as suggested in Serrano et al. [Serrano, 2006] where a framework for multiple sources like ladar, radar, and cameras is proposed to infer the suitability of a landing site. For ground vehicles ladar based classification is popular because of its reliability and direct range measurement. It has been used in conjunction with aerial vehicle data for ground vehicles by Sofman et al. [Sofman et al., 2006] and by Hebert & Vandapel [Hebert and Vandapel, 2003].

Our work in landing site evaluation uses ladar data and some of the proposed techniques in related work such as plane fitting and roughness for coarse evaluation. However, our method goes beyond pure plane fitting to actually fit a model of the helicopter geometry to a triangulation of the environment and also incorporates further constraints based on approach, abort and ground paths.

2.5 Motion Planning

Current approaches to unmanned aerial vehicle motion planning either address the problem of obstacle avoidance while reaching a goal or optimize a specific objective such as search & track. In Chapter 3 we address reaching a goal point while avoiding obstacles on an RMax helicopter and extend this work to a more general applicability in Chapter 6.

Search & track methods for UAVs optimize a similar objective as the landing site search, however current approaches ignore obstacles and are optimal for a certain task such as the methods by Frew & Elston[Frew and Elston, 2008] and Tisdale et al. [Tisdale et al., 2009]. Tompkins et al. combined sun exposure and energy to plan a mission plan that reaches a goal point[Tompkins et al., 2002]. Cyrill & Grisetti propose a method to explore an environment while also enabling improved localization and mapping [Stachniss and Grisetti, 2005]. While these approaches optimize one particular problem they do not generalize to several missions.

One problem that is particularly important for military applications is search and target tracking. For this problem optimal specific solutions have been developed by Grocholsky[Grocholsky, 2002] using an information theoretic approach. A tracking algorithm was developed by Brooker et al.[Brooker et al., 2003] to optimize a target estimate on the ground. Usually one optimizes a trajectory up to a certain horizon and is myopic after that, as there is no benefit in planning further for search and track because areas of good information are not predictable. In the problems we are addressing there is a benefit in planning with a longer planning horizon since information is not changing as quickly.

Real-time motion planning for robotics is an important problem that addresses the problem of which trajectory to take based on partial information. Stentz [Stentz, 1994], Koenig & Likhachev [Koenig and Likhachev, 2002b], and Ferguson & Stentz [Ferguson and Stentz, 2006] developed efficient incremental algorithms to replan kinematic routes once new sensor information is received. These algorithms usually assume a grid based representation of the world. Nevertheless, other representations are also possible such as a 3D path network presented by Vandapel et al. [Vandapel et al., 2005] or octree based presentations as demonstrated by Kitamura et al. [Kitamura et al., 1996]. An alternative to graph-based planning methods is the Laplacian path planning algorithm by Jackson et al.[Jackson et al., 2005] that is similar to the approach used in Connolly et al.[Connolly et al., 1990] and Li & Bui[Li and Bui, 1998].

Since routes planned only based on geometric constraints are not necessarily feasible to execute if close trajectory tracking is necessary, planning approaches that search a graph over feasible motions have also been developed by Frazzoli et al. [Frazzoli et al., 2005, Frazzoli et al., 2002] using maneuver automata, Hwangbo et al. [Hwangbo et al., 2007] using feasible grid resolutions, Shim et al. [Shim et al., 2006] using model predictive control, and Pivtoraiko et al.[Pivtoraiko et al., 2009] using a state lattice. These methods could be used in our generalized planning framework to create an initial guess for trajectory optimization.

Another direction to improve planning algorithms has been to plan to incorporate other objectives to improve future sensing as in Michel et al. [Michel, 2008], or to direct a sensor by hallucinating obstacles as in Nabbe [Nabbe, 2005], and Nabbe et al.[Nabbe and Hebert, 2007]. Other researchers have addressed the problem of staying localized while navigating without GPS. He et al. [He et al., 2008] planned for a small quadrocopter and Gonzalez & Stentz planned to stay localized for a ground vehicle [Gonzalez and Stentz, 2008]. Even though in the above approaches a goal point is assumed, objective functions are incorporated in the planning phase to permit low-cost behavior.

Trajectory optimization is used as a step after planning and sometimes sufficient on its own. The goal of trajectory optimization is to improve an initial guess of a trajectory to optimize an objective function. It has been incorporated into a planning framework by Brock et al. [Brock and Khatib, 2002],

Quinlan et al.[Quinlan and Khatib, 1993], and Kavraki et al. [Kavraki et al., 1996]. Trajectory optimization methods that have been used to avoid obstacles while trying to preserve smooth trajectories are presented in Schlemmer et al. [Schlemmer et al., 1995] for a robot arm. Richards & How[Richards, 2002] and Kuwata [Kuwata, 2007]used mixed-integer linear programming to optimize trajectories as a receding horizon controller for air and spacecraft to avoid obstacles. Recently Kolter & Ng[Kolter and Ng, 2009] presented a trajectory optimization method based on cubic spline optimization. Howard & Kelly [Howard and Kelly, 2007] presented an efficient trajectory optimization algorithm that is able to incorporate additional constraints such as smoothness.

One method that inspired our trajectory optimization algorithm is the CHOMP algorithm by Ratliff et al. [Ratliff et al., 2009] that distributes updates to command waypoints to be smooth. Trajectory optimization methods are interesting because they permit a quick reaction to changes in the objective function. To run the optimization procedure we need an initial guess that will be optimized. The difficulty for complex objective functions is to find a good initial guess as many potential goal points exist. Our approach addresses this problem by using a grid planning method to calculate a set of initial guesses that is then optimized.

Some solutions to the problem of operating aerial vehicles at low-altitude have been proposed in previous work. However, a comprehensive approach that is able to plan motion with multiple changing and unknown objectives in real-time is missing and will be proposed in Chapter 6.

3 Low-Altitude Obstacle Avoidance

3.1 Problem

This chapter presents a fast reactive algorithm that allows autonomous rotorcraft to fly close to the ground and reach a goal point while avoiding obstacles. Operationally, we would like a system that can safely fly between coarsely specified waypoints without having to know beforehand that the flight path is clear or that a waypoint is even achievable. Flying close to and among obstacles is difficult because of the challenges in sensing small obstacles, and in controlling a complex system to avoid obstacles in three dimensions. Some aspects of collision avoidance are easier for air vehicles than ground vehicles. Any object close to the intended path of an air vehicle must be avoided as opposed to ground vehicles where deviations from the nominal ground plane indicate obstacles and are often not visible until they are close. The use of small helicopters, rather than fixed wing aircraft also helps because in the worst case it is possible to come to a hover in front of an obstacle. Still, the availability of appropriate sensors, logistical issues of mounting a vehicle with sufficient sensing, computing and communication gear, and the risk involved in such experimentation has kept researchers from significant progress in this area. While some methods of obstacle avoidance on aircraft have been implemented, none to our knowledge has achieved the speed and endurance that we present here.

In order to operate in real time, we have developed a layered approach, similar in a general way, to the approach used by autonomous ground vehicles that operate in uncharted terrain: *plan globally and react locally.* Our approach combines a slower path planner that continuously replans the path to the goal based on the perceived environment with a faster collision avoidance algorithm that ensure that the vehicle stays safe.

This approach accrues two main benefits. First, it decomposes the problem of navigating in unknown terrain into two simpler separable problems– one of determining a generally preferred route and the other of staying safe. Route planning and replanning is best done online given that highly accurate maps are not available ahead of time, but such a computation is not required at the high frequency necessary for avoiding obstacles. While route planning is best off considering all the information available and is hence slower, collision avoidance requires consideration of only a shorter horizon and thus can be executed at a higher frequency. Second, any map information available ahead of time can be easily used to bootstrap the representation used by the planning algorithm to generate plans that are of better quality than those that must be generated based on sensed data.

The coarse planner creates a path to the goal point from the current location of the vehicle. The path is used to provide sub-goals to the reactive algorithm. The reactive algorithm is based on a model of obstacle avoidance by humans. This method is similar to the classic potential fields [Khatib, 1986] that compose a control by adding repulsions from obstacles and an attraction to the goal but it prescribes a potential over the steering function rather than only the euclidean position of the vehicle. This difference is important for two reasons– first it helps the vehicle get close to obstacles (steering the vehicle away only if it is headed towards an obstacle) if necessary and it incorporates the vehicle's steering dynamics into the control. As might be expected, the collision avoidance method must be tuned to deal with the specific mechanism used. Here we show how the parameters necessary for collision avoidance can be learned automatically by analysis of only one path generated by a pilot remote controlling the vehicle to avoid obstacles. The method by Fajen & Warren[Fajen and Warren, 2003] that inspired our work is a control law to steer at constant speed in 2D to avoid point obstacles given a fixed goal. We have made novel extensions to this method to deal with: non-point obstacles, irrelevant obstacles, three dimensional environments, distant goals, and varying speeds.

The combined local and global methods produce an efficient navigation controller that can both look ahead significantly as well as be nimble even when the vehicle is flying at high speeds. The main

Figure 3.1: The helicopter platform we used to test collision avoidance flying autonomously between two poles 10 m apart. The rotor span of this helicopter is 3.4 m

contribution that this paper documents is a methodology to autonomously navigate in 3D, that is to travel from A to B without prior knowledge of the environment. We have validated these ideas by a large number of experiments (over 1000 flights on an autonomous helicopter shown in Fig 3.1) in two significantly different environments at speeds (up to 20 knots) significantly greater than those attempted by anyone to date. .

Next we describe sensor processing in section 3.2.1. Section 3.2.2 explains our architecture and the algorithm is shown in section 3.2.3. The speed control is described in section 3.2.4 and the mission execution in section 3.2.5. In section 3.3.1 we describe the flight hardware and range sensor. Section 3.3.2 describes our simulator and in section 3.3.3 we show results from field tests and conclude in section 3.4.

3.2 Approach

3.2.1 Range Sensor Processing

We keep a map of the world because it improves decision making and provides more information about obstacles than an instantaneous range sensor scan. The map we use is a three dimensional *evidence grid* because it is able to express the belief of a volume of space being occupied[Martin and Moravec, 1996]. Another advantage of an evidence grid is that it is a compact view of the world that can be updated in realtime. Furthermore the grid can incorporate sensor uncertainty and represents relevant obstacle occupancy. It is arranged as a regular grid of voxels that store the log-likelihood ratio of the voxel being non-empty. The raw range sensor data is transformed to an occupancy probability and the result is mapped into an evidence grid using position and orientation information. Fig. 3.2 summarizes the flow of data.

3.2.1.1 Construction of Evidence Grids

The evidence grid algorithm assumes a static scene. It will however incorporate changes in the environment if it sees enough evidence of empty space or occupancy. Accordingly our algorithms will

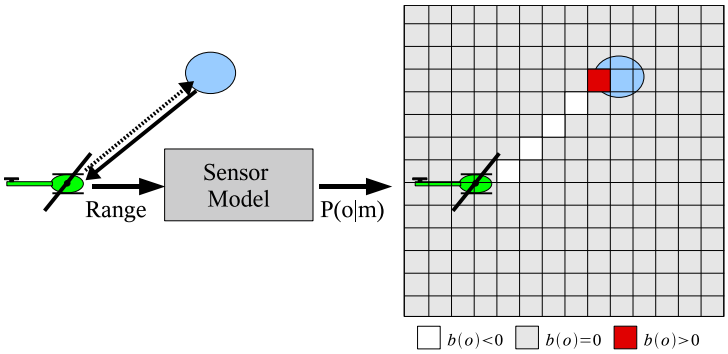


Figure 3.2: Evidence grid processing example. A laser rangefinder returns a signal indicating a hit. The signal is mapped to an occupancy probability $P(m|o)$. This probability is used to update the relevant belief cells in the evidence grid.

avoid moving obstacles suboptimally since moving obstacles will create a smeared representation in the evidence grid and no explicit tracking is performed.

The notation used in this derivation follows closely the notation used by Foessel[Foessel, 2002]. Let the grid be denoted by M where each grid cell is $m_{<x,y,z,>}$. The value of a cell is $b(m)$, the belief of occupancy of the cell. Initially it is assumed that the probability of a cell being occupied initially (prior probability) is $P(m) = 0.5$ and that a cell is either occupied or empty so $b(m) = 1 - b(\overline{m})$. $b'(m)$ is the updated belief of the evidence grid.

The sensor model maps the signal returned from the laser rangefinder to a probability of occupancy given the observation $P(m|o)$. The belief of the cells $b(m)$ of the evidence grid is then updated using Bayes rule:

$$b'(m) = \frac{P(m|o)}{1 - P(m|o)} \cdot \frac{1 - P(m)}{P(m)} \cdot \frac{b(m)}{1 - b(m)} \tag{3.1}$$

however since $P(m) = 0.5$:

$$b'(m) = \frac{P(m|o)}{1 - P(m|o)} \cdot \frac{b(m)}{1 - b(m)} \tag{3.2}$$

A representation of the belief which is better suited for our purposes and faster to update is the log odds representation. The log odds of the update rule is

$$\overline{b}(m) = \ln \frac{b(m)}{1 - b(m)} = \ln b(m) - \ln(1 - b(m)) \tag{3.3}$$

Therefore the new update rule is

$$\overline{b}'(m) = \overline{b}(m) + \ln P(m|o) - \ln(1 - P(m|o)) \tag{3.4}$$

Our robot uses a laser rangefinder that scans in azimuth and elevation (Also see section 3.3.1). The sensor returns a range and status value. Since the accuracy is independent of range we have a range-invariant sensor model. Furthermore a reported hit is extremely likely to be from an obstacle. Accordingly we map the probability to a simple model in which

$$\ln P(m|o) - \ln(1 - P(m|o)) = 127 \tag{3.5}$$

and

$$\ln P(m|\overline{o}) - \ln(1 - P(m|\overline{o})) = -1 \tag{3.6}$$

if we received a valid return. We determined these values based on experiments with the sensor and chose the values with the qualitatively best evidence grid.

The algorithm used for selecting the cells affected in the 3D evidence grid is a 3D extension of Bresenhams line algorithm[Bresenham, 1965]. A description of the algorithm are can be found in

Location	% Occupied	% Empty	% Unknown	# Occupied	# Empty	# Unknown
Ft. Benning, GA	2.10	14.20	83.70	88344	595399	3510561
Phoenix, AZ #1	1.80	6.52	91.68	75727	273342	3845235
Phoenix, AZ #2	0.26	1.40	98.34	10922	58812	4124570

Table 3.1: Percentage and number of occupied, empty and unknown cells after navigating in three different environments. A cluttered test site environment with dense vegetation and two environments with sparse obstacles and vegetation.

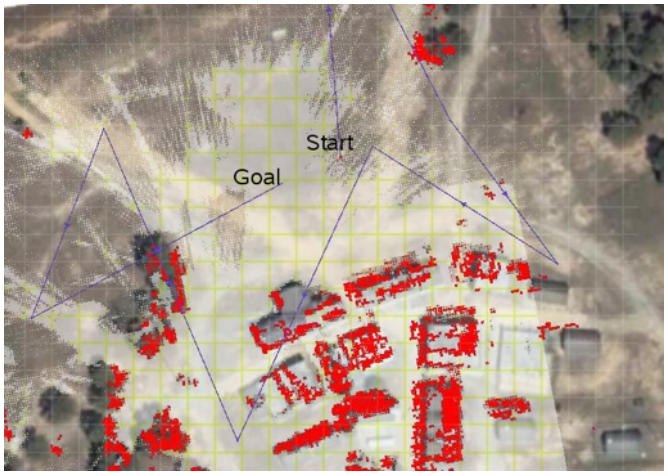


Figure 3.3: A 2D horizontal slice of an evidence grid from the McKenna military operations in urban terrain (MOUT) site at Ft. Benning, GA (USA). This model was created by flying part of the shown path. Red(Dark) represents occupancy and white is evidence of free space. An aerial image is overlaid on top of the evidence grid in order to facilitate the interpretation and show the accuracy of the constructed model. The grid spacing is 10 m.

[Liu and Cheng, 2002].

The beam of the ladar has a diversion at the maximum range that is smaller than the evidence grid cell size. Therefore processing of a new ladar hit is linear $O(n)$ in the number of cells of the ray from the current location to the hit location.

3.2.1.2 Evaluation of evidence grids

Evidence grids have advantages and disadvantages over other representations of obstacles like lists of hits representing point clouds. In the following we explore some of the issues of the use of evidence grids.

An advantage of using evidence grids is the representation of uncertainty in the occupancy for each part of the space defined by a cell. A false positive data point is erased if enough rays penetrate the cell containing a false positive hit. Dust particles and rain drops, for example can cause a false obstacle to appear. Although the robot might react to such false positives the wrong evidence added will be erased if enough evidence of empty space is accumulated. However if a real obstacle is only visible in a very small fraction of hits it is possible that the hits will get erased. The smallest obstacle in our problem specification was a 6mm thin wire that was seen by the sensor sufficiently often that this was not a problem.

Processing of new sensor information is fast in evidence grids because only a small number of cells have to be updated for a laser rangefinder and the calculation of the cells that need to be updated can be performed fast. However, the coarse discretization of the grid leads to aliasing. Aliasing misregisters obstacles and can cause true obstacles to be erased.

The grid is regular and of fixed size. Consequently, it requires a constant amount of memory

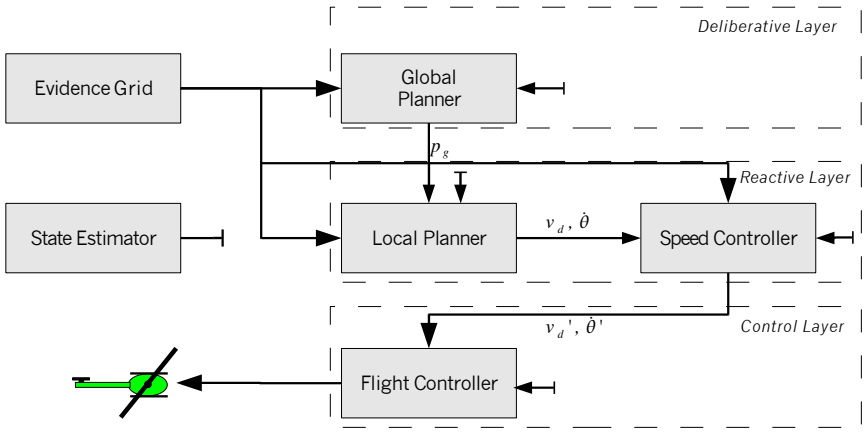


Figure 3.4: Overall software architecture of the algorithms. The higher the layer the lower the frequency of execution. A path p_g is produced in the planning layer and transmitted to the reactive layer. The reactive layer produces commands $(v_d', \dot{\theta}')$ that are then executed by the flight controller.

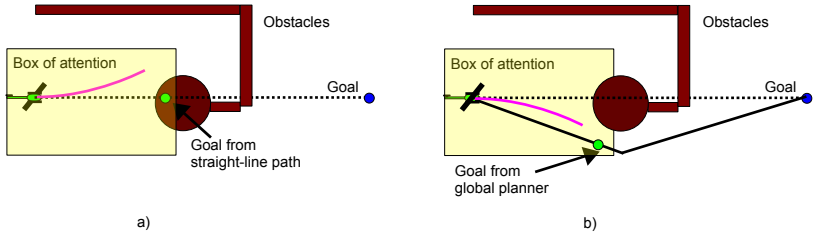


Figure 3.5: The benefit of combining local and global methods for navigation. In (a) the vehicle uses obstacles in the "box of attention" in its guidance. Since the horizon is short, this can lead the vehicle to a cul-de-sac from where it might have to either turn around or execute an expensive climb maneuver. In (b), the vehicle selects its temporary goal point along the path found by the global planner and ends up following a better route.

to store. This design reduces the computation required to update a memory location and does not require managing dynamic allocation and deallocation of cells. However in the case of sparse obstacles this can result in large amounts of wasted memory. As shown in Table 3.1 most cells are never touched. Only a small percentage(0.26% - 2.1%) of cells is occupied in Phoeniz, AZ and Ft. Benning, GA. The evidence of being empty is however useful to reject false positives.

A 2D slice of an evidence grid from McKenna MOUT is shown in Fig. 3.3. This model was created using the displayed flight path and is overlaid with an aerial view of the site. Since the evidence grid was shifted several times there are hard boundaries when the position of the grid changed. A large amount of space is also known as free space with a high certainty. This can help in deciding if a region is safe to traverse.

3.2.2 Architecture

Our layered obstacle avoidance architecture uses a combination of a global planner with a local planner to react quickly to obstacles while not getting trapped in local minima. This appears like double the work at first since we are avoiding obstacles in both algorithms. However the goal of the global planner is to find a path around obstacles. This path is not necessarily executable in the short run since the vehicle could be moving in the opposite direction. However in the long run

the vehicle should move roughly along the path of the global planner. The local planner's priorities on the other hand are to avoid obstacles and then try to follow the path from the global planner. Another advantage in using two algorithms is that reaction is faster and reliability requirements decrease with the complexity of the problem the algorithm tries to solve. Searching a path in three dimensions is difficult and finding a path might even be impossible. However our overall architecture will still avoid obstacles. If even our local planner should fail the helicopter will at worst come to a stop in front of an obstacle because the speed controller will stop early enough.

In Fig. 3.4 one can see that at the lowest layer the speed controller slows and accelerates the vehicle based on the distance to the closest reachable obstacle and on the minimum turning radius. At the next level, our local planning algorithm produces steering commands in both horizontal and vertical axes to actively avoid obstacles. At the highest layer, a global path planning algorithm generates a smooth path around obstacles to the next goal.

Since a helicopter can come to a complete hover, it can get around obstacles by moving horizontally or vertically. However, we prefer to smoothly change the direction of travel like a fixed wing aircraft, partly for energy reasons but also because such motion keeps the sensor looking forward. Hence, in the ideal case, the helicopter maintains a constant speed while it maneuvers around obstacles.

This behavior is achieved by integrating the reactive layer (Local Planner and Speed Control) with the deliberative layer (Global Planner). Since the global planner looks far ahead to the next waypoint (100s of meters) it will produce a nominal path that will not require the reactive layer to escape by flying straight up or by turning around. In the unlikely case that a large obstacle appears suddenly, it will still be avoided by the reactive layer long before it is incorporated by the planning layer. Furthermore it is is necessary to incorporate a deliberative layer because the reactive layer alone can get stuck in certain cluttered configurations of obstacles as illustrated in Fig. 3.5. In this example the helicopter might have to climb over the obstacle if it would just use the reactive layer because it might decide to turn left. Instead by integrating the local and global planner we pick a new goal point based on the planned trajectory and can therefore successfully avoid the obstacles.

3.2.3 Reactive Collision Avoidance Algorithm: 3D Dodger

In this section we present our formulation of a 3D local planning algorithm based on a control law for obstacle avoidance in people avoiding point obstacles studied by Fajen and Warren[Fajen and Warren, 2003]. The interesting aspect of this model is that the human (or robot in our case) is avoiding obstacles and reaching a goal by turning in the direction where the obstacle repulsion and goal attraction are at an equilibrium.

This is in contrast to previous approaches like potential fields [Khatib, 1986], because the repulsion and attraction does not only depend on the position of the robot but depends on the state ($[x, y, \theta]$ in 2D for example). The advantage of this model is that the direction of motion is incorporated in the reaction which will cause the avoidance maneuver to be smoother because the turning dynamics influence the reaction.

2D Formulation Fajen and Warren's model (FW) uses a single goal point which attracts the vehicle's heading. This model uses angles and distances to obstacles. The terms used in this formulation are shown in Fig. 3.6. The attraction increases as the distance to the goal decreases and as the angle to the goal increases (Fig. 3.7.A-B), yielding the goal attraction function

$$attract_{FW}(g) = k_g \psi_g (e^{-c_1 d_g} + c_2) \tag{3.7}$$

ψ_g is the angle to the goal. d_g is the distance to the goal. k_g, c_1, and c_2 are parameters which must be tuned. Similarly, each obstacle repulses the agent's heading. The repulsion increases with decreasing angle and decreasing distance (Fig. 3.7.C-D). Then for each obstacle, there is a repulsion function:

$$repulse_{FW}(o) = k_o \psi_o (e^{-c_3 d_o})(e^{-c_4 |\psi_o|}) \tag{3.8}$$

ψ_o is the angle to the obstacle. d_o is the distance to the obstacle. k_o, c_3, and c_4 are parameters which must be tuned. These attractions and repulsions are summed together (assuming a superposition

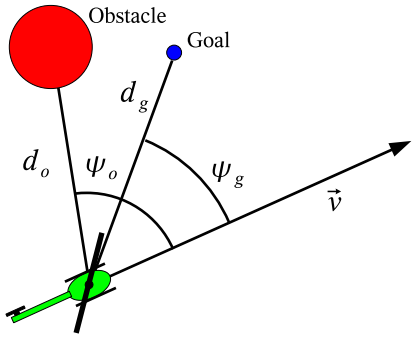


Figure 3.6: A diagram illustrating the terms used in the reactive obstacle avoidance control law. The coordinates used are expressed in a robot centric reference frame.

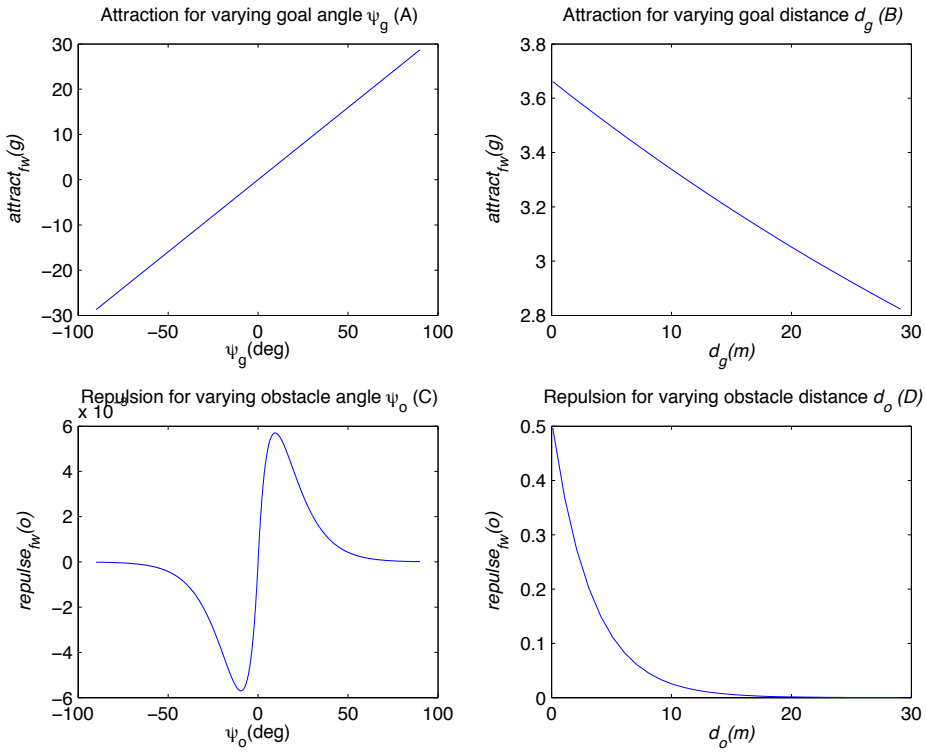


Figure 3.7: The attraction and repulsion for the original 2D control law for varying goal angle (A), varying goal distance (B), varying obstacle angle (C), and varying obstacle distance (D)

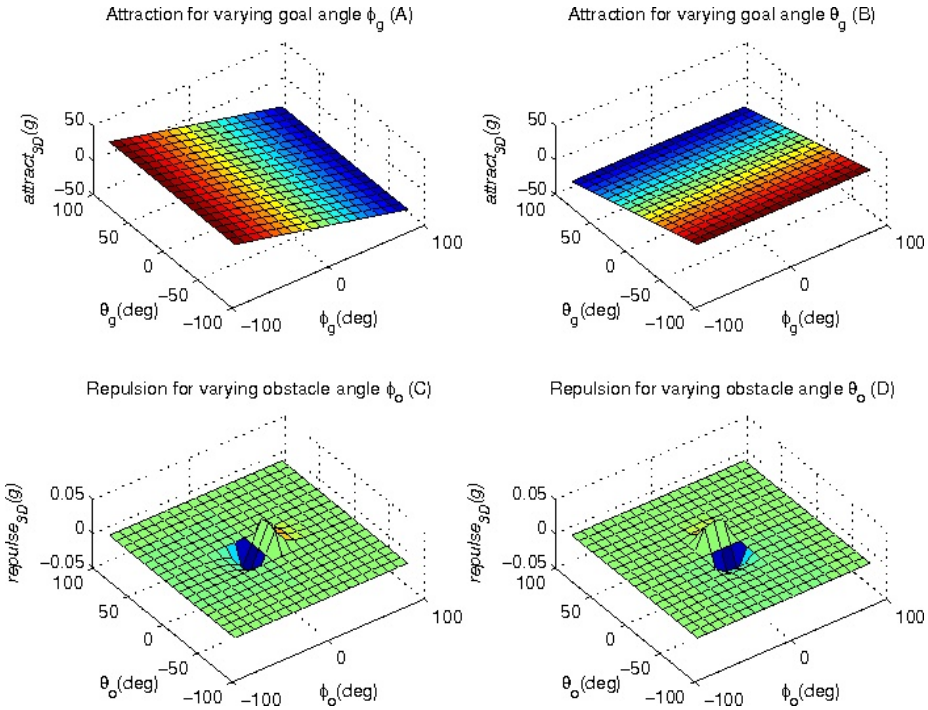


Figure 3.8: The attraction and repulsion surface for our modified control law. Plot (A) and (B) show the independence of the goal attraction on each axis. Since a sigmoid shapes the repulsion for the repulsion function in plot (C) and (D) is not uniform.

holds) and damped with the current angular velocity to get an angular acceleration command. The result is a single control model:

$$\ddot{\phi}_{FW} = -b\dot{\phi} - attract_{FW}(g) + \sum_{o \in O} repulse_{FW}(o) \tag{3.9}$$

Extension to 3D Adding another dimension to the 2D algorithm requires to add the possibility of moving vertically. In our algorithm we achieve a vertical movement by pretending we can steer also in the up-down direction. However adding this dimension also causes an ambiguity because one can choose from an infinite number of directions along the obstacle boundary, while in 2D there are only two choices, e.g. left and right.

In other words obstacles are double counted if they are avoided along both axis simultaneously. If an obstacle is to our right, for example we would still try to climb or sink to avoid the obstacle. We resolve this ambiguity by decreasing the repulsion of obstacles as the angle on the other axis increases. The repulsion of the other axis is decreased using a sigmoid function.

The goal attraction on the other hand is just a concatenation of the two axis because there is no choice in steering to the goal. Steering left-right and up-down can be expressed as two angles in two planes of the helicopter. A natural representation in 3D dimensions is therefore a spherical coordinate system $[r, \theta, \phi]$ for goal and obstacle points. The resulting command for the helicopter in left-right is $\dot{\theta}$ and in up-down is $\dot{\phi}$.

The goal attraction has two angles θ_g, ϕ_g and a distance to the goal d_g. The attraction to the goal increases proportionally with angle and decreases exponentially with distance (Fig. 3.8A-B). The vector of the two steering rates for goal attraction is therefore defined as

$$\overrightarrow{attract}_{3D}(g) = \overrightarrow{k_g} \begin{bmatrix} \theta_g \\ \phi_g \end{bmatrix} (e^{-c_1 d_g} + c_2) \tag{3.10}$$

Obstacle positions are also expressed in spherical coordinates. The repulsion increases exponentially with decreasing angles and decreases exponentially with increasing distance as shown in Fig. 3.8C-D. Also, larger angles from the other axis like ϕ for θ decrease the repulsion from obstacles. The repulsion function is

$$\overrightarrow{repulse}(o) = -\vec{k_o}.$$

$$\begin{bmatrix} sign(\theta_o) \cdot sigmoid(s_1(1 - \frac{|\phi_o|}{s_2})) \\ sign(\phi_o) \cdot sigmoid(t_1(1 - \frac{|\theta_o|}{t_2})) \end{bmatrix} (e^{-c_3 d_o}) \begin{bmatrix} e^{-c_4|\theta_o|} \\ e^{-c_4|\phi_o|} \end{bmatrix} \tag{3.11}$$

where

$$sign(x) = \begin{cases} 1 \ if \ x > 0 \\ 0 \ if \ x = 0 \\ -1 \ if \ x < 0 \end{cases} \tag{3.12}$$

and

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.13}$$

The resulting steering rate command sent is

$$\dot{\vec{\phi}}_{3D} = \overrightarrow{attract}_{3D}(g) + \sum_{o \in O} \overrightarrow{repulse}_{3D}(o) \tag{3.14}$$

because we assume a superposition principle holds.

Since the off-axis angles have less weight than the angles closer to the direction of travel the algorithm commits to going around or over obstacles after it has reacted sufficiently to an obstacle. For example, imagine the vehicle approaching a telephone pole head on. Initially both the horizontal and vertical controllers will respond to the pole, turning up and say, to the left. But as the pole moves more to the right, it falls out of the attention region of the vertical controller defined by the sigmoid function and remains in the attention region of the horizontal controller. The result is that the vehicle stops its vertical avoidance maneuver and commits to the horizontal avoidance. Note, however that this commitment is only due to the fact that the obstacle angles change and if an obstacle should suddenly appear the behavior changes. In another scenario, say the vehicle approaches a wide building. Again, both controllers initially react, but this time the building moves out of the attention region of the horizontal controller first. The result is that the vehicle commits to the vertical maneuver and climbs over the building. Allowing both controllers to initially react and *compete* for control results in intrinsically choosing the reaction that most quickly avoids the obstacle, while keeping both options open initially.

The two steering rates $[\dot{\theta}, \dot{\phi}]$ that are the output of our control law need to be converted into a representation suitable for a helicopter that typically has four control inputs $\{v_{xd}, v_{yd}, v_{zd}, \dot{\theta}_d\}$. We reduce the 4 to 2 degrees of freedom in the following way. First, we impose an artificial non-holonomic constraint for lateral velocities $v_{yd} = 0$, chiefly because we want the laser scanner to point in the direction of travel. Second, the magnitude of the velocity vector is determined by the speed controller. Therefore we are left with only 2 degrees of freedom: A heading rate $\dot{\theta}$ and a vertical velocity component v_{zd}. Since the input to the UAV is a vertical velocity we create a velocity command based on the vertical heading rate. The current velocity vector is rotated to reflect the tangent to the circle defined by $\dot{\phi}$ in Δt seconds where we use a fixed $\Delta t = 0.2s$.

Our local planning algorithm avoids obstacles in all the situations we tested; however, the behavior is not as desired in several situations. One undesirable feature of our method is the independence of the reaction from speed. Ideally the reaction of the method should depend on the time to collision since one wants to react earlier to obstacles if the speed is fast and later if the speed is slow. We tuned the behavior of our system at 4 m/s and the reaction was sufficient up to 10 m/s. However at 10 m/s the reaction was not as smooth as at 4 m/s.

Another related problem that we solved using a box constraint (described later) is that the ground

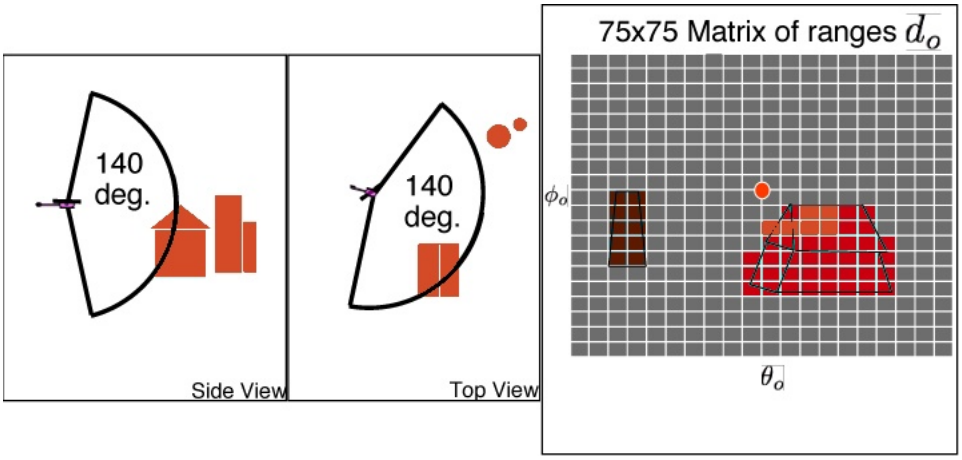


Figure 3.9: A diagram showing how an obstacle is projected into a grid-discretized spherical representation. The angle θ_o and ϕ_o determine the index into the grid. An obstacle at a similar angle is ignored because it is occluded by the closer obstacle.

plane is an obstacle just like any other building or wire. However since our goal is to fly low we need to be able to ignore the ground up to a certain altitude. Therefore we remove visible obstacles below a certain altitude.

Virtual range sensor In the original formulation only point obstacles are avoided and we observed in our experiments that even with non-point obstacles the model is able to produce the right behavior. However since we approximate the obstacles by a set of points that are summed up (superimposed), we would like to consider a minimal and relevant set of obstacles to avoid. One insight is that obstacles that are not visible from the current location of the robot are not avoided because it is not possible to hit these obstacles. Therefore it is sufficient to consider obstacles that are in line-of-sight. Furthermore, since the robot only flies in one direction and obstacles behind the current direction of travel don't matter in terms of obstacle avoidance they can be ignored. We use of a wide field-of-view(FOV) virtual range sensor as shown in Fig. 3.9 to represent obstacles.

The set of obstacles $o_i \in O$ where

$$o_i = \begin{bmatrix} \theta_i \\ \phi_i \\ d_i \end{bmatrix} \tag{3.15}$$

determines the behavior in the presence of obstacles. The virtual range sensor returns a matrix of ranges for a grid-discretized latitude and longitude pattern from a reprojected z-buffer. This projection of obstacles from Cartesian coordinates in the evidence grid to vehicle centric spherical coordinates is created by rendering the evidence grid.

We use virtual range sensor data because we want to be robust against noisy measurements and want to keep a memory of the environment. The sensor we are using has a relatively small field of view and without memory it is possible that the control will oscillate between obstacles going in and out of field of view.

The advantage of this representation is that it considers the relative size of an obstacle to be more important than the absolute size. Large obstacles at a large distance have less influence than small obstacles close to the robot. Furthermore the number of obstacles is also limited by the discretization because each entry in the matrix represents only one obstacle point o. The matrix of ranges to obstacles has a field of view of 140 degrees in both axis and each matrix entry represents the closest distance in the volume of a 2x2 degree pyramid.

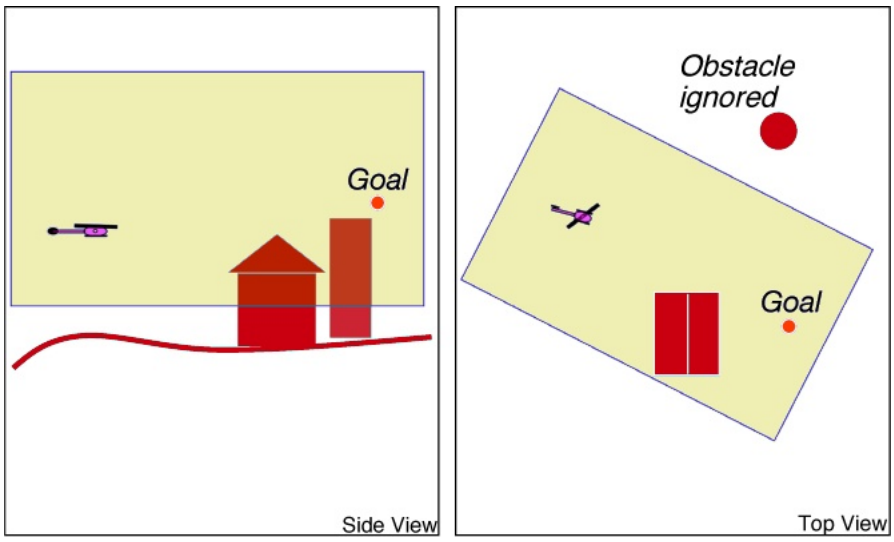


Figure 3.10: An example showing the box of attention. The box restricts the obstacles considered and is set between the goal point and the current location of the UAV.

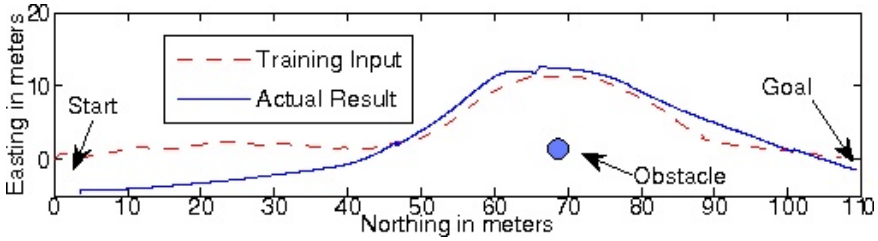


Figure 3.11: Result of training the obstacle avoidance parameters. This figure shows the path used for training as a dashed line and the path taken by the RMax helicopter as a solid line. The path with the fine-tuned learned parameters still is a adequate match for our training path.

The obstacles considered by our reactive algorithm are additionally limited by a box-shaped constraint as shown in Fig. 3.10. The reasons for further constraining the obstacles that are considered are three-fold. First the box decreases the minimum altitude of the UAV because it only extends a little bit(5m) below the helicopter. Second since the yaw axis of the box is defined by the location of the robot and the current goal point the box ignores obstacles that are not in the relevant state space for obstacle avoidance because they are not in line between the UAV and the goal point. If an obstacle was relevant and we would have to swerve more around we would eventually see the obstacle and still avoid it. Third, the box also reduces the amount of processing because only obstacles inside the box need to be considered for obstacle avoidance. The grid restricted by the box typically contains on the order of 30000 cells or approximately 0.7% of the total number of cells in the evidence grid. The evidence grid without the box contains 256x256x64 cells and has a resolution of 1 m.

Determining the parameters Our control law has a large number of parameters that need to be set to generate a desired behavior. Overall there are 12 constants

$$\overline{u} = (k_g, c_1, c_2, s_1, s_2, t_1, t_2, k_{o,1}, k_{o,2}, c_{3,1}, c_{3,2}, c_{4,2}) \tag{3.16}$$

Some of the parameters have an intuitive meaning and are defined by the problem domain but some of the parameters are tedious and difficult to set. The goal following parameters k_g, c_1 and c_2

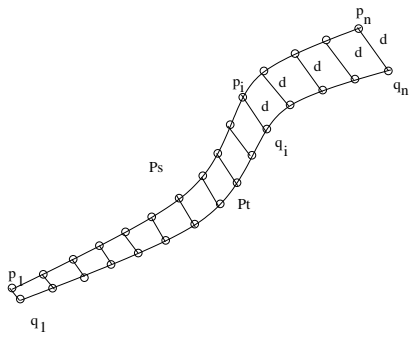


Figure 3.12: The difference between two path segments is used to optimize the parameter set $\bar{u}$. P_s is a recorded path segment while P_t was generated from a parameter set $\bar{u}$.

were determined by hand since we had a desired performance for pure goal following and tuning this subset is intuitive. The values of s_1, s_2, t_1, t_2 are used to shape the number of obstacles considered and were therefore also fixed.

In order to learn the remaining parameters our pilot flies the helicopter and tries to follow a straight line between a start and goal point while avoiding a pole obstacle in one axis. The resulting training data is shown as a dashed line in Fig. 3.11. Data about the goal point, the obstacles, and the flown path are recorded and used to determine the unknowns of the described control model. The input to the control model and human subject at any point in time is a goal point p_g and a set of obstacles O. The pilot flies the helicopter pretending he is seeing the obstacles only when the algorithm actually uses the the obstacle information.

Our training example is chosen to not require any sink or climb maneuver. This reduces the number of parameters that need to be learned, because only the horizontal commands determine the behavior:

$$u_e = (k_{o,1}, c_{3,1}, c_{4,1}) \tag{3.17}$$

Given a set u of parameters, we generate a path $P_t = \{q_i = (k_i, l_i) | i = 1..n\}$ with the same n number of points as the training path segment P_s, which contains regularly sampled points in the plane. $P_s = \{p_i = (x_i, y_i) | i = 1..n\}$.

The error between the two paths is defined as the Euclidean distance between each point pair as shown in Fig. 3.12: $d(\bar{u})_i = \sqrt{(k_i - x_i)^2 - (l_i - y_i)^2}$. Consequently, the total error minimized between two path segments is $D(\bar{u}) = \sum_{i=1}^{n} d(\bar{u})_i$. The optimization procedure minimizes the error term $min_{\bar{u}} D(\bar{u})$.

The path P_t is generated from a forward simulation of the commands sent to the robot (See Section 3.3.2 for the simulator). Since the length of the path and velocities are not controlled in this model, we use the recorded speeds to ensure P_t has the same length as the training path P_s. Since the space of parameters has many local minima we do a random search of parameters uniformly distributed between 0 and 10.

The model of the helicopter used for training is not perfectly accurate and therefore it was necessary to fine tune the parameters on the actual helicopter to improve the behavior of the real system. We varied the value of k_o systematically between $100\% - 150\%$ to fine tune the behavior on a set of test cases.

It is sufficient to use only one trajectory to train the three parameters because we are only fitting three parameters that determine the desired response with respect to the pole obstacle. In this case the optimization tries to fit how much $k_{o,1}, c_4$ and when c_3 to react to the obstacle.

Figure 3.11 shows the path the autonomous helicopter took overlaid with the training input from our pilot. The parameters used after fine-tuning still adequately matched the prediction. The initial conditions are not the same since the operator who was controlling the aircraft was not able to see the actual start point precisely.

It is interesting to note that even though we trained our method only on this simple example the parameters generalize to more complicated scenarios with trees, large buildings, and wires.

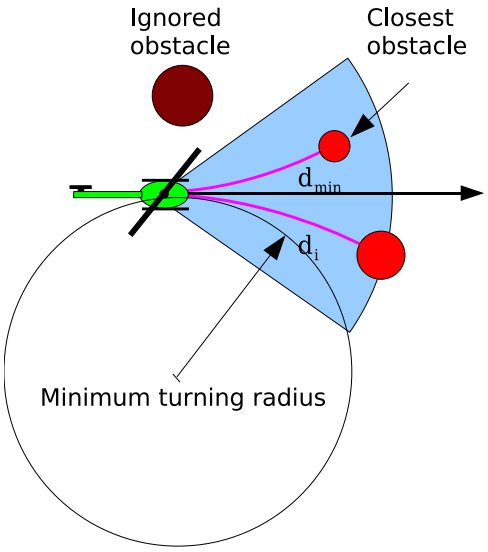


Figure 3.13: The speed controller slows the robot based on the closest reachable obstacle. Reachability is determined by the minimum turning radius based on the current velocity and the maximum turning rate of $30°/s$.

Generalization The proposed control law generalizes well in the different environments and vehicles we have tested it in. The space of parameters appears expressive enough to adjust the control law for the desired behavior of the system.

The modified FW method was trained using a very simple pole obstacle at a desert site in Phoenix, Arizona and tested using more difficult obstacle at a different test site in Fort Benning, Georgia, which was wooded with many buildings and wires. It was not necessary to changes the parameters or the algorithm. This is one indicator that the control law is general enough to be used in different environments.

In prior work we evaluated a variant of the Fajen and Warren control law on a ground vehicle [Hamner et al., 2006]. In this work we also examined the effectiveness of different learning algorithms to fit parameters for this control law. We observed that even though we trained the control law on simple avoidance cases the method generalized to more complex obstacle configurations.

3.2.4 Speed Controller

The last resort of safety in our collision avoidance is the speed controller. Since the magnitude of the speed is always set to be within the safe region and we start within a safe region by induction collision free operation can be guaranteed as long as the model of speed control is conservative in terms of deceleration. The safety limits the performance because we are operating within a stopping distance regime. It is assumed that all obstacles in the flight path of the vehicle are visible. We assure this partly by insisting that the helicopter flies without instantaneous sideways motion. In very cluttered environments, occlusion can lead to pockets that are not visible and are not viewed unless the sensor can always point in the direction of travel.

The desired velocity magnitude is modified based on three constraints: desired user speed, vehicle speed limits, and obstacle distances. First, the user-specifies a speed between waypoints that must not be exceeded. Second, the robot itself has a $3m/s$ climb rate limit and a $1m/s$ sink rate limit to prevent settling with power and excessive load. The resulting command vector is scaled if these limits are exceeded.

Third and most importantly, the closest forward reachable obstacle limits the speed of the robot. The distance to this obstacle is used to determine a maximum speed based on stopping distance. Stopping distance is a constant deceleration and reaction time approximation to the actual stopping

dynamics of the vehicle. The formula for stopping distance is

$$d_b = vt_r + \frac{v^2}{2a_{max}} \tag{3.18}$$

where t_r and a_{max} have to be determined based on the vehicle dynamics.

The inverse can be calculated as

$$v_b = -a_{max}t_r + \sqrt{2a_{max}d_m + a_{max}^2 t_r} \tag{3.19}$$

Using the inverse one can determine a speed v_b at which it is possible to stop if a stop command is issued within t_r. If this speed is set iteratively the robot will slow down if it gets closer to an obstacle. Additionally since it is less likely to turn into obstacles that are away from the current direction of travel one can increase the speed by

$$v_b' = \frac{v_b}{cos(\theta)} \tag{3.20}$$

where θ is the angle of the obstacle to the current direction of travel. Increasing the speed to v_b' is an optimistic assumption in our algorithm and is an attempt to avoid double counting of obstacles since our local planner should avoid all obstacles. If the speed control is pessimistic the interaction between steering and speed control increases the gain of turning and can lead to an overreaction to an obstacle. However since we do not completely trust the local planner we slow the vehicle when we are flying directly at an obstacle. The modification is not safe in the sense that if the local algorithm should decide to fly into an obstacle the speed controller might not be able to stop in time.

Not all obstacles are considered to determine the distance d. Instead as shown in Fig. 3.13 all obstacles in the current direction of travel that can be reached by performing the tightest turn are considered. The minimum turning radius is determined by the current speed and the maximum turning rate. It is given by

$$r_t = \frac{v_c}{\dot{\theta}_{max}} \tag{3.21}$$

where $\dot{\theta}_{max} = 30°/s$

This radius is used to calculate the forward reachable field of view. The reachable field of view limits the obstacles considered for speed control.

$$\Delta\theta = \frac{r_t\pi - d_b}{2r_t} \tag{3.22}$$

For the obstacles in this field of view O' the arc length is calculated as follows

$$d_o = 2 \cdot d \ \max(\phi, \theta) \frac{\sin(\frac{\pi}{2} - \max(\phi, \theta)}{\sin(2\max(\phi, \theta))} \tag{3.23}$$

The closest obstacle is consequently the minimum of all the obstacles that are considered:

$$d_m = min_{o \in O'}(d_o) \tag{3.24}$$

which is used to calculated v_b.

We performed a simulation of the helicopter flying into a virtual wall because we wanted to test if the speed controller could safely stop the helicopter without it hitting an obstacle. In Fig. 3.14 one can see the actual speed of the helicopter as it approached the wall. Before reaching the wall the speed drops to 0 and the helicopter stops. The commanded speed decreases as the distance to the obstacle decreases . The maximum braking acceleration used in our system was $a_{max} = 2.4\frac{m}{s^2}$ and the reaction time is $t_r = 1.1s$.

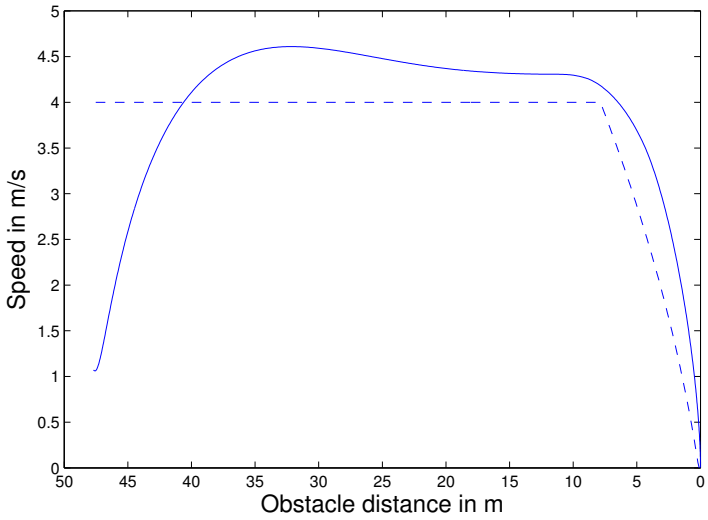


Figure 3.14: Experiment that shows the speed controller stopping to avoiding collision. The solid line shows the actual speed of our physical simulation as the helicopter approaches a wall. In this example only the speed controller is avoiding obstacles because the local and global planning algorithm were disabled. The dashed line shows the commanded speed from the speed controller determined by the closest point to the wall. The helicopter stops without hitting the wall.

3.2.5 Mission Execution

The algorithms we described so far are only able to reach a particular goal point. We add another sequencing layer that allows us to visit a sequence of waypoints. The helicopter should hover at each location and then continue on to the next point. We therefore first track the start point of the current waypoint pair and if we have sufficiently tracked it we will try to reach the goal point of that pair using the described architecture. Sometimes it is not possible to reach a specified goal point. We designed our finite state machine to avoid getting stuck at a certain point in the sequence. After a finite amount of time a mission will be finished or we have entered an error state. We achieve this with time-outs and a goal point tracking algorithm that will not collide with obstacles. First the algorithm gets a start and a goal point from the current sequence of waypoints. Then it uses the start point and direction to the goal point in the tracking algorithm to align itself with the goal point. If there are obstacles between the current position and the start point it only orients itself with the new goal direction and does not try to track the original position. This behavior ensures that an unreachable goal point or an unexpected obstacle in the straight line path will not cause a collision.

After the helicopter has achieved all conditions it switches to the obstacles avoidance algorithm. Once the obstacle avoidance algorithm has reached the goal point it switches to get a new leg with the old goal as the new start point and a new goal point. If this was the last leg it will go into the end state and stop.

Each transition is based on jump conditions. The finite state machine describing the states of the helicopter is shown in Fig. 3.15. Six labels a-f represent conditions on the transition between states described in Table 3.2. The global timeout is a global maximum amount of time a segment can take and is based on a multiple of the product between the straight line distance of a waypoint pair and the desired speed. This timer is started as soon as a segment is started. It ensures that eventually a transition to the next segment is guaranteed. The local timeout is started once the robot gets within a certain radius of the goal point and causes the robot to abandon a waypoint if it cannot get close enough to the goal after a reasonable amount of time.

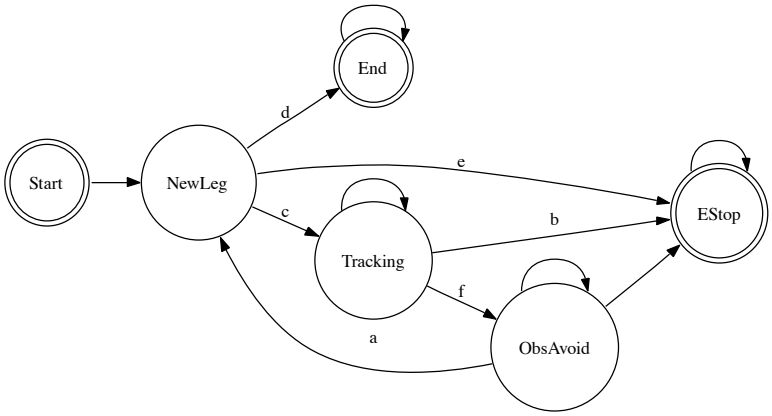


Figure 3.15: Finite state machine of the sequencing algorithm. The state machine begins execution in the "start" state and has two end states. "End" and "EStop." The conditions of the edge labels are shown in Table 3.2. “NewLeg” creates a new waypoint segment, “Tracking” holds the position of the first point if it is reachable, and “ObsAvoid” performs the obstacle avoidance.

Label	Condition
a	OR Global timeout has been exceeded: $t_{gl} > t_{globallimit}$ Local timeout has been exceeded: $t_{lo} > t_{locallimit}$ Goal is not reachable within 20m. Braking distance is larger than distance to goal: $d_b \geq d_g$
b	An exception happened.
c	A new segment is available.
d	No more segments are available.
e	No path has been uploaded.
f	AND Received OK to continue from base station? Oriented correctly to new direction: $\Delta\theta < \Delta\theta_p$ Close enough to goal point: $\Delta d < \Delta d_p$ Have tracked at at least 5s: $t_{track} > 5s$ Starting from near standstill: $v < 1\frac{m}{s}$

Table 3.2: Conditions of the edge labels for the state machine shown in Fig. 3.15.

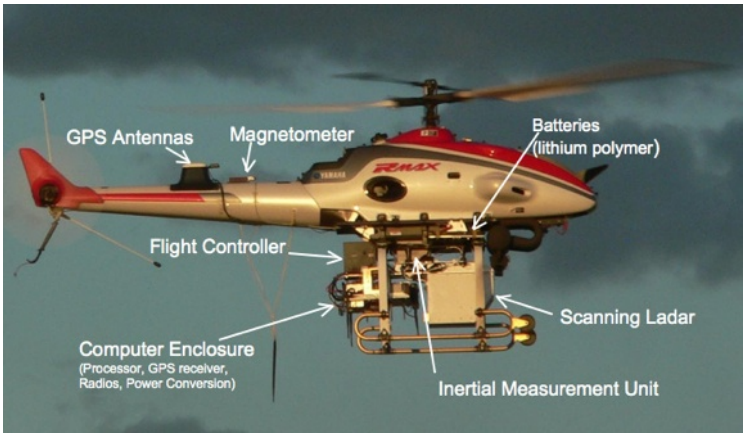


Figure 3.16: Helicopter testbed, a Yamaha RMax, used for our experiments has a payload of 29kg + fuel.

Max Payload*	31kg
Max Takeoff weight*	94kg
Main Rotor Diameter	3.1 m
Tail Rotor Diameter	0.5 m
Overall Length (with rotor)	3.6 m
Overall Width	0.7 m
Overall Height	1.1 m
Engine Type	Water-cooled, 2-stroke, 2-cylinder
Engine Displacement	246cc
Maximum Torque	2.6kgm
*(at 1 bar, 35^o C)	

Table 3.3: Specifications for the Yamaha RMax industrial remote-control helicopter.

3.3 Experiments

3.3.1 Testbed

3.3.1.1 Helicopter and Flight Control

We fitted a Yamaha RMax helicopter (Fig. 3.16), with a commercially available flight controller. The resulting system provides a reliable platform capable of carrying a comparatively large payload (29 kg @ 1200 meters elevation). The flight control system provides a robust velocity loop even in the presence of strong gusty winds. See Table 3.3 for more specifications.

The input from our algorithms was sent to a flight controller developed by the Swiss company weControl [weControl AG, 2006]. We tested the RMax+flight controller system at velocities up to 15m/s and weights up to the full payload capacity of the RMax.

One problem with using a larger helicopter is that it is susceptible to problems that normally don't occur with smaller research platforms. One potential problem is the flight mode known as a *settling with power*. This dangerous situation is caused when vortices form around the edge of the rotor disk and push the helicopter downwards. The more power the pilot adds to correct the sudden drop in lift, the stronger the vortex becomes. Eventually the helicopter will stop flying and fall to the ground. This failure usually occurs when the helicopter descends too quickly in still air and then tries to slow down. Full-scale helicopter pilots avoid this problem by maintaining a forward airspeed during descent. The flight controller has no knowledge of these constraints. We added a watchdog program between the output of our navigation system and the flight controller that overrides commands that could cause a power-on descent.

Field of View	30^o elevation 40^o azimuth
Oval Scan Rate	22 Hz
Frame Rate	1.33 Hz
Sample Rate	64kHz
Angular resolution	2 mrad
Range resolution	< 2 m
Beam divergence	2 mrad
High-Power Range	
6mm black wire	58 m
Blind Range	14 m
Low-Power Range	
6mm black wire	38 m
Blind Range	9 m
Safety Class	1

Table 3.4: Specifications for the Fibertek 3-D laser scanner. The early detection of wires makes high-speed obstacle avoidance possible. We tested detection range at a low and high power setting. At low power detection range is smaller while at high power thin wires are detected earlier. The minimum range of detection increases with the output power.

3.3.1.2 Sensor

We used a custom 3-D laser scanner from Fibertek Inc. with a 30x40 degree field of view to sense the environment. The scanner sweeps an oval pattern back and forth as shown in Fig. 3.17. This pattern is designed to facilitate detection of wires in many orientations. By contrast, a scanner that sweeps a straight scan line back and forth will have difficulty detecting a wire that is aligned parallel to the scan line.

Originally designed to operate as an operator aid for human pilots, this pattern is particularly suited to the detection of wires in many orientations as shown in Fig. 3.18. This sensor facilitates collision avoidance because it is sensitive enough to detect obstacles at distances much greater than the stopping distance of the helicopter at the maximum speed. For instance it can detect typical powerlines at 100-150 m while the stopping distance is 40 m at a speed of 10 m/s.

Table 3.4 shows the specifications for the scanner. With a large detection range and high frame rate, the sensor is well suited for obstacle detection, however there are two main drawbacks of this sensor. The first is the blind range. The current output window is not perfectly matched to the wavelength of the laser, so there is backscatter back into the receiver. The receiving unit must be blanked to keep from being blinded by the pulse. At high power, this blanking time results in a blind range of 14 meters. The routines that build the world map must take this blind range into account. Also, this blind range increases the minimum radius of turns in order to guarantee that the helicopter will see an obstacle as it comes around a corner.

The second problem with the ladar is sensitivity to dust and other airborne particles. The sensor sees clouds of pollen when flying in forested areas of Ft. Benning. This sensitivity to dust could be minimized by waveform analysis of the return signal, and is an active area of research for Fibertek.

3.3.1.3 State Estimation

The Novatel state estimation system provides 100Hz estimates in six degrees of freedom. This estimate is crucial for accurate registration of the ladar data into a coherent world map. An HG-1700 ring laser gyro inertial measurement unit(IMU) provides rate and acceleration data to the system. We used differential GPS to provide 1cm accuracy in the position estimate. The map does not have to be centimeter accurate to dodge an obstacle; however, we used the best system available in order to test the obstacle avoidance algorithms themselves with good ground truth for analysis.

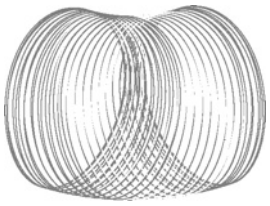


Figure 3.17: Ladar scan pattern of the Fibertek scanner. The small circle is scanned at 22 Hz while it it swept across the field in 0.75s. The laser rangefinder measures distance at a frequency of 64 kHz.

Figure 3.18: A scene generated from a series of static scans of the environment with buildings, trees and powerlines. The powerlines are visible at 170m.

3.3.1.4 Computing

All processing of sensor data, path planning and collision avoidance is performed on a Linux computer with a Pentium M processor at 1.6 GHz, 2 GB of RAM and 4 GB compact flash. The onboard accelerated graphics processor reduces computation of the virtual range sensor.

3.3.2 System Identification

We created a simulation model that approximates the dynamics of the velocity-controlled helicopter. The model is used for algorithm development and software validation. It is essential for learning the parameters of our reactive algorithm because we have to consider the closed-loop response of the system to obstacles. Also see section 3.2.3 for details on the training procedure.

3.3.2.1 Model

The flight control system described in section 3.3.1.1 controls the velocity of the helicopter. For our simulation, it is sufficient to characterize the dynamics of this velocity-controlled system. Ideally, the flight controller provides four independent degrees of freedom.The controller attempts to minimize coupling between the degrees of freedom and we can therefore use a single input single output (SISO) frequency-domain model. The model is given by

$$H_i(s) = \frac{b_0 s^k + b_1 s^{k-1} + \ldots + b_k}{s^k + a_1 s^{k-1} + \ldots + a_k} e^{-st_d} \tag{3.25}$$

where $H_i(s)$ is the frequency response of system i to the complex frequency s, a_i and b_i are real scalar coefficients, and t_d is the time delay. In the case where the numerator is of lower order than the denominator, the corresponding b_i are defined as 0. We represent latency separately to reduce the needed order to represent each system. The time delay term e^{-st_d} is explicitly modeled as a delay queue in the input to the simulation.

3.3.2.2 Experiments

We performed SISO system characterization for each of four degrees of freedom(DOF) on the velocity-stabilized helicopter: longitudinal, lateral, vertical, and yaw DOF. Characterization of each DOF consisted of the following steps:

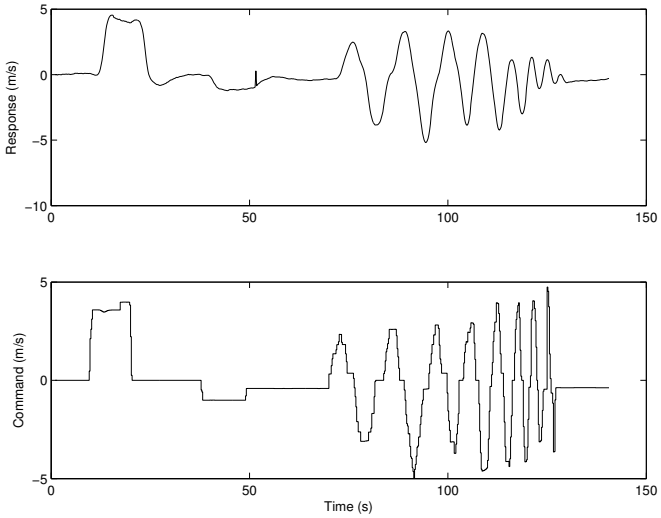


Figure 3.19: Example result of a frequency sweep experiment for dynamics characterization. This data is used to fit a dynamic model of the system for simulation.

	a_1	a_2	b_0	b_1	b_2	t_d	(FPE)
Longitudinal	1.03	0.70	0	0	0.75	1.58	0.15
Lateral	0.81	0.60	0	0	0.58	1.22	0.29
Vertical	1.28	1.28	0	0	0.93	1.06	0.08
Yaw	2.21	4.03	0	0	4.19	0.36	4.22

Table 3.5: Parameters of the SISO dynamic model of the velocity-controlled helicopter

1. Perform frequency sweeps near the point of linearization.
2. Fit linear model to data.
3. Evaluate fit on separate data.

We used the Matlab System Identification Toolbox to do the fit and evaluation. An example data set for longitudinal dynamics is shown in Fig. 3.19. All of the SISO dynamics could be well approximated by a 2^{nd} order system with two poles and no zeros. Table 3.5 shows the resulting systems. The final column, FPE, is the Akaike *Final Prediction Error* for the fit.

3.3.2.3 Evaluation of model

There were a few problems with the simulation fidelity; however, the SISO approximation proved sufficient for our needs.

There was observable coupling between different degrees of freedom. Figure 3.20 shows the inputs and outputs for each degree of freedom during an actual flight and a flight recreated in simulation using the same recorded control inputs. In this test, a constant forward velocity is given while the heading rate command is swept back-and-forth with increasing frequency. The commanded velocities for lateral and vertical motion are constant at zero. In an ideal diagonalized system, one would expect to see the system remain at this commanded zero; however, there is actually quite a bit of coupling. The lateral coupling is somewhat noisy. The vertical coupling is surprisingly repeatable and strong in this case. Figure 3.21 shows the trajectories recorded for this test. Despite the coupling between axis, the general shape of the simulated trajectory is close to the recorded path.

Wind was an unmeasurable factor for both of the shown tests. While we tried to do such evaluations in calm weather, we invariably had 3-6 knot (1.5 to 3 m/s) winds that interfered with measurements.

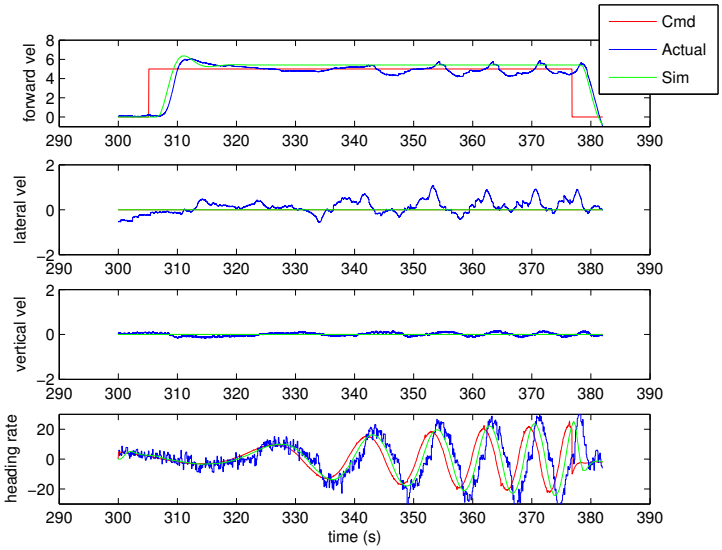


Figure 3.20: Comparison of simulated velocity to actual telemetry. These graphs show coupling between forward velocity and turn rate with altitude and lateral dynamics. The trajectories for this experiment are shown in Fig. 3.21.

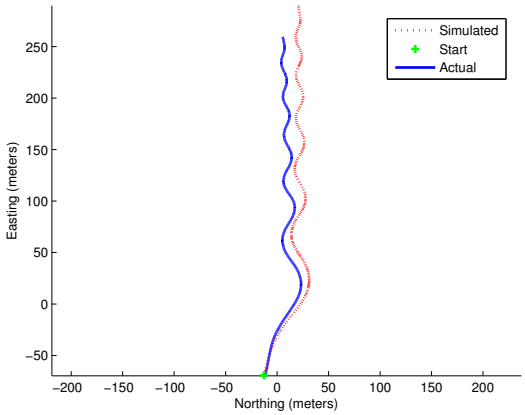


Figure 3.21: Comparison of a simulated and actual path flown by the helicopter. The general shape of the two trajectories are quite similar, though errors add up over time. The velocities and commands that generated these paths are shown in Fig. 3.20.

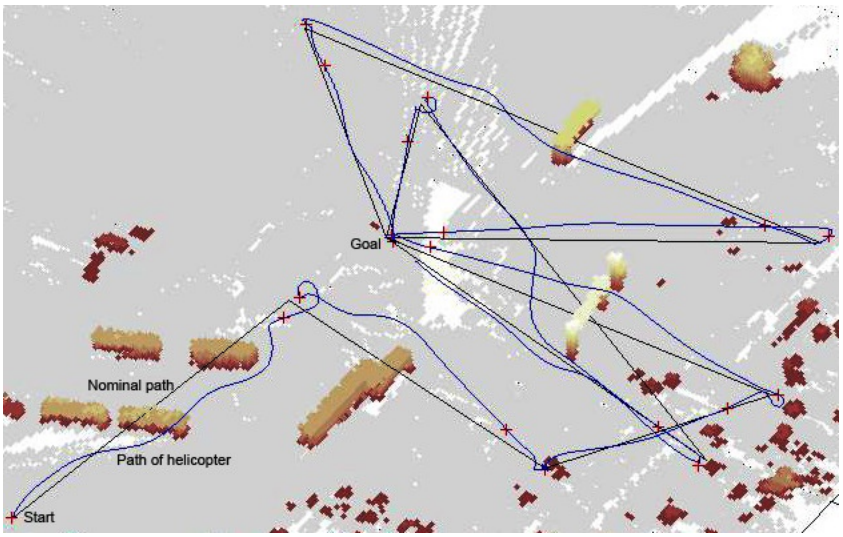


Figure 3.22: A long flight sequence over various obstacles at the Phoenix test site at $6m/s$. The environment had a series of containers, three poles and a wire as obstacles. In the beginning of the sequence two containers are stacked resulting in an overall height of 17 feet. The last stack of containers is stacked to a height of 25.5 feet.

Another problem with the modeling technique we used was the inability to model non-linear effects. The helicopter displays different flight dynamics at different speeds and flight configurations. Altitude dynamics are significantly effected by relative airspeed and helicopter attitude. For example, in a sudden stop the helicopter will pitch up and tend to climb. The dynamics are also different for climbing vs. diving.

Ideally we would solve these problems of nonlinear effects by using a full dynamic model of the helicopter. However, this is impractical in our case because the commercial *wePilot* flight controller is essentially a black box with unknown and possibly time-varying dynamics. Any hi-fidelity nonlinear model of the helicopter would be useless in simulation without a similar model of the control system. Alternatively, varying speed effects could be modeled by a linear system with coupling, but eventually we would have to use multi-linear approaches to model the changes in dynamics over time.

The current simulation recreates the helicopter dynamics well enough for the application of verifying the obstacle-avoidance routines offline. If future applications require, we will develop a multi-input multi-output linear model that incorporates coupling.

3.3.3 Results

In this section we will show some results demonstrating the overall behavior of our system at two different test sites. The runs shown in the figures were completely autonomous with varying speeds. At the start point the robot had no information about the obstacles or the ground. The UAV built the map incrementally as it was avoiding obstacles. The minimum speed was $4m/s$ and the maximum commanded speed was $10m/s$. The connection between distinct segments often shows loops and other behavior. This is due to the fact that the robot is trying to track and hold the point. Wind can blow the robot and cause it to move when it is tracking goal points.

We performed over 1000 successful obstacle-avoidance legs on our helicopter testbed in up to 24 knot winds. Our layered architecture allowed the reactive layer to quickly respond to obstacles, while the deliberative layer found a good general path. On several instances, this ability to react quickly saved the helicopter from collisions with wires that the sensor could only register at short ranges. On the other hand, the global planner was invaluable for finding routes around very large objects, such as tree lines or buildings, where the reactive layer would have a tendency to oscillate or get stuck if left to itself. Finally, in cases where both collision avoidance measures failed (during early

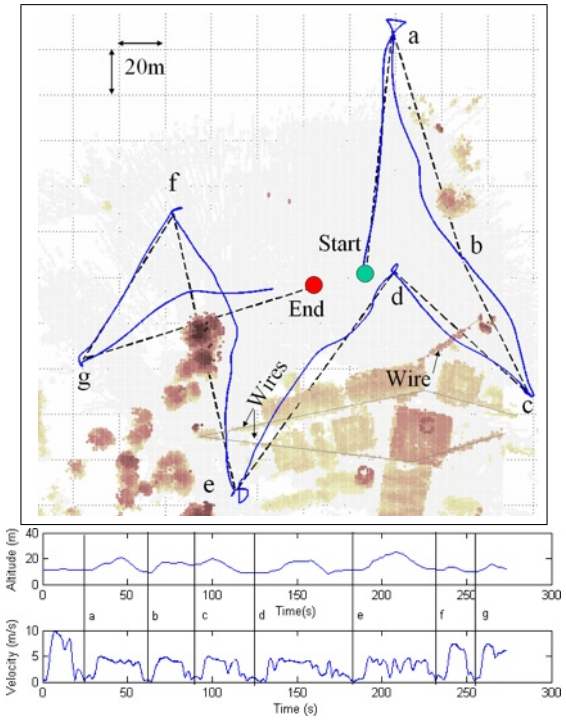


Figure 3.23: Flight through the McKenna MOUT site at Ft. Benning, GA at 4 m/s. 8 m/s was commanded at the first and 6 m/s at the last two segments of the path.

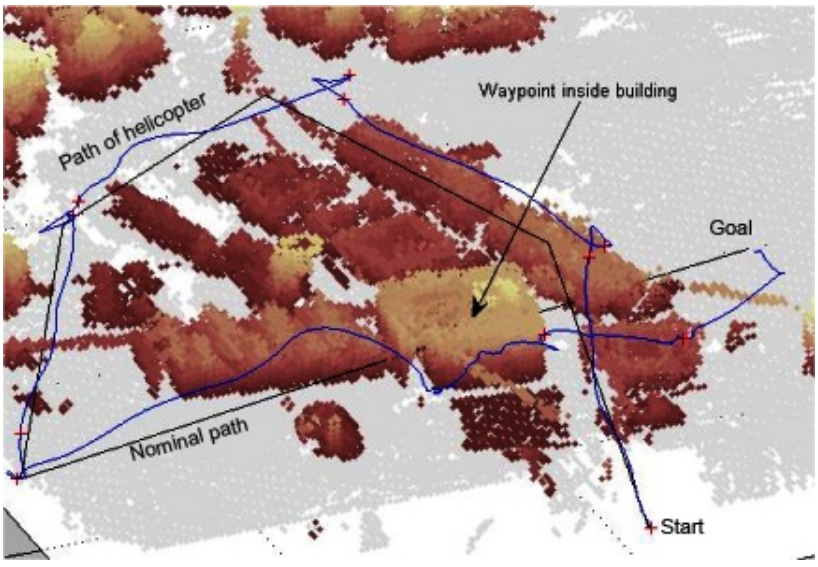


Figure 3.24: Another McKenna MOUT site demonstration at $4m/s$. This one shows the behavior of the system with an unachievable waypoint located inside a building.

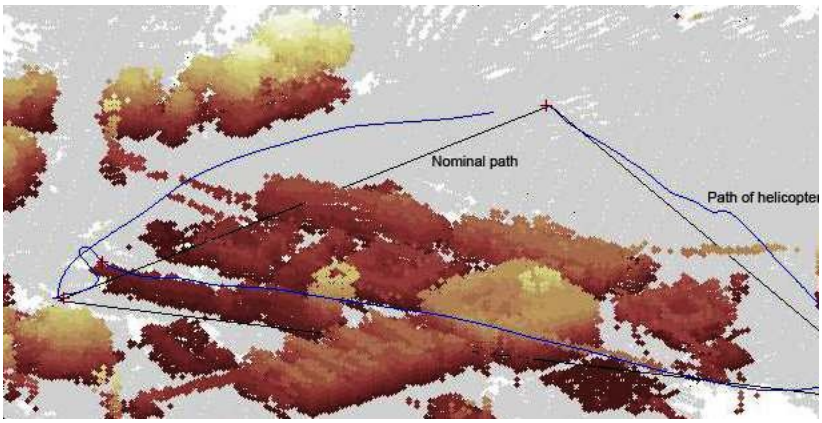


Figure 3.25: A third party provided a previously unknown waypoint sequence. The plot shows a flight of this waypoint sequence at the McKenna MOUT site at $4m/s$.

development due to software bugs), the speed control system brought the helicopter to a stop before a collision occurred.

The development phase required 600 autonomous runs to integrate and test the entire system. Once we reached a final configuration, we ran more than 700 successful obstacle avoidance runs, some of which ran in long continuous sequences that maintained autonomy for the entire maximum flight duration of 35 minutes. Only one error (discussed below) required aborting a run. Other aborts were caused when the safety pilot was not able to keep up with the helicopter.

In Fig. 3.22 the helicopter is commanded to follow a long sequence of obstacles at our first test site in Phoenix, AZ (USA). The evidence grid is shown in red. Several stacks of containers and three poles with a wire obstruct the straight lines between waypoints. In the first four segments the helicopter flies over the containers and a wire between poles. In the next segment it avoids a high row of containers by flying around them. During the second approach to the wires and poles it avoids the wires by turning because the approach is diagonal. In all cases our local planner has some climbing and some turning behavior. However depending on the configuration the turn or climb can be very minimal. One can also see that a thin reflective wire is detected and avoided without problems.

At our second test site at the McKenna MOUT site we also determined waypoints of paths flying above the town and through some large trees. Even though the environment is very different from the desert like terrain in Phoenix the behavior was still as desired without modifications to the algorithms.

In Fig. 3.23 the helicopter follows a sequence of waypoints through trees, a wire and over a town. At waypoint e the helicopter is required to fly to a fairly tight hiding spot between trees and buildings. There are approximately 10 m between the obstacles and the location of the helicopter. The altitude (~8m above ground) is so low that the helicopter is not visible nor audible from the start of the path (Also see Extension 1).

Notice also how the system begins to avoid obstacles in both the vertical and horizontal axis before committing to one or the other. This is a property of the local planning algorithm as shown in Equation 3.11, which begins to evade with both degrees of freedom until one direction becomes clear of obstacles. This behavior allows the system to implicitly decide which path provides the closest safe path as a function of vehicle dynamics. For example, in Fig. 3.23, the path between points d and e is blocked by a long building. The system begins to turn to go around it, and at the same time begins to climb over. As the vehicle climbs, it encounters a free path on the vertical axis. The horizontal avoidance component quickly drops off, and the vehicle follows a vertical avoidance trajectory. The converse happens on the leg between g and *End*. The system chooses to fly around the tall tree rather than climb over it.

In Fig. 3.24 we demonstrated the behavior in case of unachievable waypoints and reconnaissance along a boulevard. The robot first flies along the main street of the town and then goes to a point

outside the town. From there it is commanded to fly into the large building at the center of the town. Since that waypoint is sensed to be unachievable the robot does not try to get closer to it after some distance and proceeds to the next goal point on the other side while avoiding the wire.

In two other tests the robot was given a sequence of waypoints from a third party and it had to fly the paths without any preparation or tests. Both tests were successful and the robot flew the paths without intervention on the first try. A plot of one test is shown in Fig. 3.25.

During the final testing phase of 700+ runs, we encountered only one dangerous bug in the behavior, twice in the same location. In a cluttered environment sensor occlusions are quite common and leave holes in the evidence grid. The path planner will consequently plan a path through unseen obstacles. This is not a problem as long as the sensor covers this unknown area before the vehicle traverses it, as the local planner will react immediately while the planner finds a new route. In the error case, the planned path went through a large patch of ground. The geometry of the scene and sensor FOV prevented the sensor from seeing the patch before the helicopter started descending in the direction of the ground. This behavior was rare, as any holes in the evidence grid are too small to fly through without having the local planner cause an evasion. While the simple addition of a ground plane in the evidence grid would have eliminated this behavior, we believe it is essential that a UAV is able to point the range sensor in the direction of travel. Another safeguard would have been to set the speed by treating unknown cells as obstacles.

Another perception problem is that of dust. The laser that we use is very sensitive so that it can see wires from large distances. Unfortunately, dust and pollen can have the same signature as a small wire. Despite some adjacent point filtering, observed dust clouds would occasionally divert the helicopter's flight path. Eventually the evidence grid would clear these clouds using negative evidence. This false-positive error does not cause dangerous behavior, but can impede low altitude flight. On windy days in dusty areas the system chose to fly higher to avoid dust clouds.

A ceiling or upper-limit of flight is helpful in cases where the helicopter has to stay low for stealth or low-level observation; however, forcing a robot to observe the ceiling can result in an impasse (such as coming to a long tree line which extends above the ceiling). We therefore force only the path planning algorithm to respect the ceiling constraint, while the reactive algorithm has no such constraint. The system will obey the constraint if the planner can see a way around, but otherwise will do what is necessary to avoid obstacles and continue the mission. An example of this situation during actual flight is shown in Fig. 3.26(Also see Extension 2).

Figure 3.27 compares the flight paths of various combinations of the local and global planning algorithms in avoiding a wire and pole. The local planner waits to veer away from the pole, while the global planner veers as soon as it notices the obstacle. The combination is approximately the median between the two. The vehicle steers toward a goal point farther away and undercuts the wide turn the global planner generated.

The optimized global planner chooses the furthest line-of-sight point on the Laplacian-generated path and redraws the path as a straight line to that point. In this case, that point is almost at the final goal point, so the local planer is primarily responsible for the obstacle avoidance behavior, as can be seen in this example.

The combination of a local and global planning algorithm leads to an improved behavior in difficult situations but also improves the general behavior because obstacles are avoided before the reactive algorithm is really repelled by them.

3.4 Discussion

We have developed a first-of-a-kind capability suited for UAVs implemented on a helicopter that avoids obstacles of various sizes and shapes while flying close to the ground at significant speeds. In our experiments, the uninhabited helicopter started with no prior knowledge of the environment, having been provided only a list of coarse waypoints separated by up to hundreds of meters. The straight line path between the waypoints often intersected obstacles. While we regularly ran collision avoidance close to buildings, trees and wires between 4-6 m/s, the system was tested at commanded speeds up to 10 m/s. To accomplish these results our system uses a fast avoidance method that stays

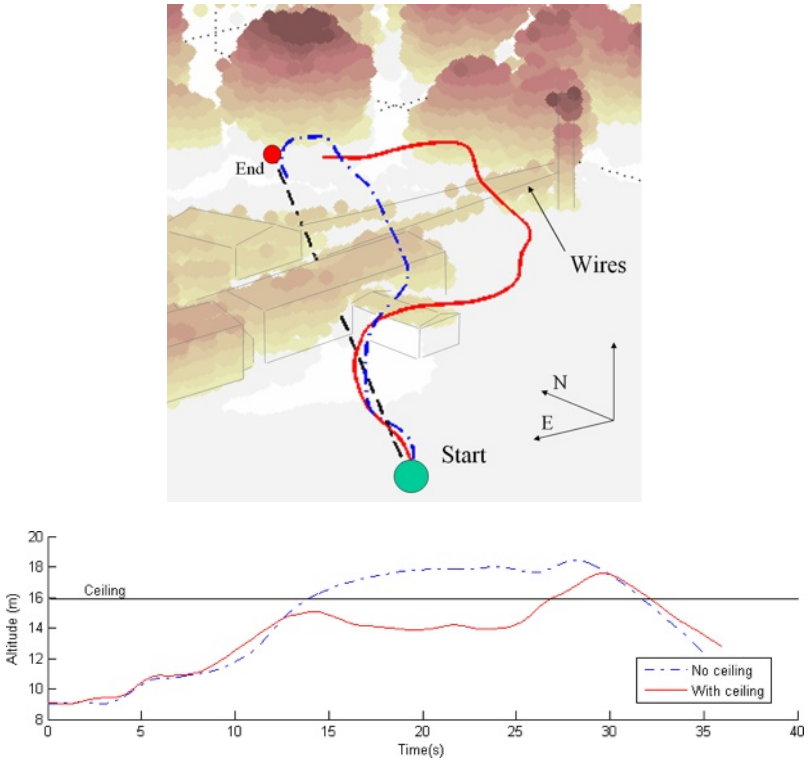


Figure 3.26: Flying with and without ceiling when specified path is directly through a building. If a ceiling has been specified, the vehicle tries to stay below a specified altitude. On the run with ceiling, the system begins by flying around buildings, but then ignores altitude constraint and climbs to safely clear power lines. Key: Blue(dash-dot)→No ceiling. Red(solid)→With Ceiling.

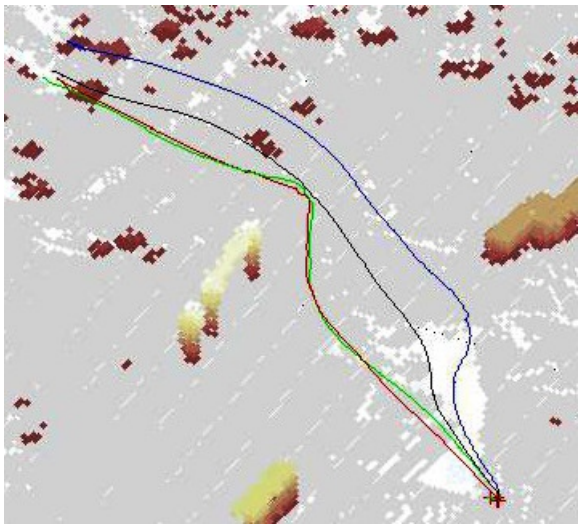


Figure 3.27: A comparison of the paths taken with different combinations of the local and global planner. Key: Blue→Just global planner. Red→Just local planner. Green→local+ global planer. Black→local + optimized global planner.

away from obstacles intelligently coupled with a global planning method that suggests a direction of travel.

Our architecture separates global planning from local planning and can therefore achieve the reaction time of a simple control law with the intelligence of a more deliberative approach. Furthermore the failure of one method should still keep the robot safe. We automatically determined the parameters of the local planning algorithm from a piloted training example.

We intend to address a few issues in future work. Our current method is not built to scale with significant increases in speed and avoids the obstacle with a margin that is irrespective of speed. Ideally the reaction should depend on speed to react earlier to obstacles if the robot is faster and later if it is slower. Another issue is the tradeoff between sensitivity to small obstacles and an excessive reaction to large objects close by (such as in an urban canyon) even if they are not in the path of the vehicle.

Furthermore we want to address that currently the repulsion from obstacles is exponential and additive and therefore sometimes exceeds the physical constraints of the vehicle. Too large commands can cause the vehicle to oscillate between one and another extreme reaction. Another improvement we would like to address in future work is that the goal point is picked at a fixed distance ahead of the vehicle. This causes the UAV to follow the path but also gives it room to avoid obstacles. Since the distance to that point is significant the vehicle can cut corners in tight situations which will cause it to go towards obstacles that are already circumvented by the path.

Our speed control method is still very conservative and limited by stopping distance. In future work we would like to be less conservative and be limited by swerving distance instead. This however requires us to rely more on the local planning algorithm.

Currently the flight altitude of the helicopter is mainly determined by the altitude of the goal point. However if the helicopter gets too close to the ground it is repelled and will climb. The box limits the obstacles considered below the ground plane to 6m. It is therefore not possible to fly closer to the ground. In future work we would like to improve our implementation to be able to automatically set an altitude above ground from defined waypoint latitude and longitude to allow nap of the earth flight.

Since path planning already produces a plan that avoids obstacles it is not necessary for the reactive algorithm to additionally avoid those obstacles. In future work instead we should only react to unexpected obstacles. As a future extension we propose a tunnel that defines the set of paths that are obstacle free at the time of planning. If unexpected obstacles should enter this region the reactive algorithm will avoid these obstacles while respecting the tunnel. The tunnel can also be modified by the prediction of the reactive algorithm if it is necessary to leave the tunnel.

4 Efficient Calculation of Obstacle Cost

The main contribution of this chapter is an incremental algorithm that enables instant updates of obstacle information in 3D grid maps for aerial vehicles at an order of magnitude lower computation time than current approaches to obstacle expansion. Incremental planning has been well studied, however in prior work it has mostly been assumed that planning is performed on a C-space expanded cost map and that calculating this cost map is not difficult. Indeed in 2D the number of cells affected to update the cost function based on a new obstacle is only a function of the square of the maximum expansion. For example for an expansion of 20 cells about 400 cells are affected. However, in 3D it is not trivial to update the cost function after new sensor data has been received because the number of cells that need to be recomputed changes cubicly. So for the same expansion we have to potentially look at roughly 8000 cells.

In this chapter, we present the Limited Incremental Distance Transform algorithm (LIDT) to efficiently perform this cost function update. In our approach, changes to the environment evidence grid [Martin and Moravec, 1996] are propagated via the described limited incremental distance transform. The list of changed costs is then used by an incremental planning algorithm to update the current plan.

We begin by describing in Section 4.1 how navigation cost functions are typically computed and their relationship to distance transforms. In Section 4.2.1 we discuss several potential approaches on computing such cost functions. Section 4.2.2 describes our novel algorithm and Section 4.3 presents experiments and results, including results from two autonomous aerial vehicles.

4.1 Problem

Incremental replanning using D* or its variants [Stentz, 1994, Koenig and Likhachev, 2002a] is a general and efficient approach to adapt to a partially-known or dynamic environment. However it is not always easy to determine how the map for planning changed on a 3D grid since it is necessary to plan in the changed configuration space [Choset et al., 2005] with changed costs.

Even though it is theoretically possible to plan with an arbitrary C-space expansion and cost function, the dominant factor is the distance to the closest obstacle. Therefore we assume that we can calculate the cost for planning around obstacles from distance.

Here, we assume a spherical robot, a reasonable assumption for a rotor-craft. A spherical expansion is easy to compute if we know the distance to the closest obstacle. If the distance is closer than the radius r_v of the UAV one is in contact with an obstacle. We set the cost for such an edge to be infinite cost.

We can express the general cost function between two vertices as follows:

$$c(k,l) = \begin{cases} \infty & if\ d(l)_o^2 < r_v^2 \\ \gamma \cdot obst(l) + dist(k,l) & otherwise \end{cases} \tag{4.1}$$

where $c(k,l)$ is the cost between position k and l, and the closest obstacle is $d(l)_o^2$. The cost consists of a scale factor γ that scales between the cost of obstacles $obst(l)$ and the cost of the distance $dist(k,l)$. Since we can express the C-space expansion in the cost function as an infinite cost, we will from here on refer to the cost function as the cost of an edge that also includes the C-space expansion.

The *dist* function can be any valid distance metric but one common metric is the squared Euclidean distance:

$$dist(k,l) = (k_x - l_x)^2 + (k_y - l_y)^2 + \alpha(k_z - l_z)^2 \tag{4.2}$$

where the x, y, z components are the displacement in the respective axis. If $\alpha = 1$ going left/right or to climb/sink is equal cost. If $\alpha > 1$ the robot will prefer to move laterally and if $\alpha < 1$ it will prefer to move vertically.

Several interesting cost functions for UAVs depend on the distance to the closest obstacle. For example, the shortest path with a clearance to obstacles:

$$obst(l) = max(0, d_{max}^2 - d(l)_o^2) \tag{4.3}$$

A maximum distance d_{max}^2 determines a cutoff beyond which the closest obstacle does not influence the path anymore. In the extreme case if d_{max}^2 and γ is large the path found will correspond to the solution of the Generalized Voronoi Graph (GVG) since the path will first lead away from the obstacle to get onto the Voronoi graph and then the lowest cost path will be on the graph and finally will go away from the graph to the goal point. In a natural outdoor environment the separation of obstacles is in many cases unbounded so that planning on that boundary would lead to too long paths.

Another useful cost function is to stay close to obstacles up to a desired distance d_{des} but not too close. This can be important for stealth reasons but also one might want to stay closer to obstacles to avoid wind or to stay localized using a limited range sensor. In this case the cost function can be expressed as follows:

$$obst(l) = \mid d_{des}^2 - min(d_{max}^2, d(l)_o^2) \mid \tag{4.4}$$

Note that for both $obst(l)$ functions it is necessary to know $d(l)_o^2$, the distance to the closest obstacle up-to the maximum distance d_{max}^2. Naively computing the distance $d(l)_o^2$ is expensive for large d_{max}^2 that are typically used in planning for UAV. The contribution of this paper is an efficient algorithm for calculating the changes to $d(l)_o^2$.

4.2 Approach

4.2.1 Distance Transform Algorithms

The distance $d(l)_o^2$ is the result that is computed by the distance transform algorithm. There are many possible distance metrics that can be applied, however the squared Euclidean distance is most useful for our application since we want the obstacle expansion and C-space expansion to be spherical. The property of the distance transform that we want can be expressed for a m x n x o grid with boolean obstacles $b[i, j, k]$ as follows:

$$EDT(x, y, z) = min(b[i, j, k] : (x - i)^2 + (y - j)^2 + (z - k)^2) \tag{4.5}$$

This property says that for every coordinate in the distance transform $EDT(x, y, z)$ we determine the minimum of the distance to all the obstacles $b[i, j, k]$. This would of course not be a very efficient algorithm in most cases however it shows what we need to compute. The Manhattan distance L_1 transform can be written like this:

$$MDT(x, y, z) = min(b[i, j, k] :\mid x - i \mid + \mid y - j \mid + \mid z - k \mid) \tag{4.6}$$

An efficient non-incremental linear time algorithm to calculate the distance transform was proposed by Meijster et al. [Meijster et al., 2000]. Even though this algorithm is very efficient, we will show in section 4.3 that repeatedly recomputing the result takes too long to be useful for navigation on a large grid. The algorithm scans the grid in three phases. In each phase the grid is scanned along a different axis forward and backward to determine a minimum. Overall the work performed is six passes through the grid for three dimensions. In two dimensions four passes are necessary.

A simple incremental approach to update the cost function in a grid is to update the grid with a mask of the distances to the obstacles. We will refer to this algorithm as "mask algorithm" in the algorithm evaluation. Every time an obstacle is added a convolution of the surrounding area is

Algorithm 4.1 Limited Incremental Distance Transform Algorithm (Helper functions).

```
INITIALIZE()
1  O ← ∅
2  foreach cell s
3    dist_s ← d²_max
4    dist_s^new ← d²_max
5    dist_s^old ← d²_max
6    obst_s ← ∅

SETOBSTACLE(o)
1  if dist_o^new ≠ 0
2    dist_o^new ← 0
3    obst_o ← o
4    UPDATEVERTEX(o)

REMOVEOBSTACLE(o)
1  dist_o^new ← d²_max
2  obst_o ← ∅
3  if dist_o < d²_max
4    UPDATEVERTEX(o)

CALCULATEKEY(o)
1  return min(dist_o, dist_o^new)

UPDATEVERTEX(o)
1  key ← CALCULATEKEY(o)
2  if o ∈ O
3    UPDATE(O, o, key)
4  else
5    INSERT(O, o, key)

DISTANCE(n, s)
1  Squared Euclidean:
2  v ← pos_n − pos_obst_s
3  return v · v

DISTANCE(n, s)
1  Quasi Euclidean:
2  v ← pos_n − pos_s
3  return v · v + dist_s^new
```

performed to check if any of the distances is larger than the distance in the mask. In the case of obstacle removal all non-obstacle cells that are in the mask of the obstacle are set to infinity and a region of two times the size of the mask is scanned for all obstacles. The region inside the removed obstacle is checked for any obstacle and the closest distance is restored. This algorithm serves as an incremental algorithm that one could implement easily.

While the runtime of the Meijster et al. algorithm depends on the size of the grid and is therefore non-incremental, the runtime of the mask algorithm depends on the number of obstacles added and removed. The algorithm we propose also depends on the number of obstacles that changed however if only a small number of distances changes less work has to be performed by the LIDT algorithm.

Kalra et al. [Kalra et al., 2006] developed an incremental algorithm to reconstruct the Generalized Voronoi Diagram (GVD) in 2D. The GVD is based on a quasi-Euclidean distance transform of the obstacles. The algorithm is the basis of the algorithm presented in this paper, however we have modified the incremental GVD algorithm to make it suitable for C-space and cost function updates and fixed a bug in their description of the algorithm. The presented algorithm also adds another variable to keep track of the changes in the distance transform while it is being computed that can then be used in an incremental planning algorithm. Furthermore we have generalized the algorithm to be applicable for different distance metrics (such as the squared Euclidean distance metric) while the original algorithm only allowed a quasi-Euclidean expansion. Also one can control the maximum amount of computation per obstacle in the LIDT algorithm because one is only interested in an obstacle expansion up to the maximum distance d_{max}.

4.2.2 Limited Incremental Distance Transform Algorithm

The Limited Incremental Distance Transform algorithm provides an efficient solution to keep an updated distance transform for changes to the cost function in the environment.

The algorithm is an incremental version of the brushfire algorithm and, as with the original brushfire algorithm it propagates a wavefront of distances to update the distance for each cell to its closest obstacle. For a good explanation of the brushfire algorithm also see Choset et al. [Choset et al., 2005]. The open list O keeps track of the wavefront and contains the cells that need to be expanded. Initially if only obstacles are added, the values of cells are lowered from d_{max} to consistent (or correct) distance values in the same way that brushfire would proceed. Since the values are sorted by increasing distance the cells with the smallest distance get updated first. Finally, the wavefront that is moving outwards terminates if the distance has reached a value that is larger than any of the neighboring cells or if the grid boundary has been reached.

Algorithm 4.2 Limited Incremental Distance Transform Algorithm (Main functions).

```
LOWER(s)
1  foreach n ∈ Adj(s)
2   if dist_n^new > dist_s^new
3     d' ← DISTANCE(n, s)
4     if d' < dist_n^new
5       dist_n^new ← d'
6       obst_n ← obst_s
7       UPDATEVERTEX(n)

RAISE(s)
1  foreach n ∈ Adj(s)
2   WAVEOUT(n)
3  WAVEOUT(s)

WAVEOUT(n)
1  if n ≠ obst_n
2    dist_n^new ← d_max^2
3    if !VALID(obst_n)
4      obst_n = ∅
5    obst_n^old ← obst_n
6    foreach a ∈ Adj(n)
7     if VALID(obst_a)
8       d' ← DISTANCE(n, a)
9       if d' < dist_n^new
10        dist_n^new ← d'
11        obst_n ← obst_a
12   if obst_n ≠ obst_n^old
13     UPDATEVERTEX(n)

INCREMENTALDISTANCETRANSFORM(O)
1  C ← ∅
2  while O ≠ ∅
3   s ← POP(O)
4   if dist_s^new < dist_s
5     dist_s ← dist_s^new
6     LOWER(s)
7     if dist_s ≠ dist_s^old
8       INSERT(C, s)
9       dist_s^old = dist_s
10  else
11    dist_s ← d_max^2
12    RAISE(S)
13    if dist_s ≠ dist_s^new
14      updateVertex(s)
15  return C
```

If an obstacle is removed a similar sweep outward propagates the changes to cells whose previous distance values are based on the removed obstacle and updates the distance for those cells since they now have a too close distance value. The cost to each of these cells is then updated based on the closest valid obstacle. Once the removal wavefront terminates each cell that does not have a valid obstacle will be updated with a valid obstacle (up to d_{max}).

It is important to note that the size of the queue in the wavefront depends on the radius of expansion. The number of cells in the queue is dependent on the radius r of the wavefront and grows linearly with the radius $\mathcal{O}(r)$. However since we are calculating the expansion in 3D the number of cells on the surface of the sphere grows with the square of the radius $\mathcal{O}(r^2)$. Also the maximum radius that has to be expanded depends on the size of the Voronoi region that is affected. One worst case example is an empty grid with one obstacle that is removed. In that example first all the cells going outward have to be invalidated and then all cells have to be lowered correctly again. In the worst case it is necessary to look at the grid twice for every obstacle removed.

For our application we are interested in computing the distance transform only out to a maximum distance d_{max}. As such the incremental distance transform propagation can be terminated once this distance is reached. This can save a significant amount of computation if the Voronoi region that changes is large.

The algorithm pseudocode is split in two parts the helper functions are shown in Alg. 4.1 and the main functions are shown in Alg. 4.2.

In INITIALIZE all cell distances are set to d_{max}^2 and the obstacle pointer is emptied. As the environment changes obstacles are removed and added with SETOBSTACLE and REMOVEOBSTACLE. If an obstacle is added its distance is set to zero and the obstacle points to itself. Then the obstacle is added to the queue to be expanded. Similarly a removed obstacles distance is set to d_{max}^2 and it is added to the queue with the priority of the old distance it used to have.

Since the update to the grid should always be with increasing priority the key is calculated from the smaller of the two distance values in CALCULATEKEY and the heap is updated in UPDATEVERTEX with the new priority unless the element has an infinite priority.

Using our algorithm one can calculate a squared Euclidean distance in DISTANCE or a quasi-Euclidean distance that is the shortest distance on a 26-connected grid. It is possible to calculate the squared Euclidean distance because we always keep track of the location of the closest obstacle in $obst_s$. The $obst_s$ pointer tells us if a grid cell needs to be updated because it points to an obstacle. If that obstacle changes all cells pointing to that obstacle need to be updated.

The main work of updating the distance transform and keeping track of changed cells happens in INCREMENTALDISTANCETRANSFORM which returns a list of updated distances C. If we added or removed obstacles the open list O will not be empty and so we take the first element of the list and LOWER the node if it is over-consistent and RAISE it otherwise. Since all nodes have to be made LOWER eventually we can keep track of the changed distances in lines 7-9.

Figure 4.1: The virtual campus environment of Carnegie Mellon University, Pittsburgh, PA that is used in the simulation experiments. Buildings that were added to the digital elevation model are shown in white. A hemispherical 200m range 3D range sensor is simulated to update the environment map held by the robot.

LOWER updates the distance of each adjacent node and adds it to the queue if the distance changed. Also we update the associated obstacle if it changed.

RAISE on the other hand propagates out a removed obstacle in WAVEOUT and so we first set the distance to be infinity and try to get a new distance for an adjacent node. If the associated obstacle changed we put the item back on the queue.

The algorithm terminates when the open list is empty. At this point all the cells in the grid have consistent distance values and have a valid obstacle pointer if their distance is less than d_{max}. A list of changed cells is in C.

We assume that the open list O has three operations: INSERT inserts an element in the open list with a given priority key, UPDATE updates the key of an element already in the queue, and POP returns and removes the top element from the priority queue.

Even though one can implement the open list O as a binary heap there is a fast data structure that can be used in our application because the maximum distance is limited to d_{max} and we are operating on a grid with integer values of the keys. Since there is only a small range of key values we create a hash table with the distances as key values. On every update we keep track of the lowest distance element. If we pop an element we update the lowest distance if it changes. To insert we just add the element to the list at the appropriate key value. This data structure allows $\mathcal{O}(1)$ for INSERT, UPDATE, and POP.

4.3 Experiments

There are certain tradeoffs between using an incremental and non-incremental algorithm that need to be considered. In this section we examine parameters that influence the performance of the LIDT algorithm and compare it to the fastest non-incremental algorithm and a simple incremental algorithm we denote the “mask" approach (described in Section 4.2.1). A recent survey [Fabbri et al., 2008] showed that the algorithm developed by Meijster [Meijster et al., 2000] is fastest in almost all test cases over a variety of problems. We therefore also compare the incremental algorithms to a 3D implementation of this algorithm.

4.3.1 Simulation

To determine the effectiveness of the limited incremental distance transform algorithm for aerial vehicle planning we evaluated it for missions in a simulated environment and compared it with two

Figure 4.2: The autonomous quad-rotor aerial vehicle used for testing. Here, the vehicle is close to a typical pole and wire obstacle in the environment. The system is equipped with a computer, a GPS, an inertial measurement unit, and a ladar scanner to sense the environment. The size of the vehicle with rotors is less than one meter.

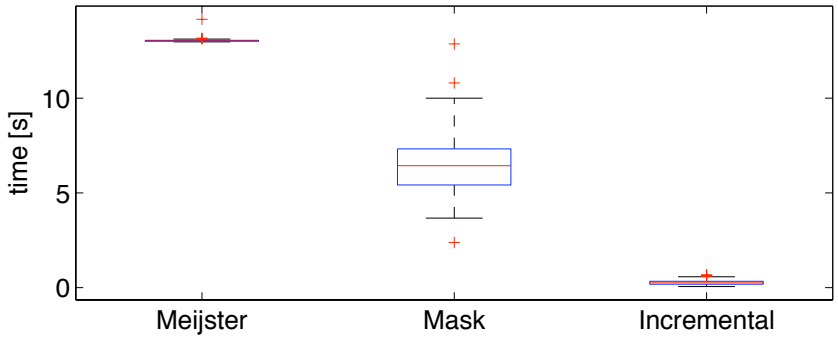


Figure 4.3: A comparison of running three distance transform algorithms in the environment shown in Fig. 4.1. The algorithms are run with the same sensor inputs on a grid that was initialized with an elevation model. *Meijster* is the non-incremental distance transform algorithm by Meijster et al. *Mask* is a simple incremental algorithm that updates based on a distance mask for each obstacle. *Incremental* is the limited incremental distance transform algorithm. The environment map changes as the robot discovers new obstacles and removes invalid obstacles from its initial map. $d_{max} = 20$. The mean of the number of obstacles added was 370 ± 32.3. The mean of the number of obstacles removed was 14 ± 6.5. The grid size considered is 512x512x80. Box and whisker plot legend: The red line is the median, the blue box extends from the lower quartile to the upper quartile, and the whiskers extend to 1.5 of the interquartile range. Red crosses represent single run outliers.

other distance transform algorithms: the non-incremental distance transform algorithm by Meijster et. al and the simple incremental 'mask' algorithm.

We want to simulate an algorithm load that is similar to a real extended mission of our micro aerial vehicle in a simulated environment of the campus at Carnegie Mellon University, Pittsburgh, PA, USA. See Fig. 4.1 for a screen shot from our simulation.

The simulation consists of a second order dynamic model of our quad-rotor helicopter shown in Fig. 4.2, a hemispherical 3D range sensor with a range of 200m, and a geometric model of campus.

A path tracking algorithm controls the helicopter and regulates speed based on the distance to the closest obstacle. As soon as a sensor measurement is received the evidence grid is updated and changes are given to one of the three distance transform algorithms. The changes to the grid and the cost function are then propagated to a D* Lite planning algorithm. The stimulus for the algorithms is sensor data that is received as packets of 3D range images covering a 180 x 180 degree field of view at 10 Hz. Each packet contains 961 points and the distance transform algorithms are updated after each packet. All algorithms run on a 2.5GHz Intel Core 2 Duo processor.

The size of the evidence grid is set at 512x512x80 with a resolution of 2 meters. We picked these dimensions because the size allows a typical plan length for micro aerial vehicle missions. Since there is a significant height variation at Carnegie Mellon it is necessary to have at least 160 meters of altitude change to enable climbing over obstacles as well as create plans that reach to the ground. The D* lite planner operates within the whole grid because we have some prior knowledge about the terrain as a digital elevation model that we update as we move. The cell size is set to two meters since the accuracy of our sensor data registration is about the same.

We ran an experiment in this environment with the robot avoiding obstacles that it saw within its range sensor and the three algorithms calculating the changes to the cost function for D* Lite. The maximum expansion for the grid was set to $d_{max} = 20$ and the environment map was initialized with the prior digital elevation model. During its traverse, a mean of 370 (standard deviation 32.3) obstacles were added and 14 (standard deviation 6.5) obstacles were removed.

Overall the limited incremental distance algorithm performed over an order of magnitude better than the competing approaches (Fig. 4.3 and Table 4.1), because the expansion distance is large and a number of obstacles have to be removed each iteration. As the number of obstacles removed decreases the 'mask' algorithm performs better because obstacle removal is an expensive operation for this algorithm. The non-incremental Meijster et al. algorithm has to traverse the grid several times and therefore cannot perform as well as the incremental algorithms which only have to update a small local region.

As the expansion distance decreases there is a point at which the mask algorithm becomes more efficient because it can use the processor cache better since it performs lots of sequential accesses. However if d_{max} increases significantly the runtime of the mask algorithm will increase beyond the non-incremental algorithm by Meijster because it must perform double work.

The runtime of the limited incremental distance transform depends on the size of the Voronoi region affected and in many cases if obstacles are close to each other then the affected regions are relatively small. In the case of the simulation and in realistic scenarios the changes to the map will be local and in a neighborhood of existing obstacles. In this case since only a small number of cells need to be updated the limited incremental distance transform has a significant advantage. One factor that was held constant in these experiments is that one range image is calculated at each time. However one could accumulate multiple measurements and if the same voronoi region is affected the cumulative update time would be reduced.

The standard deviation in Table 4.1 indicates that the computation time for the LIDT algorithm varies less than for the Mask algorithm. In a second experiment we evaluated the performance of the maximum distance d_{max} on computation time for our campus environment.

We ran a total of 37461 updates with varying maximum expansion values for the limited incremental distance transform. During testing we reset the environment map periodically and then recalculated the incremental distance transform with obstacles based on the digital elevation model which exemplifies the overhead cost of the incremental distance transform for starting from scratch.

The initial overhead of the incremental distance transform algorithm is significant if a large number of obstacles already exist in the environment map since the expansion has to perform more work per cell than the distance transform algorithm. A scatter plot of the overhead is shown in Fig. 4.4.

Algorithm	Incremental	Mean calc. time	Std. Dev.
Meijster	No	13.05s	0.16s
Mask	Yes	6.61s	2.21s
LIDT	Yes	0.27s	0.12s

Table 4.1: Calculation times of one update for the algorithm by Meijster et al., the mask algorithm and the limited incremental distance transform algorithm (LIDT). $d_{max} = 20$

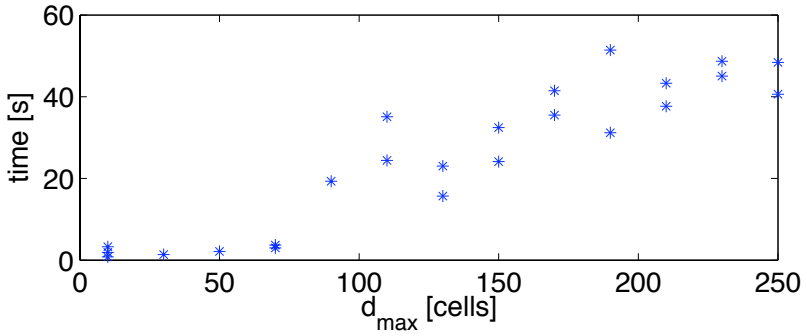


Figure 4.4: Initial calculation times for an empty 512x512x80 grid initialized with obstacles from a digital elevation model for different expansions. As the expansion increases so does the initialization time because a lot of cells need to be updated for a large expansion in the limited incremental distance transform.

After the initial overhead, however, subsequent updates are inexpensive. With an increase in the distance d_{max} the median computation time increases as well as the overall spread in computation. Since the potential number of cells affected by a change in the environment increases with d_{max} we also see a larger variation in computation times. Even with a large expansion of 250 cells, however, the limited incremental distance transform algorithm can still outperform the algorithm by Meijster. At such an expansion distance the 'mask' algorithm would not be feasible because on every update it essentially needs to check every cell in the grid.

4.3.2 Airrobot Quad-Rotor Vehicle

Our autonomous quad-rotor robot (Fig. 4.2) wants to avoid obstacles with a wide berth if possible because it increases the safety of the path and gives a better perspective for sensing. The robot is equipped with a regular GPS, IMU, and a ladar scanner to sense the environment. All computation is performed on-board and a planning cycle is performed at about 3Hz. It can fly missions of up to 20 minutes and can be given a series of waypoints it should reach. Since typically almost all of the waypoints are low the straight line path to the goal is typically obstructed by obstacles. The ladar scanner returns distances to obstacles and the information is processed into a global map as a 3D array in memory and a path is planned using a distance transform expansion. The position of the robot typically has an uncertainty (2m) that is incorporated as part of the C-space expansion by increasing the size of the vehicle. The robot can avoid obstacles in three dimensions since the planning algorithm also operates in three dimensions.

Fig. 4.6 shows the quad-rotor avoiding a tree in its straight line path to the goal. The cost function used for the planning algorithm in this case is the same as as in Eq. 4.3 with $d_{max} = 11$. Since that expansion is large and we are planning in 3D it is beneficial to use an incremental distance transform algorithm to update the changes to the cost function. The C-space expansion is set to 2 meters but since it is expensive to go close to obstacles the path gives the obstacle a wide berth. In this experiment the obstacle is avoided with a speed of 2m/s (Also see Extension 3).

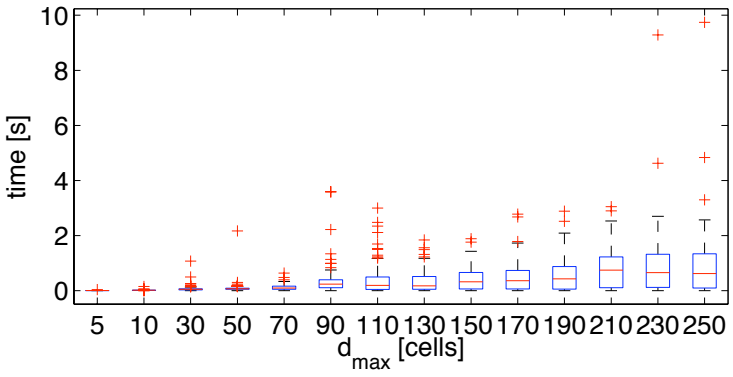


Figure 4.5: A box and whisker plot of the computation time in seconds for an increasing value of the maximum distance expanded d_{max} for the limited incremental distance transform. The computation time spreads out more as d_{max} increases since a changed obstacle cell can affect a larger Voronoi region however the region affected can also be small if the obstacle added is close to existing obstacles. The grid size considered is 512x512x80. For the legend of the box and whisker plot see Fig. 4.3.

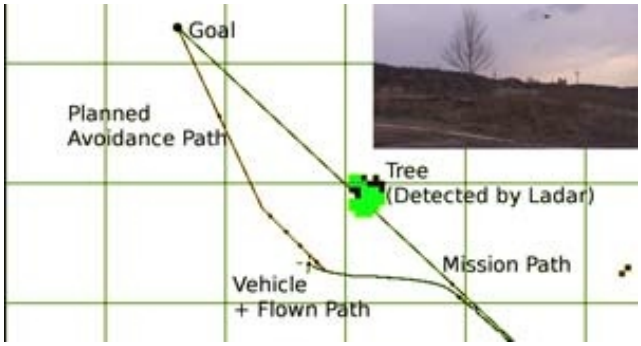


Figure 4.6: Avoiding obstacles with a large obstacle expansion on a quad-rotor. This plot and picture show our autonomous quad-rotor micro aerial vehicle (Fig. 4.2) avoiding a tree that is in the straight line mission path to its goal. As soon as the robot detects the obstacle it plans a path in 3D with a wide berth around the obstacle. The cost function is set up in such a way that the robot will prefer to move laterally and therefore we only show a top view. Since $d_{max} = 11$ the obstacle is avoided by a large margin. The mission path is shown in black and the planned avoidance path is shown in red. The black line shows the path of the vehicle as recorded by GPS. The tree obstacle is shown in green.

4.3.3 Autonomous Helicopter

Recently we tested the LIDT algorithm on an autonomous helicopter. See Section 5.3.2.1 for details on the setup off the Boeing Unmanned Little Bird helicopter. The helicopter has a rotor diameter of 8.3 meters and is manned with a pilot that can take control if necessary. The helicopter performed a completely autonomous mission to fly and come to a hover at a goal location. The straight line segment connecting the start and goal point intersected an obstacle. The helicopter had to avoid a man-lift obstacle as shown in Fig. 4.7 with a wide berth. The vehicle has to immediately update the limited incremental distance transform based on processing 83000 range measurements per second and compute a new path using the D* lite path planner. The planner creates a new trajectory and sends the new trajectory to the helicopter if it significantly changed since the last time a trajectory was sent. We performed five experiments at two speeds (10 knots and 21 knots) and initially at two altitudes 25 and 18 meters AGL. We initially tested at 10 knots and switched to 21 knots ground speed. Figure 4.8 shows the resulting paths. We believe this to be the first demonstration of a full-size autonomous helicopter avoiding obstacles. Extension 4 shows on of the obstacle avoidance runs.

The 1 meter evidence grid is updated constantly to reflect the new environment and after each update the distance transform is also updated to reflect the new distances to obstacles. The size of the grid was set at 1024x1024x512 and the maximum expansion was set at $d_{max} = 20$. The computation times for each of the updates to the incremental grid for five avoidance runs is shown in Fig. 4.9. The computation time for updating the evidence grid is half of the the ladar packet update interval (300ms). This ensures that we will not lag behind in processing and keeping the most current maps.

Most of the updates are performed between 10-40 ms as can be seen in the box-and-whisker plot in 4.9(a) and the computation time scales linearly with the number of changed cells as shown in Fig. 4.9(b). The times are not only on a line because the time measured is actual wall-time and not CPU time. For example, sometimes the algorithm has to wait to acquire a synchronization lock to update the data structure.

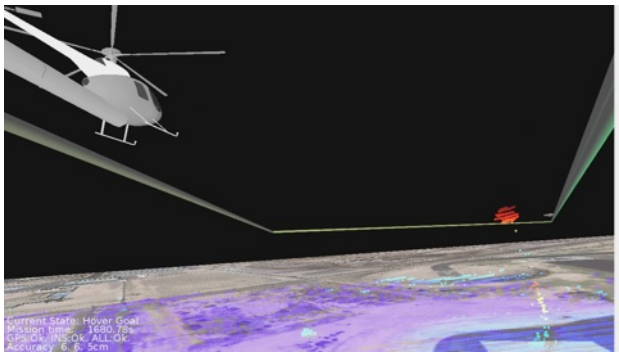
Overall the algorithm performed without problems and enabled the aircraft to avoid obstacles with a wide berth.

4.4 Discussion

We have presented a completely incremental framework for planning paths in 3D that enables recomputation of costs an order of magnitude faster than current approaches. This speed up is made possible by using a novel limited incremental distance transform algorithm. This algorithm exploits the local nature of cost function updates when obstacles are added or removed from the map and enables autonomous aerial vehicles to respond to newly observed obstacles (or obstacles that no longer exist) in real-time. We have provided results from simulation demonstrating the benefits of the approach and illustrative examples from a physical implementation on a quad-rotor micro aerial vehicle autonomously navigating in Pittsburgh, PA and an unmanned helicopter avoiding obstacles in Mesa, AZ.

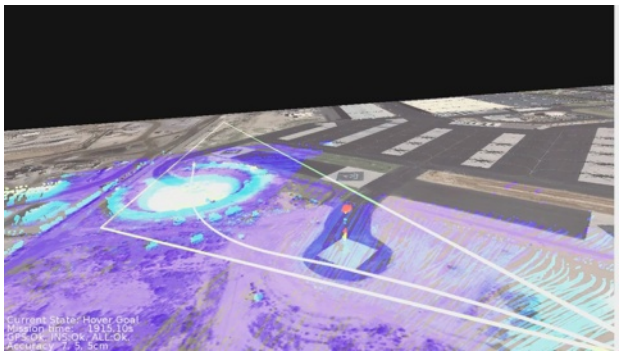
In future work we would like to further improve the computation time of the algorithm by reducing the number of expansions required to guarantee calculating a Euclidean distance transform. One issue that become apparent during testing on the Unmanned Little Bird was that the current algorithm does not permit shifting the grid to a new center. Therefore for every new segment the obstacle expansion had to be reinitialized which takes a significant amount of calculation time. In the future shifting grids can be achieved by making the grid scroll based on a the current position.

(a)



(b)

(c)



(d)

Figure 4.7: Obstacle Avoidance on the Boeing Unmanned Little Bird at 21 knots. In Fig. (a) the vehicle has not yet detected the man-lift obstacle. (b) shows the point cloud that is used as input for the evidence grid mapping. (c) shows the vehicle as it avoids the obstacle. In (d) one can see the path the vehicle took to avoid the obstacle overlaid with the point cloud and aerial imagery.

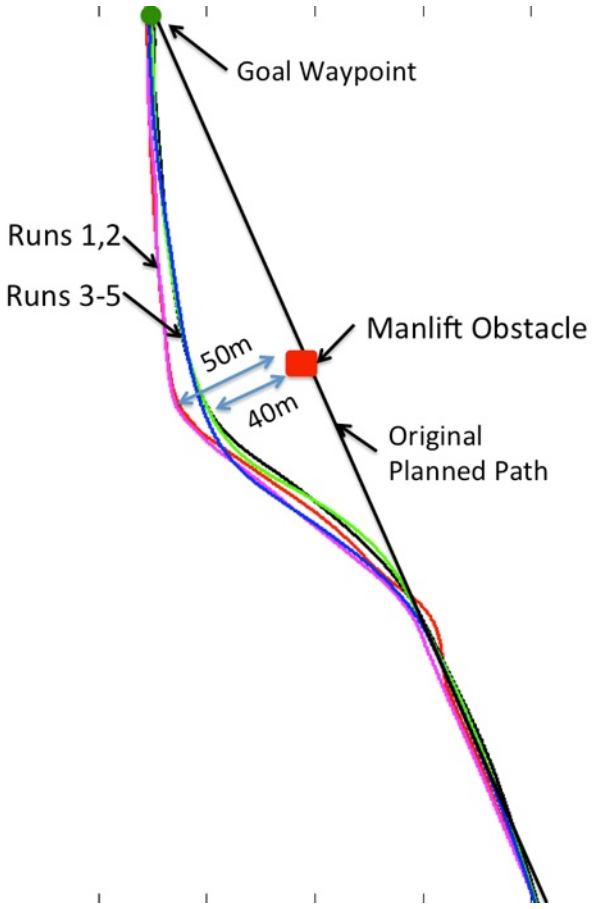
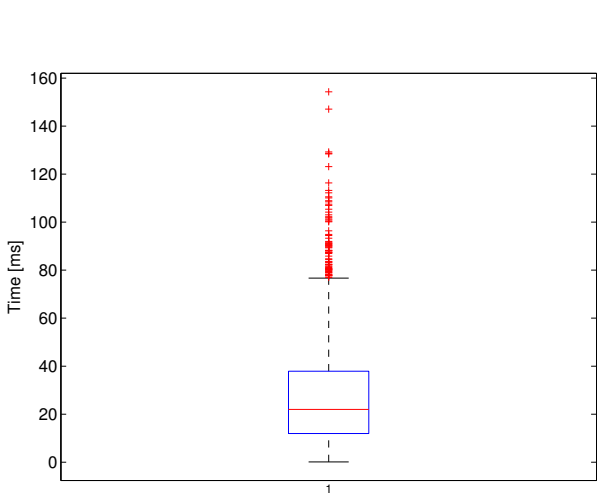
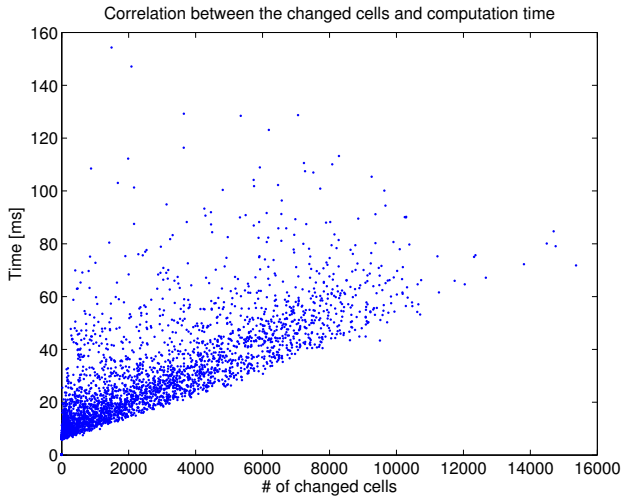


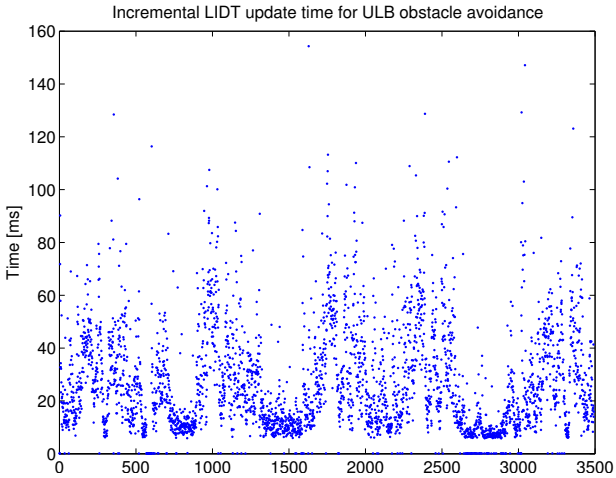
Figure 4.8: Top-down view of five obstacle avoidance runs against a manlift obstacle. Runs 1 and 2 were at 10 knots groundspeed, and runs 3-5 were at 21 knots. Since 3-5 are at a higher speed the vehicle reacts later to the obstacle.



(a) Box and whisker plot. Legend in Fig. 4.3.



(b)



(c)

Figure 4.9: Incremental computation times for the LIDT algorithm on the Unmanned Little Bird helicopter. The maximum obstacle expansion was set to $d_{max} = 20$. The results are cumulative for five obstacle avoidance experiments. The initialization time was 5.9s. A set of new ladar data is processed every 300ms and therefore the maximum update time is less than half of the data frequency.

5 Evaluating Landing Sites

Assessing a landing zone (LZ) reliably is essential for safe operation of vertical takeoff and landing (VTOL) aerial vehicles that land at unimproved locations. Currently an operator has to rely on visual assessment to make an approach decision; however, visual information from afar is insufficient to judge slope and detect small obstacles. Prior work has modeled LZ quality based on plane fitting, which only partly represents the interaction between vehicle and ground. Additionally, we want to be able to guide an unmanned rotorcraft to a safe landing location.

A NASA study by Harris et al. analyzed rotorcraft accidents from 1963-97 and found that 36.19% of them were related to collision with objects, roll over, and hard landings[Harris et al., 2000]. A partial cause for accidents is a lack of situational awareness of the pilot regarding the suitability of landing sites. Reliable detection of landing sites will increase the safety of operation by presenting vital information to the pilot or unmanned aerial vehicle before a landing approach.

Our approach consists of a coarse evaluation based on slope and roughness criteria, a fine evaluation for skid contact, and body clearance of a location. Additionally we consider the ground path to a goal location and approach paths to the landing site. Remaining potential locations of landing sites are represented by an information gain map that is continually updated. We investigated whether the evaluation is correct for using terrain maps collected from a helicopter and tested landing in unknown environments. This chapter defines the problem of evaluation, describes our incremental real-time algorithm and information gain map, and discusses the effectiveness of our approach.

In results from urban and natural environments from two different helicopter platforms, we were able to successfully classify and land at LZs from point cloud maps collected on a helicopter in real-time. The presented method enables detailed assessment of LZs without getting close, thereby improving safety. Still, the method assumes low-noise point cloud data. We intend to increase robustness to outliers while still detecting small obstacles in future work.

The main contributions of this chapter are

- an analysis of the problem of landing site evaluation.
- an incremental model-based method to calculate and evaluate landing sites for VTOL vehicles.
- an algorithm that creates a map of locations of potential future LZs.
- results based on ladar sensor data that show LZs found in real environments.
- results of the first helicopter that selects its own landing sites and lands.

This chapter first conveys the problem, our approach, and then presents results for the landing site evaluation and landing.

5.1 Problem

The suitability of a landing zone depends to a large degree on the characteristics of the aircraft that will be landing on it. We consider larger vehicles such as manned helicopters, however the problem scales to smaller aircraft. Landing of a helicopter can roughly be separated into two independent phases: an approach, and a ground contact. During the approach, the helicopter ideally keeps a steady and relatively shallow descent angle in a straight line to come to a hover at some altitude above the ground. Then it orients with respect to the ideal location on the ground and touches down. In the particular problem we are considering a ground goal has to be reachable from the LZ so that a human or robot could traverse the last segment.

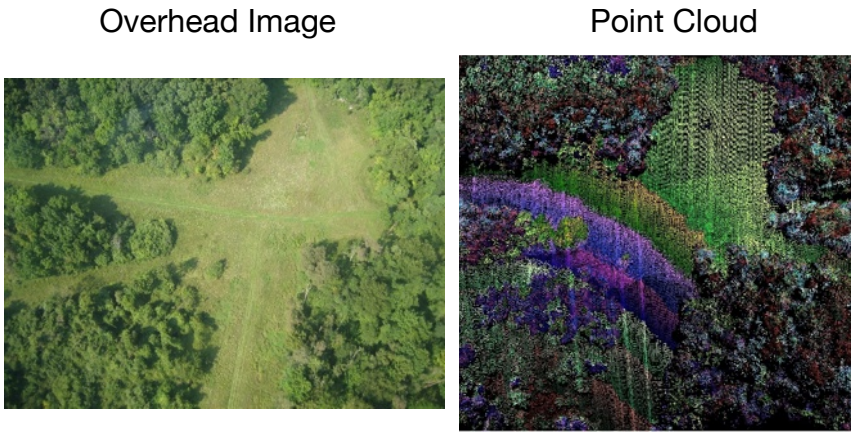


Figure 5.1: The problem of finding good landing zones for rotorcraft. On the left is an overhead image for reference. The input to the algorithm is an incrementally updated point cloud that is evaluated in real-time to find suitable landing zones. An example is shown on the right.

Assessment of a LZ needs to include two main factors: the ground conditions and the approach conditions. The ground conditions are all factors that are relevant when the aircraft is contact with the ground, while the approach conditions are related to moving to the LZ.

The ground conditions that need to be considered for assessing an LZ are

- minimum size of the site
- skid contact
- static stability on ground based on the center of gravity of the aircraft
- load bearing capability of the contact surface
- loose foreign objects (FOD) and vegetation surrounding the site
- clearance of aircraft/rotor with surrounding terrain and objects

while the approach conditions are the

- clearance of the path with respect to the terrain
- wind direction
- direction of the abort paths

The ground path needs to be evaluated for traversability cost and path length.

To ensure a reliable and successful landing it is necessary to consider all these factors in a landing site evaluation algorithm. However some factors such as the load bearing capability, and foreign objects are difficult to classify purely based on the geometric approach that we use. In our approach we assume that a geometrically good site will be able to hold the load of the helicopter and FOD objects will be classified as sufficient roughness to cause a rejection. Additionally, we want the algorithm to return a map of good places to search.

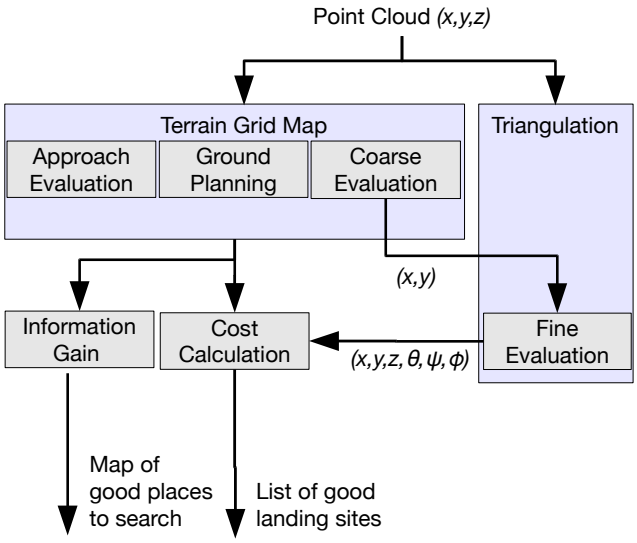


Figure 5.2: The steps in our landing site evaluation algorithm. The input is a list of globally registered points. The output is a sorted list of good landing sites that include location as well as landing heading and a map of good places to search. Two representations of the environment are created. One representation is based on a terrain grid map and the other is based on a triangulation of the environment and is compared to a 3D model of the helicopter.

5.2 Approach

The input to our algorithm is a set of 3D points $\{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^3$ that are registered in a global coordinate system. These points could be generated by a ladar range scanner and registered using an inertial navigation system with GPS, for example. Additionally the current wind direction θ_{wind} optionally is useful in choosing an approach into the wind. This avoids crosswind and reduces the required groundspeed. Figure 5.2 shows the data flow in our approach where the raw measurements are then processed to create a regular grid for the coarse evaluation of potential landing areas, and a separate triangulation used for the fine evaluation. Any updated grid cell is reevaluated to reflect changes in the available landing sites. A list of the available landing sites could be displayed to an operator or could be transmitted to the guidance system of an autonomous vehicle. A touchdown point is defined by a 3D position, an orientation, an approach and abort path direction, and a ground path. Additionally we calculated a map of good places to search based on the sensor data already collected.

5.2.1 Coarse Evaluation

The first step in our algorithm is a coarse evaluation of a regular terrain grid map. Each cell in the map, roughly the size of the landing gear, tests how the vehicle would come into contact with the ground. The map keeps a set of statistics in a two dimensional hash table of cells. The hash key is calculated based on the northing-easting coordinate at a fixed resolution. This provides a naturally scrolling implementation and cells can be set as data are measured. As the vehicles moves and the map scrolls, old cells will be overwritten. This data structure allows the algorithm to quickly add new data to the map. Each cell in the grid keeps a set of statistics: The mean, minimum, and maximum z-value, the number of points, and the moments necessary to calculate the plane fit.

The statistics accumulated in each terrain cell are evaluated to assess a particular location. Based on these tests a series of binary attributes and goodness values can be computed. The statistics can be accumulated with neighboring cells which allows us to compute the suitability of larger regions. The binary tests are designed to reject places where the ground is too rough or too step to be valid;

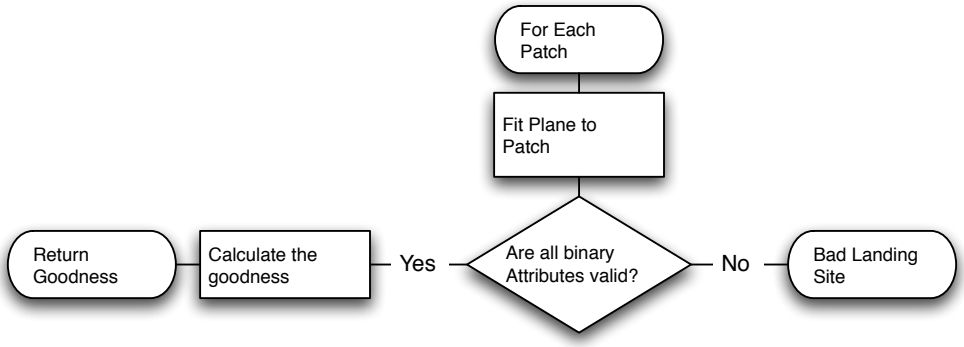


Figure 5.3: A flow chart showing the control flow of evaluating a patch in the terrain grid map. Only if all binary attributes shown in table 5.1 are valid a landing zone will be considered.

#	Description	Symbol	Operator	Value
1	Do we have more than x points for a plane fit?	c	$>$	15
2	Is the spread of the points less than x cm?	σ_n	$<$	50
3	Was the plane fit successful?		$=$	True
4	Is the residual for the plane fit less than x?	r	$<$	4
5	Is the slope of the plane less than x degrees?	α	$<$	5

Table 5.1: The binary attributes considered in the coarse evaluation for landing site evaluation. Only if all operations evaluate to true a landing site can be available.

furthermore, the algorithm cannot make a reliable estimate of a landing site, if not enough data is available.

The binary attributes are computed as follows: First, we check if enough points are available to perform a plane fit. We require at least 15 points per cell, which is an average density of one point every 20 cm for a 3 m cell. This average density is the required density for the smallest obstacle we consider; however, we make an optimistic assumption since points are not necessarily distributed evenly. Next, we consider the standard deviation of the z-value. The algorithm rejects cells if the vertical standard deviation is larger than 50 cm, rejecting any area with large altitude changes. The value has to be large enough to avoid interfering with the results of the plane fitting since a large change in altitude also characterizes a large slope. Finally, the slope and residual of a plane fit are checked for a planar surface. The residual will be a good indicator for the roughness of the terrain since it measures deviation from the plane. Currently the slope threshold is set at 5 degrees and the residual threshold is set at 4.

The mean altitude and standard deviation of the plane is calculated based on Welford's algorithm by keeping the mean and the corrected sum of squares [Welford, 1962]. After receiving a new altitude measurement in a patch the mean altitude μ_c and sum of squares S_c are updated:

$$\mu_{new} = \mu_{c-1} + (z - \mu_{c-1})/n \tag{5.1}$$

$$S_c = S_{c-1} + (z - \mu_{c-1}) \cdot (z - \mu_{new}) \tag{5.2}$$
$$\mu_c = \mu_{new}$$

The standard deviation σ can be calculated from S as follows:

$$\sigma_c = \sqrt{\frac{S_c}{c}} \tag{5.3}$$

Additionally we want to evaluate the slope and need to fit a plane with respect to the measurements. A fast algorithm to calculate a least squares plane fit is a moment based calculation. The fit is a least squares fit along z and therefore assumes no errors in x and y exist (Also see Templeton et al.[Templeton, 2007]). We only have to keep track of the moments

$$M = \begin{pmatrix} M_{xx} & M_{xy} & M_{xz} \\ M_{xy} & M_{yy} & M_{yz} \\ M_{xz} & M_{yz} & M_{zz} \end{pmatrix} \tag{5.4}$$

and the offset vector

$$M_o = \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} \tag{5.5}$$

and the number of total points c. The moments can be updated for a new measurement (x, y, z) by updating the moments starting from 0 for all initial values.

$$M_x = M_x + x,\ M_y = M_y + y,\ M_z = M_z + z \tag{5.6}$$

$$M_{xx} = M_{xx} + x^2,\ M_{yy} = M_{yy} + y^2,\ M_{zz} = M_{zz} + z^2 \tag{5.7}$$

$$M_{xy} = M_{xy} + x \cdot y,\ M_{xz} = M_{xz} + x \cdot z,\ M_{yz} = M_{yz} + y \cdot z \tag{5.8}$$

$$c = c + 1 \tag{5.9}$$

Consequently, an update is the constant time required to update the 10 values and the plane fit is calculated in closed form. Another advantage is that we can accumulate the moments from the neighboring cells to fit a larger region in the terrain that represents the whole area of the helicopter, not just the landing skids. See Ahn [Joon Ahn, 2004] for more details on moment based fitting. The plane normal $n = \begin{pmatrix} A \\ B \\ C \end{pmatrix}$ and residual r are calculate as such:

$$M_n = M_{xy}^2 c - 2 \cdot M_x M_{xy} M_y, + M_x^2 M_{yy} + (M_y^2 - c M_{yy}) M_{xx} \tag{5.10}$$

$$A = (M_y^2 M_{xz} - (M_x M_{yz} + M_{xy} M_z) M_y + c(-M_{xz} M_{yy} + M_{xy} M_{yz}) + X_x M_{YY} M_z)/M_n \tag{5.11}$$

$$B = (-(M_{xy} M_z + M_{xz} M_y) M_x + c M_{xy} M_{xz} + M_x^2 M_{yz} + M_{xx}(-c M_{yz} + M_y M_z))/M_n \tag{5.12}$$

$$C = (-(M_x M_{yz} + M_{xz} M_y) M_{xy} + M_x M_{xz} M_{yy} + M_{xy}^2 M_z + M_{xx}(M_Y M_{yz} - M_{yy} M_z))/M_n \tag{5.13}$$

$$r = \sqrt{\left| C^2 c + 2ACM_x + A^2 M_{xx} + 2BCM_x + 2ABM_{xy} + B^2 M_{YY} - 2CM_z - 2AM_{xz} - 2BM_{yz} + M_{zz} \right| / c} \tag{5.14}$$

We can infer the slope and roughness based on the normal and residual. The slope can be calculated as follows

$$\alpha = \arccos(v_{up}^T n) \tag{5.15}$$

where $v_{up} = (0,\ 0,\ -1)$ is the up vector and n is the plane normal.

If all the binary attributes shown in table 5.1 pass, the algorithm calculates the goodness attributes used in the final evaluation. All potentially good landing sites are now evaluated with the fine

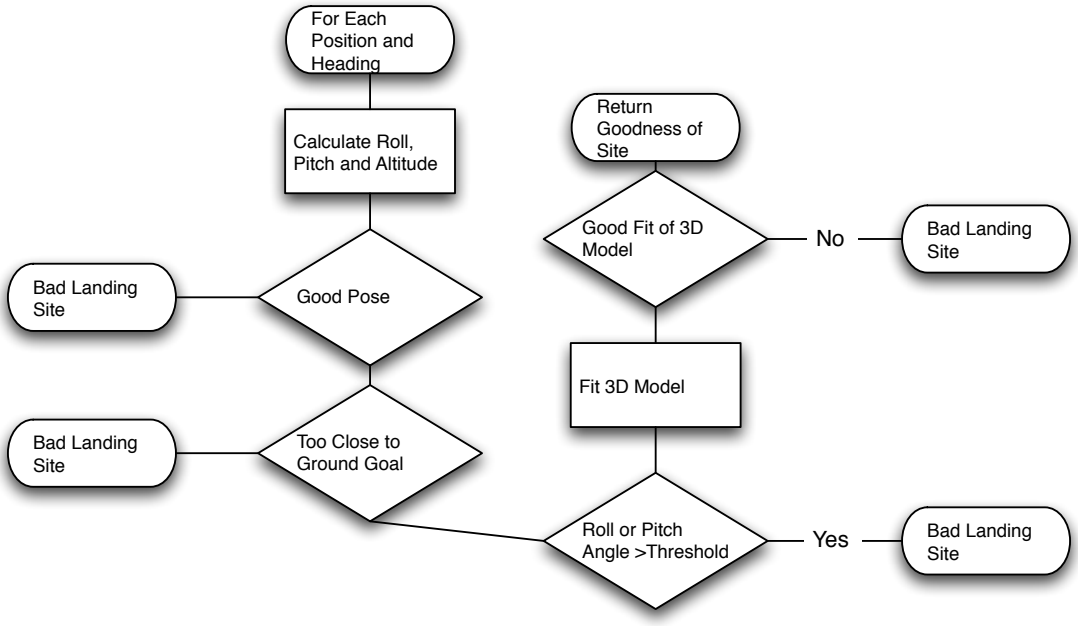


Figure 5.4: The flow chart of the fine evaluation algorithm. The pose of the helicopter is calculated based on a (x, y, θ) position and heading. If the pose is calculated successfully a surface model of the aerial vehicle is evaluated.

evaluation to decide if touching down is feasible.

5.2.2 Fine Evaluation

Once we have determined a list of general locations, we finely evaluate each potentially good site with respect to a 2D Delaunay triangulation of all available data. A Delaunay triangulation is a triangular network of points that can be constructed efficiently (See Preparate & Shamos for more details[Preparata and Ian Shamos, 1985]). The reason for this staged processing is that it is easier to know when to incrementally update a cell-based representation, and the incremental calculation for the coarse grid is much faster than for the triangulation.

While the coarse evaluation considers the plane geometry and roughness of a landing site, the triangulation tries to simulate the helicopter static contact with the surface. As a first step we incrementally update a hierarchical triangulation with a maximum number of points per patch area. For the implementation we use see Fabri et al. [Fabri et al., 1996]. If the point density is exceeded, we randomly remove points from the triangulation to keep below a maximum amount of memory used. Currently, the patch size is set to 3 meters and the maximum point density is set to 200 points/cell, which corresponds to an even point spacing of approximately 20cm.

Once the coarse evaluation calculates a list of potential (x, y) locations, the fine evaluation tries to find the best orientation and determines if the chosen location is feasible. We test a series of possible orientations θ of the helicopter (8 in our experiments) by intersecting the virtual landing gear with the measured terrain to determine what the resulting pose of the helicopter would be (Also see figure 5.4). Currently our algorithm only works for two landing skids, which is a very common configuration of landing gear.

First we determine the intersections of the skids with the triangulation by walking along the two lines representing the landing skids, and extract the height above ground as shown in figure 5.5. Now we assume that the helicopter's center of gravity is on top of the landing skids and calculate the two highest points of the intersection. These two points will be the points that determine the position of the skid in the static case. We repeat this process for all eight potential landing gear positions.

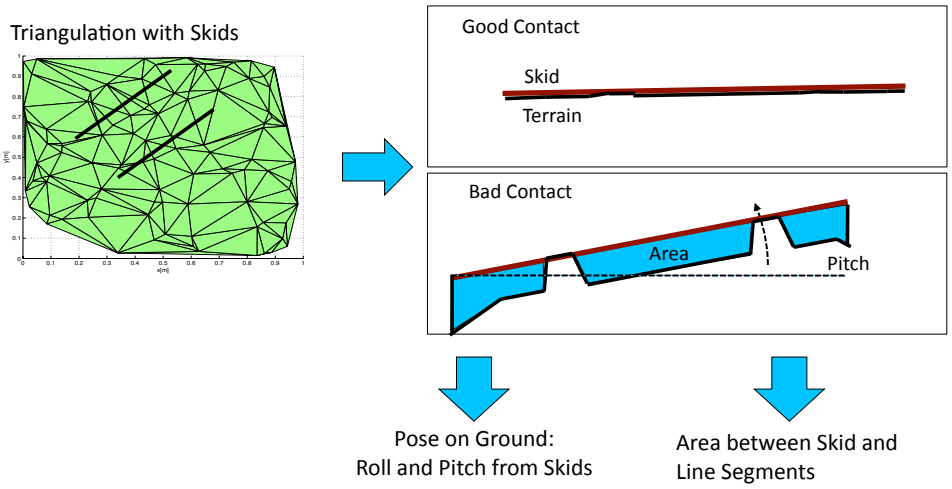


Figure 5.5: Illustration of skid - triangulation contact. The skids in contact with the triangulation determine the roll and pitch of the vehicle. A good contact is characterized by a small angle and small area between the skid and triangulation.

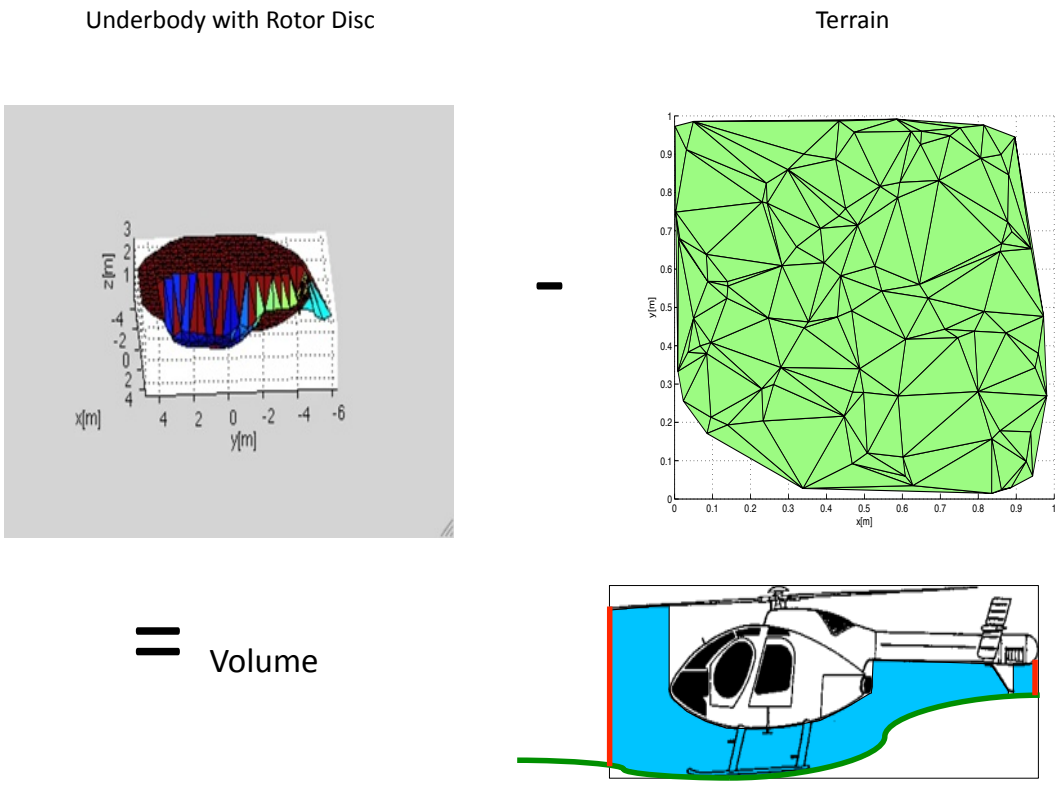


Figure 5.6: Underbody volume between model and triangulation. The underbody model is positioned with the pose calculated in Figure 5.5. The volume between the body surface (top-left) and triangulation (top-right) is used to check for collision and calculate a goodness measure of a landing site (bottom-right).

The algorithm to position the landing gear is as follows: First we determine all intersection points

$$p_i^l = (x_i^l, y_i^l, z_i^l) \tag{5.16}$$

where

$$p_i^l \in \mathbb{R}^3,\ i \in \{1, \ldots, n\} \tag{5.17}$$

between the landing gear l and the triangulation and partition the points into two sets: Points aft a_i^l and in front f_i^l of a distance along the landing gear. Currently we set the distance to be half the length of the landing gear because we assume that the center of mass is centered on top of the landing gear; however. this position can be changed depending on the mass distribution.

Next we look at the z-values and find the maximum z-value in the first and in the second half of the landing skid for each side.

$$p_{aft}^l = argmax(a_i^l) \tag{5.18}$$

$$p_{front}^l = argmax(f_i^l) \tag{5.19}$$

A minimum distance between p_{aft} and p_{front} is set to ensure that the slope will fit to a stable value. Now that we know how the landing gear will be positioned, we calculate the area between the line of the skid and the triangulation to get a measure of contact area. For example, if we were to land centered on railroad tracks at an 90 degree angle our skids would rest on the rails because they are the highest points above the ground. In this case even though the roll, pitch, and roughness will be small (rails are very small deviations from the ground and usually level) the integral of the area under the landing skids will have a large value because most of the landing skid is elevated above the ground. For skids in full contact with the ground the integral would be zero.

The next step is to determine the roll and pitch of the helicopter when it is in contact with the ground. We use the center of the two landing skid lines to determine the roll angle. Finally we use the lower landing gear line to calculate pitch because the center of gravity will shift towards that line. Using this algorithm we are now able to predict the position and orientation of the helicopter when it is in contact with the ground. The roll angle α is calculated from the interpolated centers p_c^l of the skids

$$p_c^l = \frac{p_n^l - p_1^l}{2} \tag{5.20}$$

$$\alpha = \arctan 2(z_c^2 - z_c^1, \sqrt{(x_c^2 - x_c^1)^2 + (y_c^2 - y_c^1)^2}) \tag{5.21}$$

The pitch of the helicopter depends on the roll angle since more weight will shift towards the lower skid. The pitch angle β is calculated as

$$\lambda = \begin{cases} 1 & if\ \theta_{roll} > 0 \\ 2 & otherwise \end{cases} \tag{5.22}$$

$$\beta = \arctan 2(z_{front}^\lambda - z_{aft}^\lambda, \sqrt{(x_{front}^\lambda - x_{aft}^\lambda)^2 + (y_{front}^\lambda - y_{aft}^\lambda)^2}) \tag{5.23}$$

With the orientation and position, we now calculate the volume between the undersurface of a 3D model of the helicopter with the triangulation of the terrain as shown in figure 5.6. This volume allows us to predict if a tail strike or similar bad contact with the ground would occur given the predicted position and orientation of the helicopter. It also gives us a measure of goodness that will maximize the volume under the helicopter. The calculation of the volume is simplified by taking a regular sample of points on the surface of the helicopter at an interval e. Even though some errors

can occur we can over-approximate this regular sampling when we create the model. Given the sampled surface as a set of points on the underbody of the helicopter p_s, we first have to transform the points given the pose of the helicopter p_c^h and the orientation (α, β, θ). The transformed points $p_s^{'}$ are then compared to the intersected point p_t of the triangulation. The volume is defined as

$$v_j = \begin{cases} e^2(z_s^{'} - z_t) & if\ z_s^{'} - z_t > 0 \\ -\infty & otherwise \end{cases} \tag{5.24}$$

$$V_{underbody} = \sum_{j=0}^{m} v_j \tag{5.25}$$

If the underbody surface intersects with the triangulation, the volume becomes $-\infty$ because we have a collision with the terrain.

5.2.3 Approach and Abort Path Evaluation

Once we have all the measures related to the goodness of landing sites we also need to verify that a landing approach is feasible. We therefore utilize the coarse terrain evaluation to check a linear landing approach path at the desired glide slope and final height above ground for clearance of obstacles. Additionally the algorithm also checks that an abort path that deviates by at most 45 degrees from the approach is obstacle free. This is an added measure of safety because we can abort an approach at any time by following the abort path from the current altitude.

The approach path is defined to start at some altitude a above ground with a glide slope κ and end at a final hover point. However we also want to ensure that there is some margin for the waypoint controller to converge onto the approach path and glide slope. Typically one can just expand the obstacles by a constant amount to keep a buffer. However since at the end of the glide path we want to be very close to the ground (1-3 meters) this approach will not work. Instead the glide slope is offset by a lower glide slope $\kappa_{C-space} < \kappa$. This lower glide slope provides a one-sided funnel for the path controller to converge on the correct glide slope.

From these parameters we can define the path as a line segment starting at p_{start} going to p_{hover}. Let the distance of a point $p_k$be given by the Euclidean distances. Additionally depending on the vehicle and control accuracy a lateral safety-buffer is defined as $b_{lateral}$ and a vertical safety-buffer is defined as $0 < b_{vertical} < a$. Now all cells on the approach path that are in the range of the vertical safety buffer $c_k \in \{1, \ldots, o\}$ are checked.

The simple volume as a goodness measure of an approach path places an equal cost independent of the distance clearance from obstacles. Therefore the cost of being close to an approach path decreases exponentially as the distance increases.

$$b'_{lateral} = \min(b_{lateral}, dist(p_{hover}, p_k) \sin(\kappa_{C-space})) \tag{5.26}$$

$$c_k = \begin{cases} \exp(-|z_{c_k} - z_k|) & if\ (z_{c_k} - z_k - b'_{lateral}) > 0 \\ \infty & otherwise \end{cases} \tag{5.27}$$

$$C_{approach} = \sum_{k=1}^{o} c_k \tag{5.28}$$

The lateral buffer is reduced once the helicopter comes close to the hover point because a collision with the terrain would be detected otherwise. The approach path is checked for a set of headings and the goodness is based on the risk of the approach and abort path.

5.2.4 Ground Path Planning

The final factor in landing site evaluation is the path length from the goal location to the landing site on the ground. This planning is performed assuming that a human or robot has to be able to

Description	Function	Weight
Goodness from Rough Evaluation	$w_s\alpha + w_r r$	ω_1
Area under the landing gears	$\frac{1}{A_{gear}}$	ω_2
Volume under the Helicopter	V_{heli}	ω_3
Distance from the closest bad landing site	d_l	ω_4
Distance to the ground goal point	d_g	ω_5
Cost of the approach and abort path	$C_{approach}$	ω_6
Cost of the ground path	C_{ground}	ω_7

Table 5.2: A linear combination of factors determines the goodness of a landing site. The weights needs to be set or learned based on user preference to determine which of the acceptable sites is the best.

reach the goal location from our landing location. This cost-to-go is another factor for our final cost function.

It is important that the traversability planner is robust to missing and some wrong cost estimates since the plan dictates if a site is reachable and will even be considered for landing. For instance, a skipped ladar packet at a pinch point will cause a hole in the terrain map making the any other landing site inaccessible.

The cost of traversability is therefore continuous with optimistic binary thresholds for non-traversability and optimistic estimates for missing data. The cost function is estimated from the standard deviation σ_c and the altitude difference between the mean of neighboring cells $\Delta\mu_c = \max_{n \in Neighbors(c)} |\mu_c - \mu_n|)$. The cost is calculated as follows

$$C_{ground} = \begin{cases} \max(\sigma_{scale}(\sigma_c - \sigma_{min}), & \Delta\mu_{scale}(\Delta\mu_c - \Delta\mu_{min}) \\ \infty & if\ Known\ Obstacle \end{cases} \tag{5.29}$$

The cost-to-go is then repeatedly computed using a dynamic-programming wavefront algorithm [Choset et al., 2005] from the ground goal location p_{goal} up to a maximum cost-to-go that we consider. In very rough terrain or with lots of missing cells only short paths will be generated while on smooth terrain such as roads long paths will be found which represents the time to traverse. The cumulative cost is then used as another factor in the goodness evaluation.

5.2.5 Goodness Assessment

Landing site selection is a problem where many factors need to be considered to evaluate the cost of a site. Depending on the weights for the goodness calculation, different sites will be chosen as best sites. A linear combination of the factors described in Table 5.2 is considered to calculate the goodness of a landing site. Currently we manually set the weights $\omega_1, \ldots, \omega_7$ to determine a goodness.

5.2.6 A 1D Example

Next we will show a simple 1D example of how the coarse and fine evaluation algorithm works in Figure 5.7. We start out with a set of 1D points that we want to evaluate. The points were generated from a 10 degree sloped line with added Gaussian noise. From this simulated terrain we can calculate the statistics and fit a plane. If the slope is sufficiently small the fine evaluation algorithm selects two points from the triangulation of the terrain. Since this is a 1D example it will be just the same line. Based on these two points we can calculate the pitch of the helicopter and fit the underbody.

Based on the result of the fit for the data set we have the following values

- Slope $\alpha = 9.6$,
- Spread $s = 17.2$
- Residual $r = 4.2$

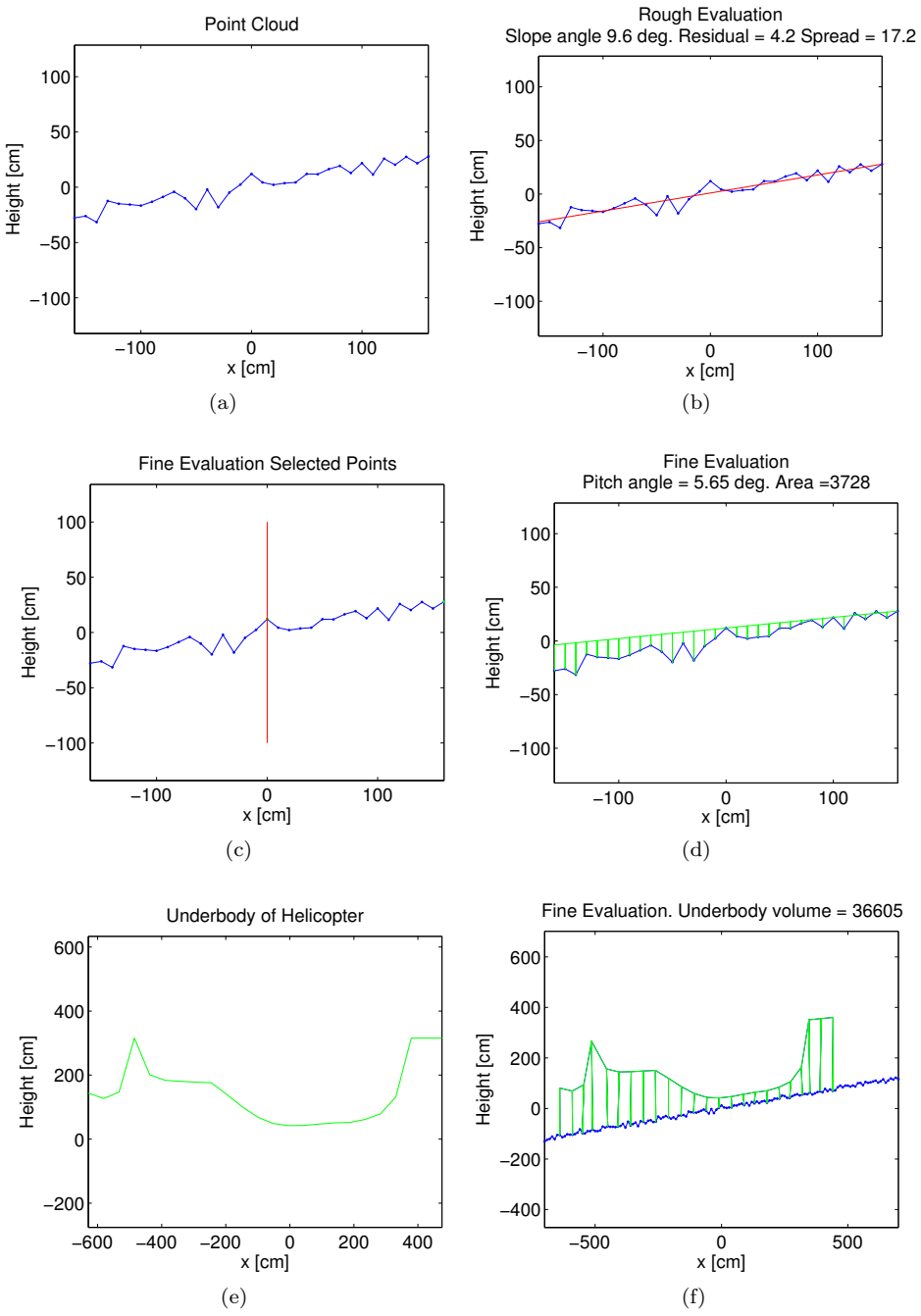


Figure 5.7: A 1D example of evaluating a landing zone. The input to the algorithm is a set of points (in (a)) that are then used in the coarse evaluation to fit a plane and calculate the point spread in (b). If the slope is sufficiently small and the spread is small enough the fine evaluation is performed in (c) where first the two highest points in the left and right half are found. These points determine how the landing gear will interact with the ground. In Fig. (d) we calculate the area and pitch angle. The final step is to take a set of sample points from the underbody of the helicopter in (e) and calculate the volume of the underbody with respect to the triangulation.

Factor	Term	Value
Basic Information Gain	I_l	
No data		0.5
Observed		0.05
Prior Knowledge		0.75
Time since last visit	$e^{-\frac{t}{\beta}}$	$\beta = 100$
Additional Factors:		
Distance from ground goal	$I_g = f(d_{max} - d(x, x_{goal}))$	$f = 0.022$
Unknown approach path cells	I_{app}	0.15
Unknown cells between ground goal and a potentially good landing site	I_{hole}	0.03
Exploration frontier for ground path	$I_{frontier}$	0.7

Table 5.3: The factors that are incorporated into the information gain map. All information gain terms are added to determine the final information gain map that is used for planning.

- Pitch $\beta = 5.65$
- Area $A_{gear} = 3728$
- Underbody volume $V_{heli} = 36605$

If we consider the binary attributes in table 5.1 we can assume that attributes 1,3, and 6 are successful. The point spread (2) is less than the threshold. However the slope (5) is larger than the threshold. So in this case we would not even consider the fine evaluation. If we look at the fine evaluation values we can see that we would also reject the landing site in the flow chart shown in figure 5.4because the calculated pitch value is larger than the threshold. The volume and area would not be considered given that we would reject the landing site.

5.2.7 Information Gain Map

The only possibility to get a current assessment of the terrain is to actively collect ladar data that we can use the build the map. However building a map is challenging, since we will have sensor occlusions with the terrain. Additionally, since we require a landing site to be reachable on the ground some locations are more important to search because they could enable new landing sites to become valid. Once a good site is known there is still some value to keep exploring, because we can discover more information about approach and landing paths.

According to each of the factors considered in the above cost function we construct an information gain map based on the factors shown in Table 5.3. Each of the sub-functions is a different term that adds to the information gain. The total information gain is expressed as

$$\mathsf{info}(\mathbf{P}) = I_l + I_g + I_{app} + I_{hole} + I_{frontier} \tag{5.30}$$

First we discourage visiting already observed sites if the time difference of the last visit to now is small (I_l). Next we also explore sites that are close to the ground goal first (I_g) and also try to fill in cells that are not visited and are between potentially good landing sites and the ground goal (I_{hole}). Also we encourage exploring cells that are on the frontier between unknown cells and cells with a path to the ground goal ($I_{frontier}$). Finally we also value approach paths higher to encourage exploration along approach paths (I_{app}).

The information gain map is qualitatively based on how the sensor observes and how the landing site evaluation algorithm behaves. Even though we get good results with this approach, in future work we would like to improve the map by throughly modeling the mutual information of the different factors and dependencies to calculate the information gain map. A proper modeling of the map will also provide us with the expected information gain depending on speed, and altitude. We smooth

the initial information gain map with a Gaussian to account for uncertainty in the sweep width of the ladar scanner and the lack of modeling the orientation. The information gain map is updated as we get new information and is one of the objective functions considered for motion planning in Chapter 6.

Additionally to representing the current state of information we need a way to predict the outcome of sensing to enable long range planning. We can define a prediction action $action_{info}(\mathbf{P})$ that takes as input the path $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$. The prediction assumes that everything was sensed and no ground path was found. The area updated is a line along the current path direction. The sweep width is determined based on the field of view $\alpha_{scanner}$ of the scanner and the current altitude p_z above ground:

$$l_{sweep} = 2\tan(\frac{\alpha_{scanner}}{2})(p_z - z_{dem}(\mathbf{p})) \tag{5.31}$$

In order to evaluate the effectiveness of the information gain map in guiding the aerial vehicle we tested the information gain mapping in simulation. Our simulation provides similar inputs (ladar and pose measurements) that one would expect on a real vehicle and therefore it is used to demonstrate the information gain mapping algorithm. As more information is sensed about the terrain the information gain map updates to reflect the changes in sensed information. Since sensor data is used to estimate the available information gain the algorithm is able to include the occlusion information and other factors in the evaluation.

In Fig. 5.8 we show one example of the helicopter using the information gain map to explore the environment. As it explores the environment more areas become known. As soon as a path from the location on the ground becomes available in Fig. 5.8c the cells on the frontier become valuable. In Fig. 5.8d the approach path to the landing site is an valuable area to explore.

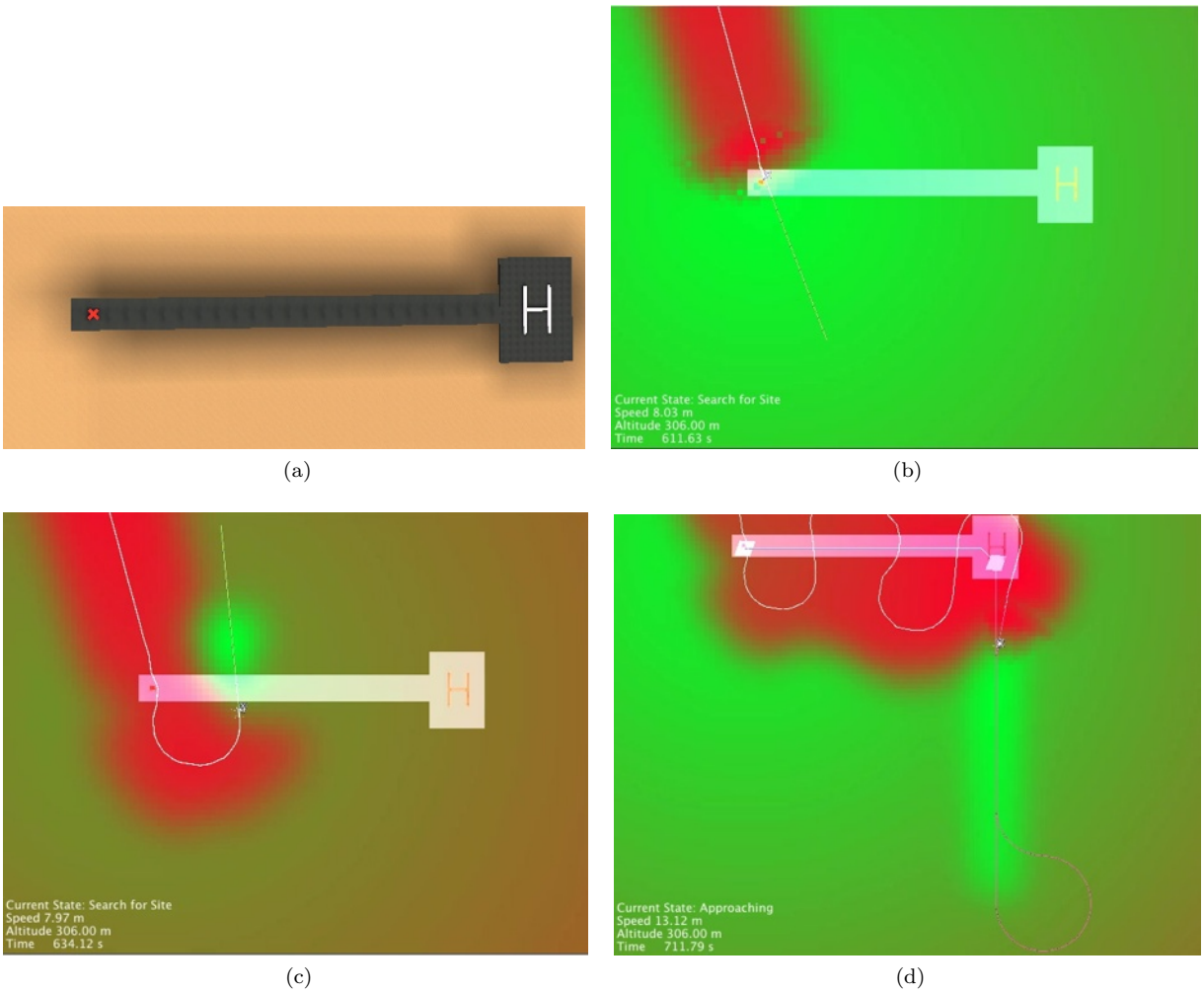
Our information gain mapping is based on the terrain evaluation and represents an estimate of the available information left in the world. The function is non-trivial because the landing approach can add valleys of desirable information that need to be explored and many other factors contribute to the final value.

5.2.8 Computational Efficiency

A naive online implementation of the algorithms above will be infeasible to evaluate in real-time. We parallelize and incrementally compute updates to enable a real-time implementation. The following techniques enable real-time computation on a multi-core processor:

- All trivial parallel evaluations are spread across multiple cores. This includes evaluating
  - the fine evaluation of a set of sites.
  - the detection if any landing approaches need to be recomputed.
  - a series of landing approach directions.
- A triangulation is only created if a fine evaluation is performed in that region.
- A fine evaluation is only performed if any of the rough terrain cells change.
- Fine evaluations are performed radially outward from the ground goal location p_{goal}. As computation time permits landing sites further and further away will be checked.
- An approach path is only evaluated if the terrain cells related to it have changed.
- Only a fixed number of approach and abort directions are evaluated.

Our current implementation will use as many cores as are available and can use those cores efficiently. Using this approach we are able to calculated landing sites in real-time at normal helicopter speeds.



(a) (b)

(c) (d)

Figure 5.8: An example information gain map in simulation for the environment shown in (a). The only available landing sites are at the "H" and the helicopter should land close to the red cross. The only path to the landing sites is the gray area between the red cross and the H. As the helicopter flies through the environment it accumulates more information. Available information is shown in green already gathered information is shown in red. In (c) we can see that the frontier of locations with a path to the ground location have a large amount of information to be gained. Since no more frontier exists the approach path in (d) has become a valuable location for exploration.

5.3 Experiments

We present two experimental results in this section. The first set of results evaluates only the coarse and fine landing site evaluation algorithm in a variety of different urban and vegetated environments. The second set of results demonstrates the autonomous landing of the unmanned little bird helicopter in an unknown environment that includes ground path planning, approach path generation, and calculating the information gain map.

5.3.1 Fine and Coarse Landing Zone Evaluation

In this section we show results for the coarse and fine evaluation based on 3D point-cloud datasets we collected. In these results we assume a straight line path from the landing site to a ground goal location. The results are shown for natural and man-made environments to illustrate several environments and varying cost weights.

5.3.1.1 Setup

For our first experiment we collected point cloud datasets at various locations around Pittsburgh, PA, USA. We overflew the areas in approximately straight lines with speeds between 10-20 knots. On the bottom of the helicopter we mounted a laser scanner, inertial navigation system, and a documentation camera (See figure 5.9). The Riegl LMS-Q140i-60 scans a line perpendicular to the flight direction at 8000 points per second. The scan frequency of the ladar is 60 Hz. The data are fused by a Novatel SPAN INS/GPS to calculate a globally registered point cloud. The inertial measurement unit has a drift rate of 10 degrees/hour (0.02 degrees reported accuracy) and the GPS is corrected with a base station to a reported 1 cm accuracy. In total we collected data on 9 sites with a flight time of 8 hours. We discuss results from three representative sites.

The coarse evaluation is robust to errors in z since it performs a least squares fit to determine the slope. The fine evaluation relies on a good point cloud registration and is not robust to misregistered points. Nevertheless, other methods exist to improve point cloud registration to get a better input point cloud from lower performance INS systems (See for Example [Thrun et al., 2003]).

5.3.1.2 Results

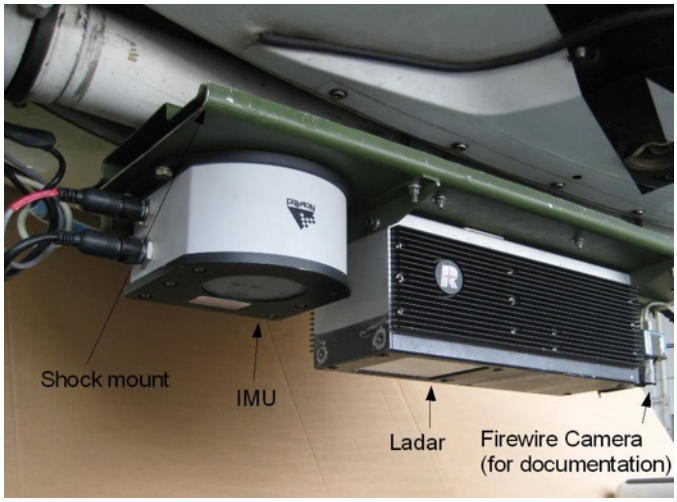
The first site was scanned at an altitude of about 100 meters (Fig. 5.10). It is a field, and our ground goal point is located next to the slope on the hill (see Fig. 5.10d for exact goal location). The result shows the coarse and fine evaluation of landing sites in the environment. Several possibly good landing sites in the coarse evaluation are rejected in the fine evaluation phase. Finally, the best landing site is picked in a spot with small slope and a large clearance from obstacles.

In the "sloped field" dataset (Fig. 5.11) we have a slightly more complex scenario because a lot of landing sites exist and we must choose a best one. Also a large number of smooth slopes need to be rejected based on the plane fitting. The next dataset (Fig. 5.12) is very tight and a possible landing site needs to be chosen that still fulfills all the constraints. We also scanned this dataset at an altitude of about 100 meters. Just based on the coarse evaluation one can see that most points are not suitable and only a small area exists that can be considered suitable for landing. In the fine evaluation some too optimistic locations from the coarse evaluation are rejected. The main reason for the rejection is that the landing gears cannot be placed successfully. After evaluating the cost a point in the middle of the open area that is closer to the ground goal location is picked.

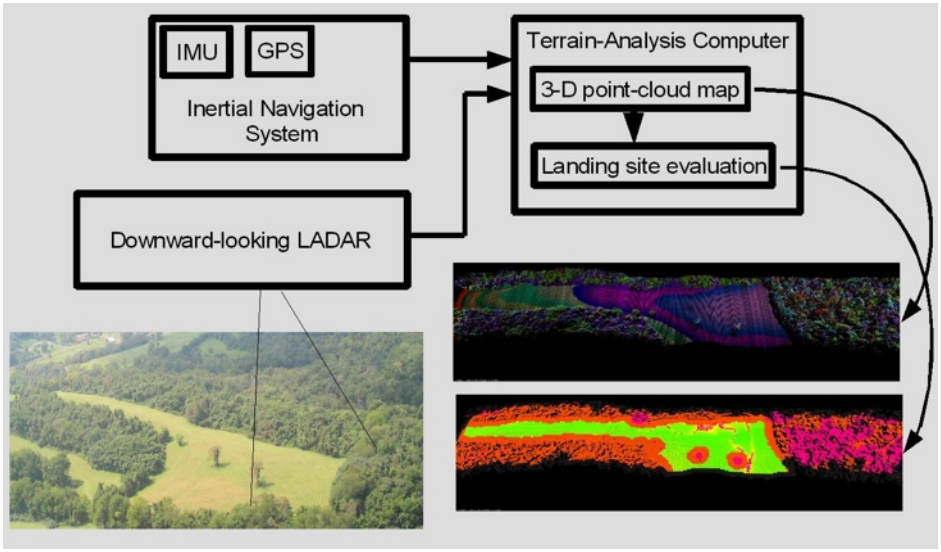
While the previous dataset was mostly in vegetated terrain we also collected a dataset at a coal power plant (Fig. 5.13) at 150m elevation above ground. The geometry of obstacles at the power plant is very different from vegetation, but the algorithm is still able to determine good landing sites. The ground goal location is close to a car, and one can see how the fine evaluation rejects the car and the red region around the ground goal. The red region is a safety buffer that prevents landing directly on top of the ground goal. Extension 5 shows the overflight and real-time evaluation of landing sites.

As a final experiment, we varied the weights for our evaluation function for four different parameters in the same terrain (Fig. 5.14). First we changed the weight for slope and roughness (Fig. 5.14a).

(a)



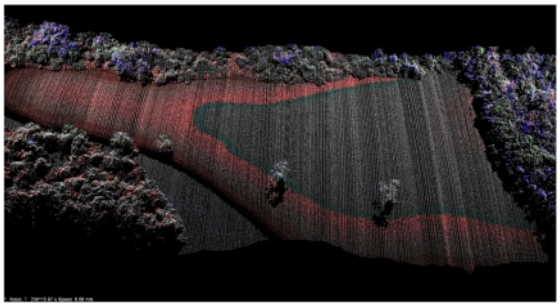
(b)



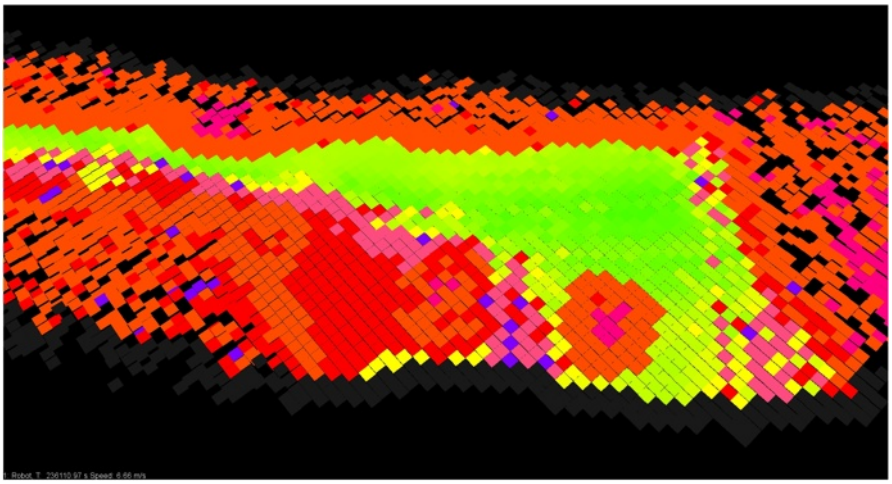
(c)

Figure 5.9: The experimental setup for data collection on a EC 135 helicopter in Pittsburgh, PA, USA. We mounted a ladar, and inertial navigation system on a helicopter as shown in Fig. (a). The sensor platform is shown in Fig. (b). The inertial measurement unit was mounted next to the ladar to minimize errors in the registration of the data. In Fig. (c) one can see the data flow for terrain evaluation. We register the data into a global 3D point cloud and evaluate data based on the point cloud.

(a) Picture



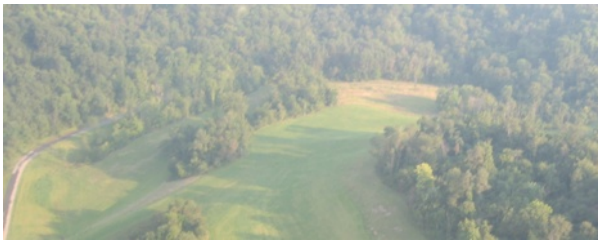
(b) Input point cloud



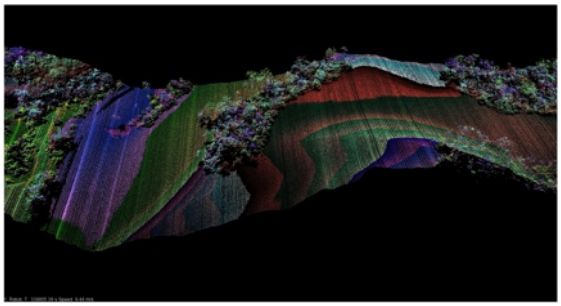
(c) Coarse evaluation

(d) Fine evaluation

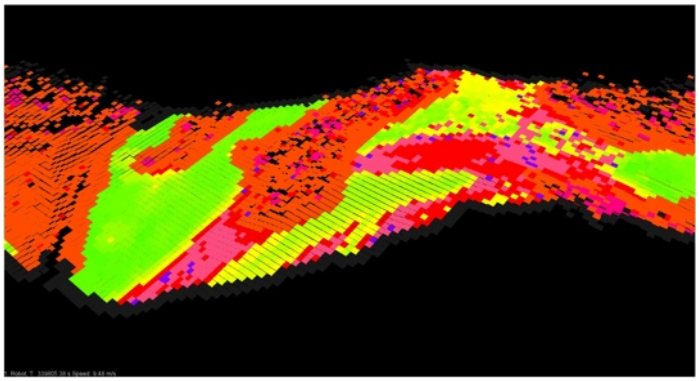
Figure 5.10: Results from the "three trees" area. Fig. (a) shows an aerial view of the scanned area and (b) shows the point cloud at the end of the scan. The legend for the coarse evaluation (c) is as follows: Red = slope & roughness exceeded, orange = roughness exceeded, pink = slope exceeded, light orange = large spread, purple = large residual, dark purple = bad fit. The legend for the fine evaluation (d) is as follows: Red = bad site, yellow to green = goodness of site, white circles = best site, red circle = no land zone, white box = goal location.



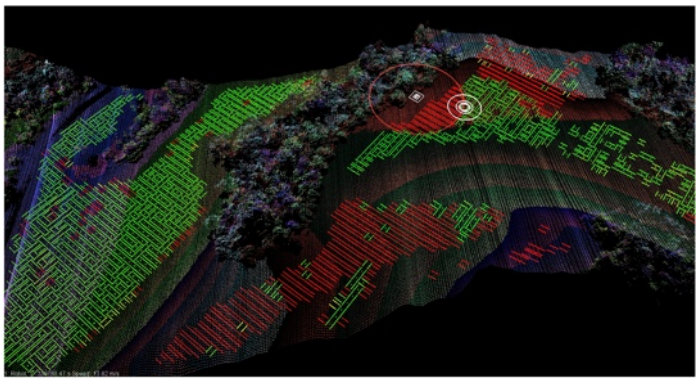
(a) Picture



(b) Input point cloud



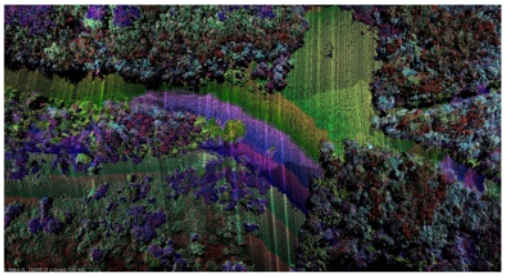
(c) Coarse evaluation



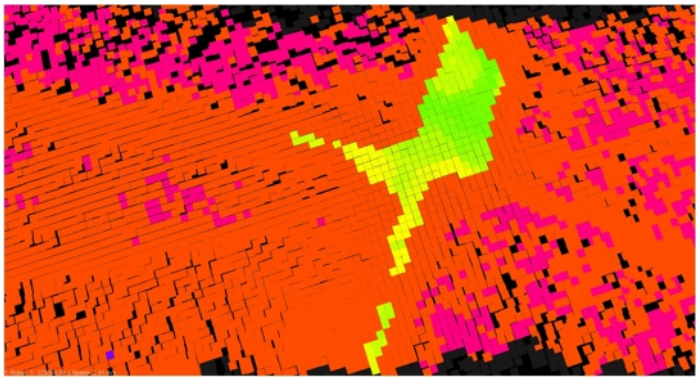
(d) Fine evaluation

Figure 5.11: Results from the “sloped field” area. For a legend of the results see Fig. 5.10.

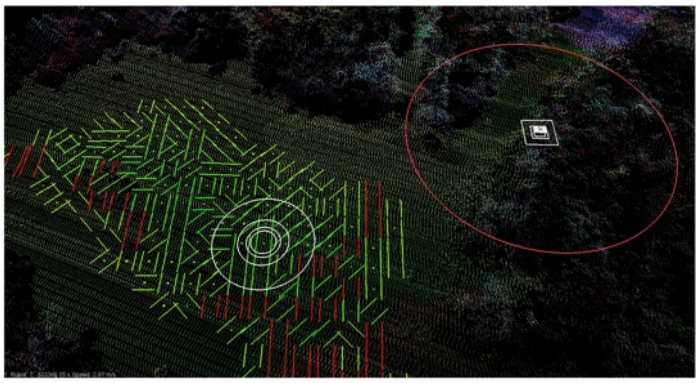
(a) Picture



(b) Input point cloud

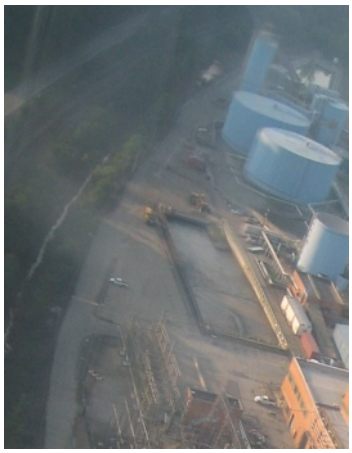


(c) Coarse evaluation

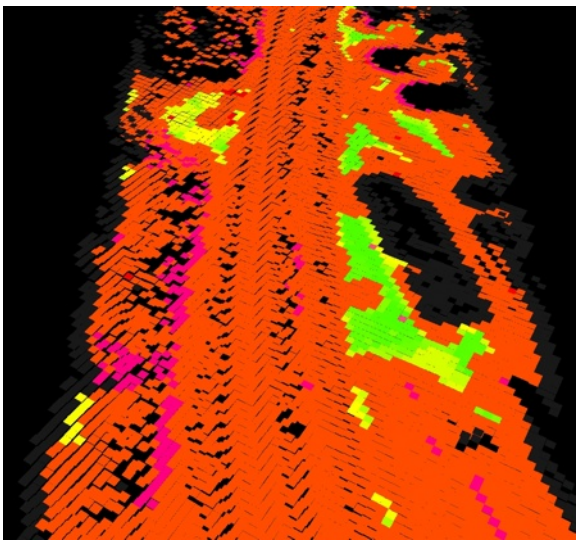


(d) Fine evaluation

Figure 5.12: Results from the “wooded intersection” area. For a legend of the results see Fig. 5.10.

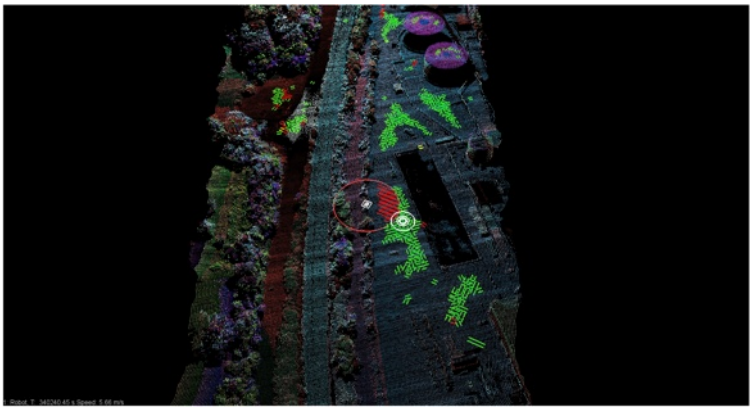


(a) Picture



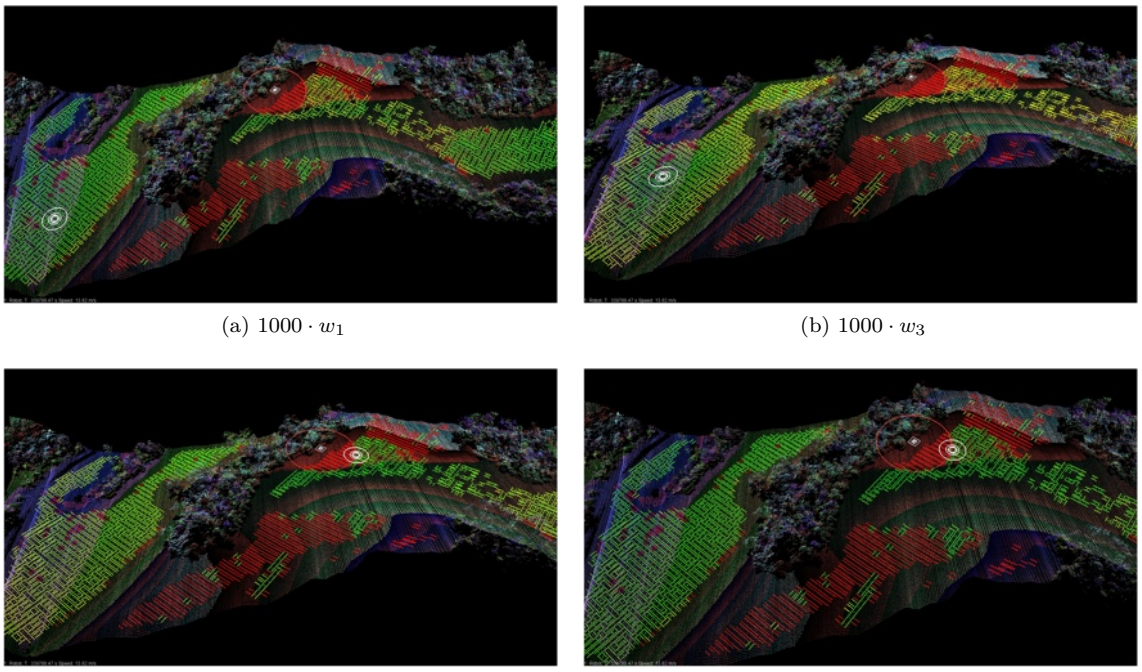
(b) Coarse evaluation

(c) Input point cloud



(d) Fine evaluation

Figure 5.13: Results from the "power plant" area. For a legend of the results see Fig. 5.10.



(a) $1000 \cdot w_1$ (b) $1000 \cdot w_3$

(c) $100 \cdot w_6$ (d) $\frac{1}{1000} \cdot w_4$

Figure 5.14: The result of varying the weights for the final decision on selecting the best landing site. For a legend of the results see Fig. 5.10. In this plot we keep the function constant and vary the weight for some of the factors described in Tab. 5.2. In (a) we increase the weight of $1000 \cdot w_1$ - the slope and roughness, for (b) we increase $1000 \cdot w_3$ to emphasize the clearance of the helicopter above the terrain, for (c) we try to minimize the distance to the casualty by increasing $100 \cdot w_6$, and in (d) we decrease the weight of the distance to the closet obstacles by changing $\frac{1}{1000} \cdot w_4$. One can see that depending on the cost function the distribution of good sites changes, as well as the best picked landing site. The weighting does not influence the binary decision of acceptable sites because it is based on hard thresholds of the helicopters constraints.

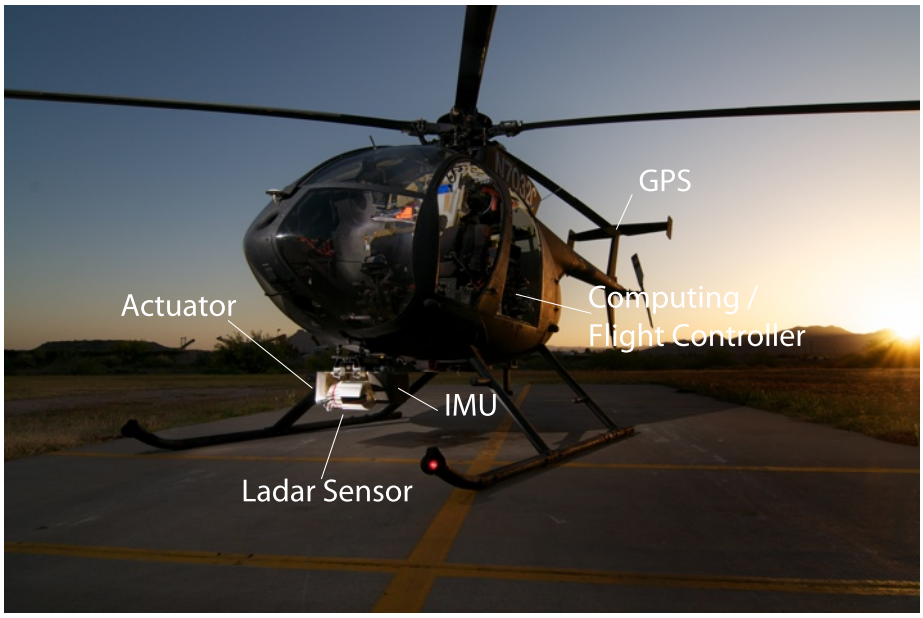


Figure 5.15: System setup on the Boeing Unmanned Little Bird Helicopter. The sensor and IMU are shock mounted to the helicopter in the front. The GPS antenna is located in the tail and the computing and flight controller are located in the cargo bay. The rotor diameter is 8.3 meters.

Compared to the nominal evaluation, one can see that we now choose a landing site further away from the ground goal point because it is in smoother and less sloped terrain. A similar location is chosen in Fig. 5.14b; however, since we try to emphasize the volume of the underbody, the shape of the LZ is actually more convex than the evaluation in Fig. 5.14a.

If we give a large weight to the distances to the casualty, we expect the distance to be small. We can see in the evaluation for Fig. 5.14c that far points have low weight and close points have the lowest weight. In Fig. 5.14d, the result is almost the same as Fig. 5.14c, except that now points close to the boundary also have a low value. Varying the weights, one notices that the weight vector has a large influence on the resulting chosen landing site.

5.3.2 Landing the Unmanned Little Bird Helicopter

In the previous section we presented landing site evaluation results in a large variety of terrain around Pittsburgh, PA. In this section we present results from Mesa, AZ. In this environment we tested the approach and ground path planning in addition to the coarse and fine evaluation. The ground path planning was performed to a ground goal location. Furthermore, the helicopter was completely autonomous and was able to land itself based on the evaluation.

5.3.2.1 Setup

We equipped the Boeing Unmanned Little Bird Helicopter [Boeing, 2010] with a custom scanning LADAR that operates in two modes: Forward-scanning for obstacle detection and downward-looking for landing zone evaluation. The scanner has a 100Hz, 100-degree fast horizontal scan and a controllable slower nodding scan at about 1 Hz. For landing zone evaluation it was pointed down and did not oscillate on the slow axis. The Riegl VQ180 pulses at about 85kHz with full-waveform analysis and has a range of 150 meters. The scanner is rigidly coupled to a Novatel SPAN INS/GPS to calculate a globally registered point cloud. The inertial measurement unit has a drift rate of 10 degrees/hour and the GPS is corrected with a base station to a reported 1 cm accuracy. The helicopter and main system components are shown in Fig. 5.15. The scanning Ladar is mounted

in the front with the IMU and the GPS antenna is located on the tail. The computing and flight control computers are located in the cargo bay. We discuss results from test flights at Boeing in Mesa, AZ (USA). The experiments were performed completely autonomously, however a test pilot and flight test engineer were onboard at all times to supervise and initiate the tests.

Our test site was located in a busy airspace and we had to fly certain paths to avoid populated areas. A fixed search pattern was set that would overfly the ground goal and an area with good landing sites.

5.3.2.2 Results

The following scenario was tested: An injured person (casualty) needs to be picked up and is located somewhere near a possible landing site as shown in Fig. 5.19(c). We want the vehicle to find a good landing site that is close to the ground location such that the ground goal can be reached from the landing site.

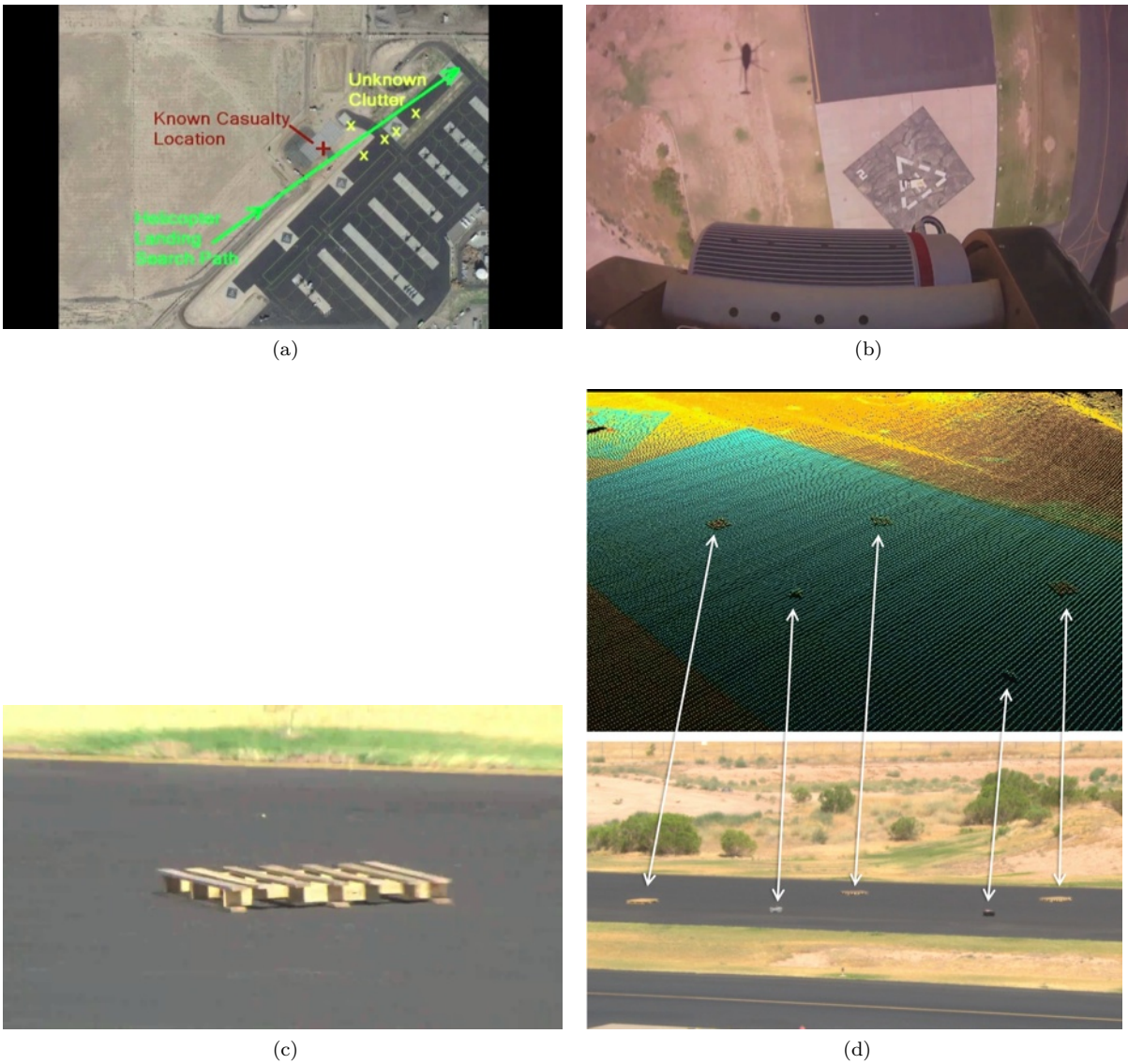
The helicopter takes off autonomously, is guided to a fixed search path and creates a terrain map. After the helicopter takes off the landing pads were made inaccessible. Figure 5.16 shows pictures from observer and onboard cameras and views from the input data that show small obstacles such as crates, and black plastic cases that need to be avoided as landing sites. The landing site evaluation is performed as the vehicle flies over. The final landing site is picked after the ground traversability plans are not changing because if we still have potentially good landing sites that might be available we should keep searching for landing sites for a while.

After the overflight the landing site is picked as shown in Fig. 5.17. The coarse evaluation shows that crates, pelican cases, and a cart that are parked on the helipads are detected and rejected. After the landing sites are found with the fine evaluation in Fig.5.17(c) the approach and abort paths are evaluated in (d). The path to the ground goal is shown in (e). Extension 6 is a video that shows one of the landing site search and landing runs.

Even though it was not utilized for planning in the real experiments the landing site evaluation algorithm was updating the information gain map as shown in Fig. 5.18 (Also see Extension 7). Initially no landing sites are known so there is a bias toward the ground goal. As landing sites are found in the red region (5.18b) the approach path becomes valuable to search. In Fig. 5.18c a ground path has been found and therefore all locations that are at the boundary between the ground path and unexplored cells become valuable since landing sites might exist that have a ground path. One can see that in south-east there is a large frontier because the area is flat (Also compare to the aerial image in Fig. 5.16a). In the north-west direction the area is blocked by a fence that the ladar detected and therefore no frontier is found. The information gain map is utilized in Chapter 6 for planning in simulation.

From the parameters of the lowest cost landing site an approach path is computed and sent to the vehicle. On this trajectory the vehicle turns around and descends at the desired sink rate and comes to a hover about 3 meters above the ground. After coming to a hover the vehicle orients itself to the optimal heading and then executes a touchdown. Part of the final approach segment is shown in Fig. 5.19 where the helicopter is close to the final landing site. The first view is a cockpit view of the clutter on the runway and vehicle on its way down to land. The second view shows the selected final landing site and other potential sites to land. The last picture shows the helicopter and the location of the ground goal. After a stable hover is achieved above the final goal location the vehicle touches down at the landing site.

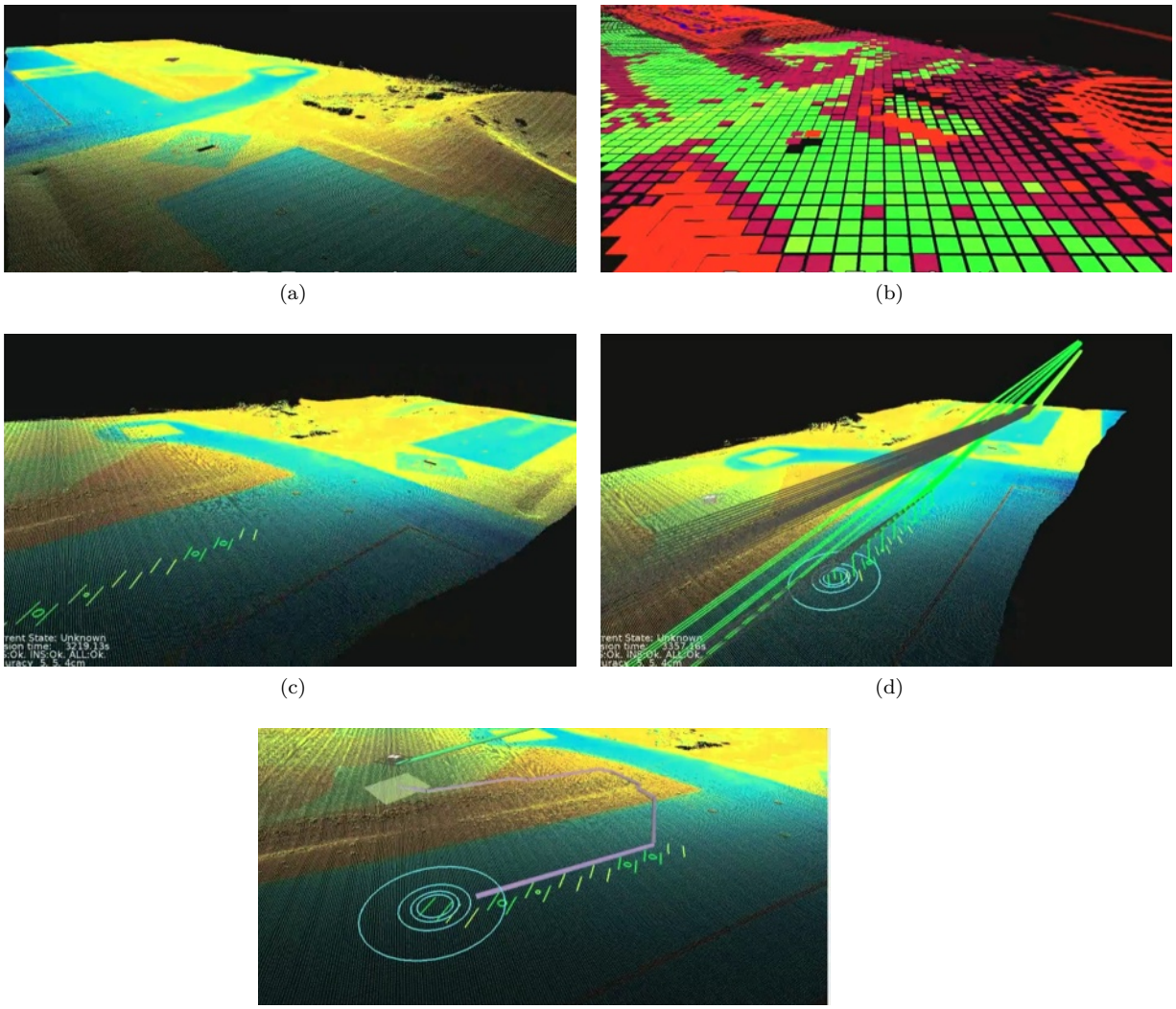
After system integration and testing runs, a total of 8 landing missions were demonstrated with varying ground clutter and approach directions. Figure 5.20 shows two typical landing missions. After an autonomous takeoff and climb out, the aircraft approaches the flight line from the south-west. It overflies the landing area at 40 knots and 150 ft AGL to search for acceptable landing sites. The computation time for landing zone evaluation is low enough that the vehicle can pick landing zones,



(a) (b)

(c) (d)

Figure 5.16: The problem setup and input sample data. In (a) one can see the straight line search path that was performed to look for landing sites. A ground goal was given by a "casualty" that needs to be picked up. In (b) one can see a downward looking view from the helicopter looking at some obstacles on the runway. (c) shows an example obstacle and (d) shows a set of obstacles. (d) shows a the correlation between obstacles (pallets, and plastic storage boxes) and the point cloud which is the input to the algorithm. The data was collected at 40 knots from 150 feet AGL.

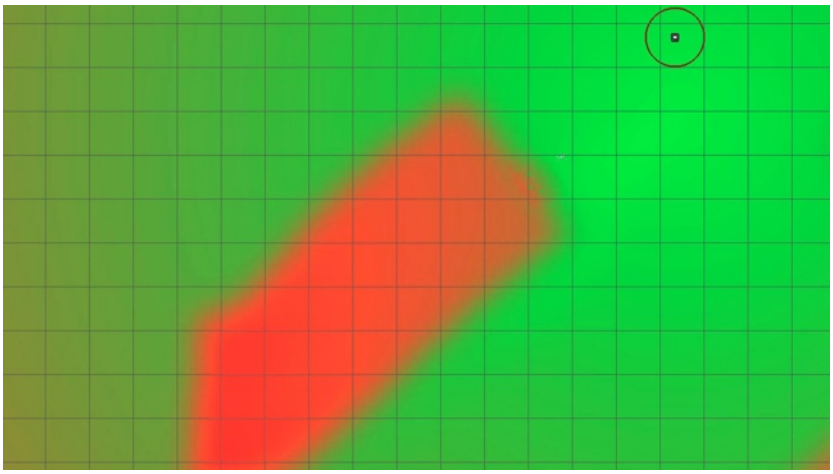


(a) (b)

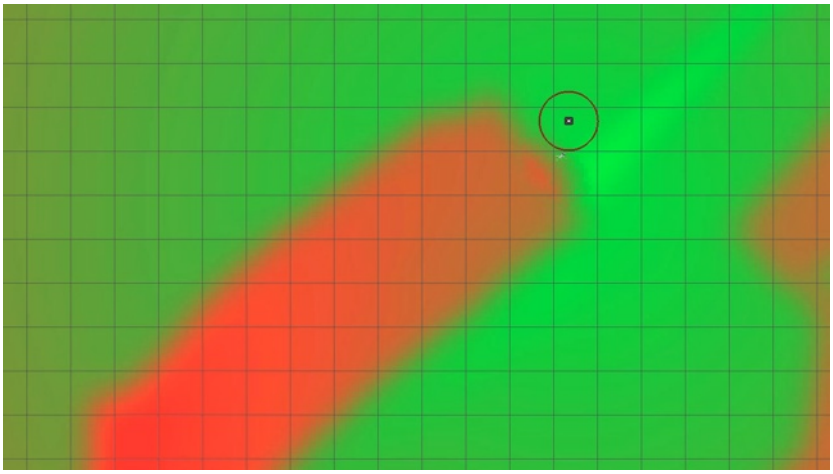
(c) (d)

(e)

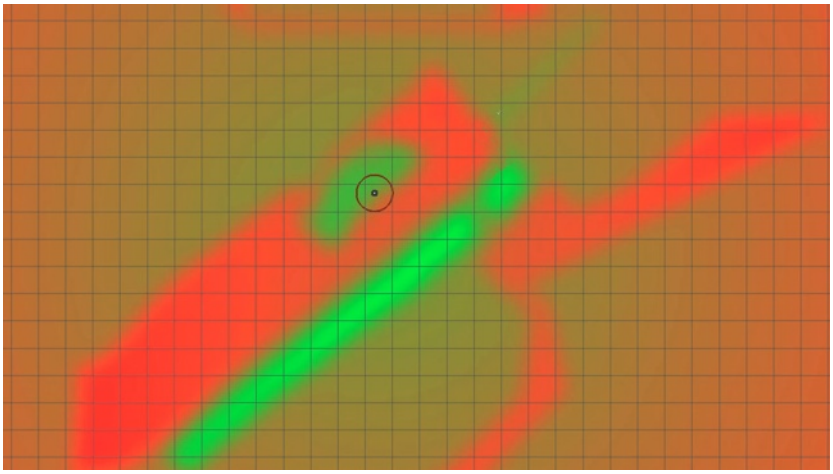
Figure 5.17: Example results for the landing site evaluation in Phoenix, AZ. (a) is the input point cloud. (b) shows the coarse evaluation. For a legend see Fig. 5.10. (c) shows good landing sites that were discovered by the vehicle and (d) shows the approach paths in green and the abort paths in grey. (e) shows the best site that is accessible from the ground goal. The purple trajectory shows the ground path to the ground goal ("casualty").



(a)



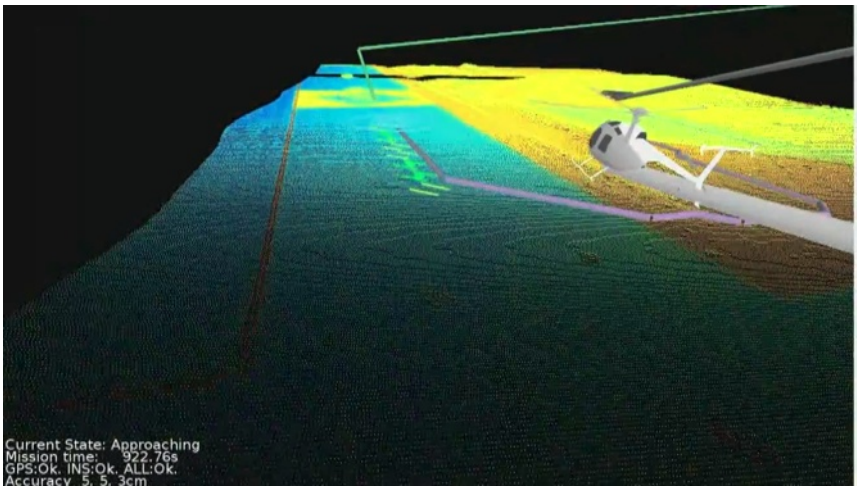
(b)



(c)

Figure 5.18: Information gain map for the problem setup from Fig. 5.16a. In (a) the robot has not yet discovered landing sites and the lowest cost is at the goal location (circle). In (b) landing sites without a path have been found and therefore the approach paths become more interesting to explore. In (c) a path has been found and there is a large frontier of places that are now valuable to search because a ground path could be found. Approach paths and holes in the map around the ground goal also have a higher information gain value.

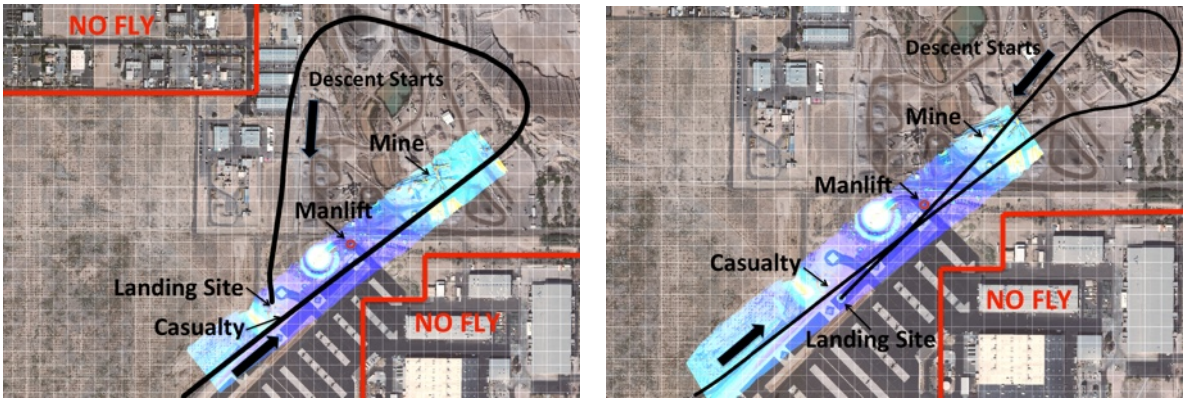
(a)



(b)

(c)

Figure 5.19: Landing approach and touchdown. Figure (a) shows a pilots view during the approach and (b) shows a similar view in the data. In (c) one can see a picture of the helicopter after touchdown and the ground goal location.



(a) (b)

Figure 5.20: Typical landing missions including survey overflight, descent on final and touchdown. In Fig. (a) the system is not penalized for flying the final approach through unsurveyed terrain and chooses to land from the north. In Fig. (b) the system is penalized for descending through unsurveyed terrain and therefore chooses an approach that maximizes coverage of already seen areas.

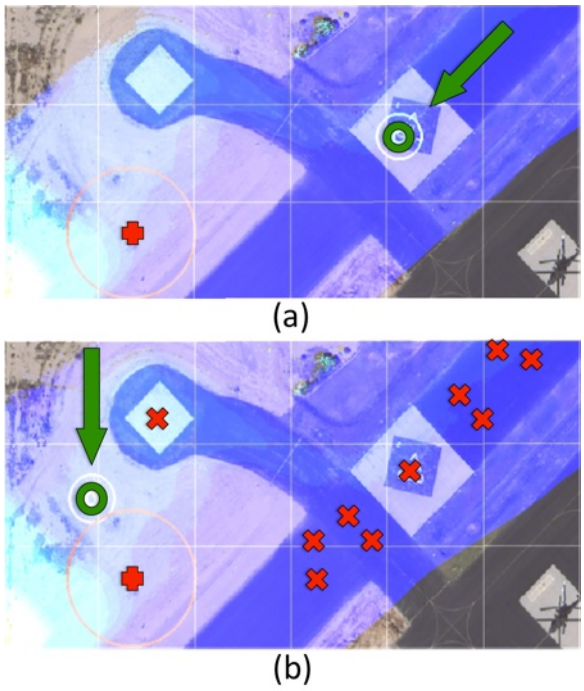


Figure 5.21: Example of chosen landing sites for two test cases. The ground goal is shown as a red cross and the approach direction and touchdown point are indicated with a green arrow and green circle respectively. Fig. (a) shows a landing without obstacles where the vehicle picked a location on the runway since it has the lowest cost. In Fig. (b) the runway is cluttered with obstacles (red x's. See Fig. 5.16d). The vehicle selected to land closer to the casualty instead even though the clearance is smaller.

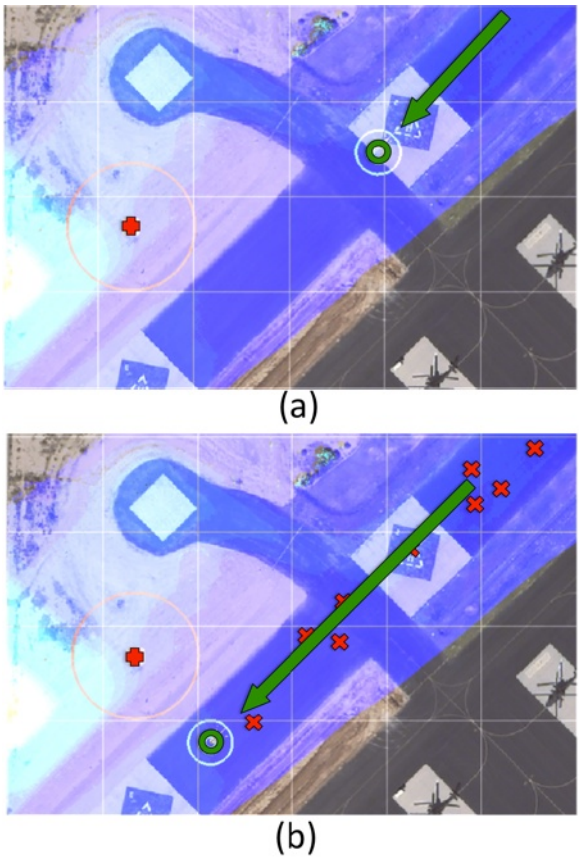


Figure 5.22: Example of chosen landing sites with the system preferring known approach directions. In Fig. (a) the system selects a site on the runway that is reachable with a low-cost path the site selected is similar to Fig. 5.21a. In Fig. (b) the vehicle will still prefer an approach through measured terrain. It picks a landing zone that is on the runway further away from the ground goal since it is the only clear-area that has an obstacle clear glide slope. The area close to the ground goal is free and clear however it is not chosen as a landing site since a hill (Bright area in lower-left) increases the approach cost and the approach is not known to be free.

	Mean		Total		
Algorithm	Time	Units	Time(ms)	Count(#)	Units
Adding Points	17	ms / 100000 points	2723	15826707	points
Coarse Evaluation	699	ms / 100000 cells	7575	10843248	cells
Fine Evaluation	4578	ms / 100000 sites	5446	118951	sites
Approach Evaluation	7663	ms / 100000 paths	9116	118951	paths
Approach Change Detection	21	ms / cycle			
Traversability Planner	113	ms / cycle			
CPU Utilization	30	percent			

Table 5.4: Computation times for five successful landing runs of the autonomous helicopter.

and approach paths in real-time. The point cloud, aerial image and resulting path is shown in Fig. 5.20 for two typical missions. The approach cost ω_6 was varied between Fig. 5.20a and 5.20b to penalize for an unknown approach cost.

A close-up of two missions flown with a low ω_6 (approach cost) weight are shown in Fig. 5.21. Since no obstacles are present in Fig. 5.21a the vehicle prefers a landing site on the runway since it is a wide open area with a low landing site cost. As we cluttered the runway Fig. 5.21b the vehicle instead now prefers a closer site with a shorter path. This result was unexpected since we expected the vehicle to land on the runway instead because there was a no-fly zone to the north. However, the approach it found went east of the no-fly zone and landed in a feasible location.

In the next two experiments the approach cost weight was increased (Fig. 5.22). This did not change the outcome of landing without obstacles since it also previously landed on the runway. In the case of obstacle clutter however the vehicle now preferred an approach that flew over the scanned area. Since most of the obstacles were low it was able to overfly the obstacles before landing.

Table 5.4 shows the computation times for the optimized algorithms that minimize re-computation of new data. The computation was performed onboard with Intel 2.6 GHz Core 2 Quad computers Adding new data is fast and takes currently about 17 ms/100000 points which is much faster than realtime. The coarse evaluation is very fast at $699ms/100000$ cells. The fine evaluation is much slower which justifies the coarse evaluation to limit the number of fine evaluations performed. Approach evaluation also requires many evaluations and is therefore slower.

5.4 Discussion

We presented, to our knowledge, the first geometric model-based algorithm that gives estimates of available landing sites for VTOL aerial vehicles. In our algorithm we are able to incorporate many constraints that have to be fulfilled for a helicopter to be able to land. While the current state of the art is similar to our coarse evaluation algorithm, the fine evaluation allows us to consider more aerial-vehicle-relevant constraints and reject false positives in the coarse evaluation. Furthermore, since the two paths of evaluation are based on two different representations of the terrain, we conjecture that the combined system will be more robust to failure cases of a pure cell-based representation. Additionally, we include constraints based on the approach, abort and ground paths to determine the feasibility of a landing sites.

We also presented results based on real sensor data for landing site evaluation at vegetated and urban sites. All of the hard constraints on accepting or rejecting a landing site are based on the actual constraints of the vehicle geometry and weight distribution. However, which of the good landing sites is preferable is an open problem because combining the different metrics is non-trivial. In future work we propose to learn the weight vector of the landing site evaluation based on human preferences or self-supervised learning in simulation.

Results of tests for the first autonomous helicopter that is able to select its own landing sites was presented in this chapter. Results are shown in the context of a casualty evacuation mission where the path to a ground goal also has to be found.

Even though the inertial navigation system we are using has a high accuracy (0.01 degree reported accuracy IMU, 1cm GPS) it is not sufficient to register the points with high enough accuracy to evaluate landing sites in front of the helicopter at the glide slope, because the smallest potentially lethal obstacle we consider is the height of a rail (10cm). One could apply scan matching techniques such as the one proposed by Thrun et al. [Thrun et al., 2003] to smooth the matching of adjacent scans, however, in the process one might smooth over important obstacles.

Currently our algorithm will accept water as a potentially good landing site since it is flat and smooth. We have observed two behaviors of water with our ladar scanner: In some water the beam is reflected and we get a smooth surface, and in other situations we do not get a reflection and therefore no measurement. No measurement in this case is good because we will reject such areas. On the other hand, we might keep searching the water areas because we are not able to gather data on them. Some ways to directly sense water would be with other sensor modalities such as a camera of a suitable spectrum, or a radar.

While water is a false positive, we will also reject potentially good landing sites (false negatives) because we currently classify vegetation as roughness. We have no information that allows us to distinguish grass from steel wire, for instance, and therefore we take a pessimistic approach.

We are making several assumptions that might not hold in all conditions: the algorithm assumes that small objects will not damage aircraft (no FOD), and the evaluation assumes that the terrain can bear the load of the helicopter. These assumptions could be relaxed by incorporating secondary sensors such as cameras that can classify the material of the terrain.

In future work one could encode the concept of prior known helipads as low cost regions in the landing site cost calculation. The only landing gear we model in the fine evaluation are two skids. In future work we would like to make the algorithm more flexible to be applicable to different kinds of landing gear.

6 Multiple-Objective Motion Planning

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 unmanned air vehicles 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 need a framework that can combine multiple objectives that need to be discovered and might change over the course of a single mission.

In the previous chapters we examined the case where a vehicle has to perform one particular mission. Either avoid obstacles while visiting a sequence of goal points or searching for landing sites and landing automatically. However, even though problems are relevant, the planning algorithm should be agnostic to the particular mission type.

The safety of a reactive landing site search cannot be guaranteed for full-size helicopters because one can no longer assume that stopping is possible in time. One reason is that a strong braking acceleration can cause two dangerous conditions: low-rotor RPM and a vortex ring state. Therefore, it becomes infeasible to guarantee safety by decreasing the speed if the search behavior would cause a collision. Instead the algorithm's commands have to be safe by construction and combinations of different objectives should not influence the safety.

Here we propose a framework (Fig. 6.1) 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 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. These attributes are exploited by planning algorithms to increase the planning horizon and plan in real-time.

In summary the main contributions of this chapter are a

- 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.

Below we describe the general problem (6.1) and then show the approach(6.2). We conclude with an application of the framework to multiple missions on an unmanned helicopter(6.3) and discuss (6.4).

6.1 Problem

The problem of planning with changing objective functions is formally defined here. While the formulation below is general, in Sec. 6.2.5 we give the specific details of the problem instance for different missions of an autonomous helicopter.

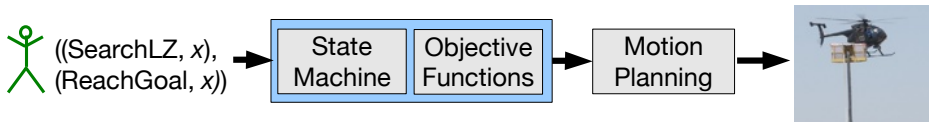


Figure 6.1: A high-level overview of the planning framework. The input to the motion planner is a objective function that is set through a state machine based on user commands. The particular missions we address in this paper are "search for landing zones close to x," "approach a landing zone," and "reach goal x."

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 $\mathbf{P} = \{\mathbf{p_0}, \ldots, \mathbf{p_n}\}$. Alternatively we will refer to the trajectory by time steps as $\{\mathbf{p}(0), \ldots, \mathbf{p}(t_f)\}$ where t_f is the final time. The length of the trajectory is denoted as $|\mathbf{P}|$. 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.

Discrete Mapping Finding global solutions in continuos 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 continuos and discrete states with the following functions:

$$v = \mathsf{node}(\mathbf{x}),\ \mathbf{x} = \mathsf{state}(v) \tag{6.1}$$

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 $J(\mathbf{B})$.

Motion Model The forward motion model is defined as a differential equation as

$$\dot{\mathbf{x}}(\mathbf{x}, u, t) = f(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t) \tag{6.2}$$

with the command input

$$\mathbf{u}(\mathbf{x}, t, \mathbf{P}_c) \tag{6.3}$$

defined by the state, time, and command vector $\mathbf{P}_c$. We explicitly specify $\mathbf{P}_c$ since it is the command vector that is optimized. The initial state is given by $\mathbf{x}(0) = \mathbf{x_0}$ and the equation of motion is integrated as follows:

$$\mathbf{x}(t_f) = \mathbf{x}(0) + \int_0^{t_f} \dot{\mathbf{x}}(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t)dt \tag{6.4}$$

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:

$$J(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t_f) = \int_0^{t_f} c(\mathbf{x}(t))dt \tag{6.5}$$

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

$$\begin{array}{rc} \text{minimize:} & J(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P_c}), t_f) \\ \text{subject to:} & \dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & J(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t_f) < \infty \end{array} \tag{6.6}$$

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

Mission State Machine The operator (or higher level mission planning) can specify the mission through a string over an input dictionary. The words from the dictionary express the command to perform a state transition. However a state transition will only be completed if the associated guard is true or in other words the cost function associated with each state is below the threshold or the segment has timed out. The definition of the state machine is similar to a hybrid automaton [Henzinger, 1996] however since this is a different problem some unnecessary fields have been removed and others added.

The state machine is part of the cost function and therefore it is possible to change mission states during an optimization problem. In the next paragraph we explain how the state machine interfaces with cost functions. The state machine is defined as follows $H = (\Sigma, \mathcal{S}, \mathcal{X}, \mathcal{T}, \mathcal{C}, M, D)$.

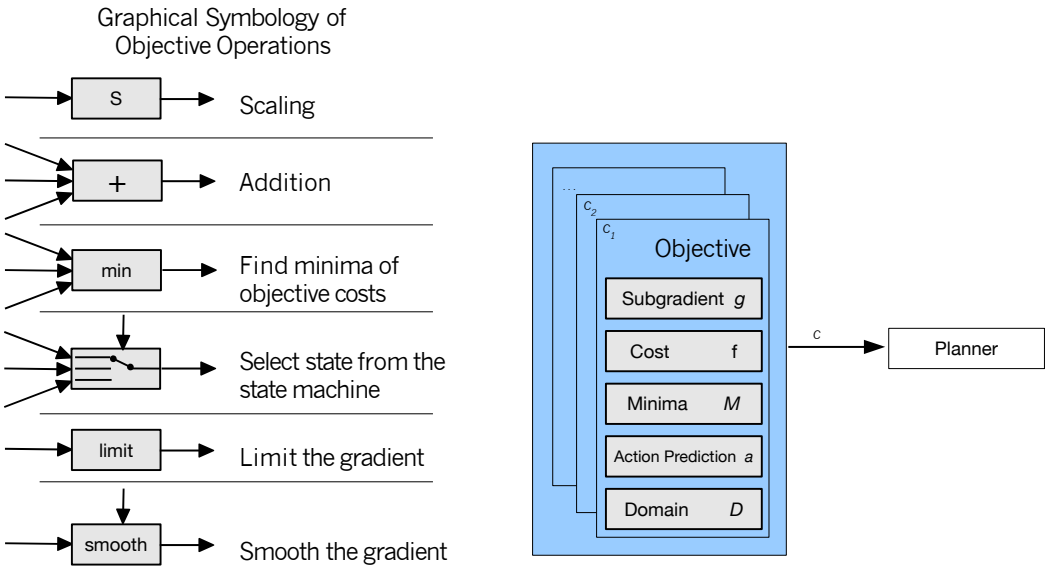
- Σ is the input mission dictionary that defines commands that the operator can give to influence the state machine.
- $\mathcal{S}$ is the discrete set of modes.
- $\mathcal{X}$ is the state space in the continuous system.
- $\mathcal{T}$ is the time.
- $\mathcal{C} := \{c_s, s \in S\}$ is the set of cost functions associated with each state.
- $M := \{m_i, \ldots, m_m \in (\Sigma \times \mathcal{P})\}$ is a list of mission tuples consisting of an item from the input dictionary and a position.
- $D := \mathcal{S} \times M \times \mathcal{C} \times \mathcal{T} \to \mathcal{S}$ is the state transition function.

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. In many cases we have to numerically search over a limited domain to find the minima. However, if we augment the cost function with a method for us to directly tell us its minima, we can exploit the structure of certain cost functions. Consider for example a goal cost function which has exactly one minimum at the goal:

$$c(\mathbf{p}) = (\mathbf{p}_{goal} - \mathbf{p})^2 \tag{6.7}$$

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 and a grammar of how the functions can be combined. The cost function tuple is $C = (f, g, \mathcal{M}, a, \mathcal{D})$ where



(a) Operations on objective functions.

(b) Interface to planner.

Figure 6.2: Objective functions define the planning interface. An objective function is a combination of several primitive objective functions with the operations shown in (a). Additionally each objective allows queries not only to the cost and gradient at a location, but also to the minima, domain as shown in (b). Since the objectives are not static we also have the function a that allows predicting the outcome of an action.

- $f := \mathcal{P} \times \mathcal{T} \to \mathbb{R}_0^+$ is the cost of being at a certain position in a certain time.
- $g := \mathcal{P} \times \mathcal{T} \times \mathbb{N} \to \mathcal{P}$ 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.

The following operations are defined on cost functions to allow us to express multiple-objectives and flexible missions:

- $c' = \omega c$: Scales the cost function c by ω.

$$c' = (\omega f, \omega g, \mathcal{M}, a, \mathcal{D})$$

- $c' = c_1 + c_2$: Adds two cost functions.

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

- $c' = \min c$: Finds the minimum function values as minima. This operator ignores minima from c and evaluates the cost function f instead.

$$c' = (f, g, \min(f), a, \mathcal{D})$$

- $c' = c[h]$: Selects the best function based on the associated state in the state machine. Note that the switching occurs as part of the optimization problem and therefore it is possible to optimize a path across switching states.

$$c' = (f[h], g[h], \mathcal{M}[h], a[h], \mathcal{D}[h])$$

- $c' = \text{limit}(c, \omega)$: Limits the subgradient of c to ω.

$$c' = (f, \min(1, \frac{\omega}{|g|})\, g,\, \mathcal{M}, a, \mathcal{D})$$

- $c' = \text{smooth}(c, \mathbf{W})$: Distributes the gradient update with a smoothing matrix W. This changes the trajectory smoothly to account for the fact that a gradient at one location can affect the whole resulting path.

$$c' = (f, \mathbf{W}g, \mathcal{M}, a, \mathcal{D})$$

Now that we have defined the elements and elementary operations on the cost function we need to define a language. Expressions in this language can be parsed and the language permits a graphical or textual expression of an optimization problem. The combination of cost functions is defined as a context-free grammar $G =< \mathbf{T}, \mathbf{N}, \mathbf{S}, \mathbf{R} >$. The terminal symbols T are defined as the cost functions c_i, scale factors ω_i,and operations:

$$\mathbf{T} = \{c_1, \ldots, c_n, \omega_i, \ldots, \omega_m, h_1, \ldots, h_p, , , +, \min, [], \text{limit}, \text{smooth}, \mathbf{W}\}$$

The non-terminal symbols and the start symbol are

$$\mathbf{N} = \{C, D, W, S, L, M\}, \mathbf{S} = S$$

Now the rules for the grammar can be defined as:

$$\begin{aligned}
\mathbf{R} = \{ S &\rightarrow C \\
S &\rightarrow WS \\
S &\rightarrow +(L) \\
L &\rightarrow S, M \\
M &\rightarrow L \\
M &\rightarrow \emptyset \\
S &\rightarrow \min(S) \\
S &\rightarrow S[H] \\
H &\rightarrow h_1 | \ldots | h_p \\
S &\rightarrow \text{limit}(S, W) \\
S &\rightarrow \text{smooth}(S, \mathbf{W}) \\
C &\rightarrow c_1 | \ldots | c_n \\
W &\rightarrow \omega_1 | \ldots | \omega_m \}
\end{aligned}$$

Using this grammar cost function expressions can be succinctly defined and parsed. As illustrated in Fig. 6.2a, one can also express the cost functions graphically in a tree like structure.

6.2 Approach

Our approach makes the problem feasible 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 distinct initial guesses while the continuous part optimizes within a local perturbation 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 motivation for using a coarse planner in the

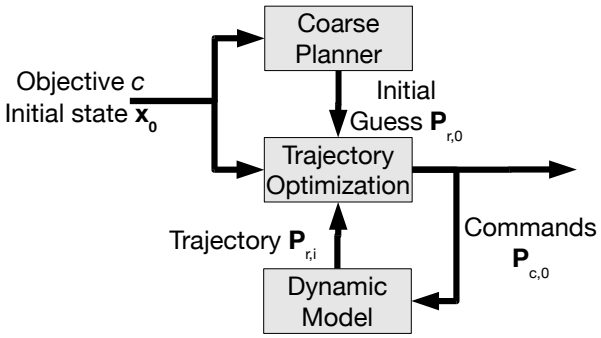


Figure 6.3: Detailed planning architecture. The input to the planning algorithm is the objective and the initial state and the output is the command given to the vehicle. In our approach a coarse initial guess is determined based on the minima and cost of the objective and the coarse planner plans in a discretized state space.

first step is that the resulting trajectory will be modified anyway and there is no advantage in incorporating the dynamics at this planning stage.

A pure gradient-based approach will work only if a consistent gradient from the initial state to an optimum exists. Sometimes this is the case in tracking applications or very reactive search scenarios however for more structured cost functions we need to consider the optima of the cost function to be able to achieve a reasonable behavior. It is therefore necessary, to optimize over a trajectory planning horizon instead of only looking at the local gradient.

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 towards 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. 6.3.

6.2.1 Problem Approximation

An intuitive method to express the planning problem is in terms of an optimal control policy which minimizes 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 uncountable we look at a reduced set of parameters $\mathbf{P}_c$. The set of parameters makes the problem feasible since only a discrete set of parameters 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 defined 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 $\mathbf{P}_c$.

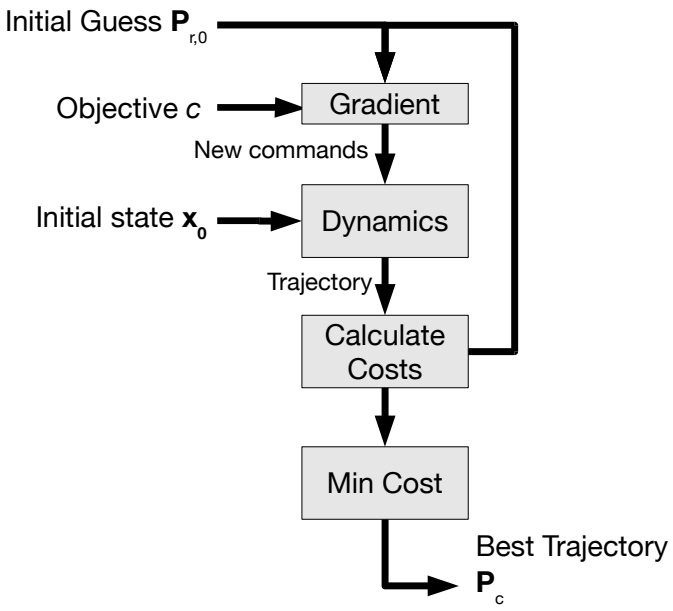


Figure 6.4: The trajectory optimization algorithm flow chart. After we are given an initial guess, state, and objective the algorithm changes the command trajectory to minimize the resulting cost of the trajectory.

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

6.2.2 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 trajectory optimization algorithm performs a gradient descent in the command trajectory space based on the gradient of the resulting trajectory. Since the gradient at a particular location also influences the overall shape of the trajectory the gradient is distributed over the trajectory. The change to the trajectory is then mapped to the command trajectory and the new command trajectory is updated. The cost of the new trajectory is calculated and a set of cost values is stored. Optimizing the trajectory with respect to the gradient does not necessarily optimize the overall cost and smoothness of the trajectory. Therefore after convergence the best trajectory is picked from the set of all the optimized trajectories and sent to the robot.

The algorithm for trajectory optimization is shown in Alg. 6.1. 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.

Algorithm 6.1 $\mathbf{P}_c$=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,$\mathbf{P_{r,0}}$= initial reference trajectory, $\mathbf{P_{c,0}}$= initial 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
$b = \text{argmin}_{\forall i \in \{0,\ldots,m\}} t_i$
$\mathbf{P}_c = \mathbf{P}_{c,b}$

Algorithm 6.2 $\mathbf{Q}_c$=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.

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

6.2.3 Initial Guess Generation

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. 6.2. The recGuess recursively explores the outcome of predicting actions to increase the planning horizon in Alg. 6.3.

If, for example, a minimum is close we would like to explore where the next good position might be, to have a sufficiently long path segment to optimize. At least two initial guesses are added: A simple and a planned initial guess. The simple initial guess does not consider the cost of traversing a cell while the planned initial guess tries to optimize the cost for planning.

Simple Initial Guess The first algorithm is the "simple" initial guess that just connects a start configuration $\mathbf{x}_0$ 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. 6.2.5 we show how 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.

Plan Initial Guess The second algorithm searches the planning graph of valid trajectories. One could find the optimal path using a A* graph 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.[Abraham et al., 2010] 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.

Algorithm 6.3 $\mathbf{Q}_c$=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**
  $\mathbf{Q}_{t1}$ = PlanInitialGuess$(G, \mathbf{x}, c) \cup$ 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
        $c_{new} = c$
        $a_{c_{new}}(\mathbf{P}_{c,i})$
        $\mathbf{Q}_{t2}$ = 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

Algorithm 6.4 $\mathbf{Q}_c$=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**
  $\mathbf{Q}_c = \mathbf{Q} \cup$ simple$(\mathbf{x_0}, \mathbf{P_i})$
end for

Algorithm 6.5 $\mathbf{P}_c$=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

Each subpath of the alternative route has to be locally-optimal path (or 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. 6.5) 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 [Niederreiter, 1988]. 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 fulfills 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.

In future work it would be desirable to plan in a D* like fashion incrementally and reuse as much precomputation as possible. However in this algorithm it is difficult to enable D* like optimizations because we have to use a bi-directional search to enable finding alternative routes. Furthermore the locallyOptimal function requires another A* search that is different for every path that is checked.

6.2.4 Base Cost Function Definition

The following cost functions form a vocabulary that is used to express missions. Some of these functions might only be useful for aerial vehicle applications. The first objective considered is the obstacle objective. In Chapter 4 we demonstrated an algorithm that efficiently updates the distance to the closest obstacle using the limited incremental distance transform. Using the distance function $d(\mathbf{P})$ we can express the obstacle objective 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 used by the LIDT.

The next cost function is setting a desired altitude above ground (AGL) based on a digital elevation model (DEM). The cost depends on the altitude and the gradient is only along one dimension. The domain of the function is either all of $\mathcal{P}$ or one could limit it to the extend of the available DEM. The tuple of the altitude cost function is expressed as

$c_{alt}(\mathbf{P}) = (g_{alt}, f_{alt}, M_{alt}, a_{alt}, D_{alt})$

- $g_{alt} = \begin{bmatrix} 0 \\ 0 \\ z_{dem}(\mathbf{P}) + z_{alt} - p_{z`} \end{bmatrix}$ where $z_{dem}(\mathbf{P})$ is the altitude of the DEM and z_{alt} is the desired altitude above ground.
- $f_{alt} = |(z_{dem}(\mathbf{P}) + z_{alt} - p_{z`})|^2$
- $M_{alt} = \emptyset$
- $a_{alt} = \emptyset$
- $\mathcal{D}_{alt} = \mathcal{P}$

So far we have only expressed what we would like to stay away from but have not explicitly expressed a goal waypoint. The waypoint function defines the cost as the squared distance from the goal at the final point of the path. Similarly, the only gradient is defined at the final point on the path to pull the trajectory towards the desired goal. A minimum is located at the goal since this would be

a point we would like to reach. The action prediction might remove that minimum if we are close enough or too much time has passed.

$c_{wp}(\mathbf{P}) = (g_{wp}, f_{wp}, M_{wp}, a_{wp}, D_{wp})$

- $g_{wp} = \begin{cases} 0 & \text{if } i \neq n \\ \mathbf{p}_{goal} - \mathbf{p}_i & \text{otherwise} \end{cases}$ where $\mathbf{p}_{goal}$ is the goal position.
- $f_{wp} = |g_{wp}^2|$
- $M_{wp} = \{\mathbf{p}_{goal}\}$
- $a_{wp} = action_{goal}(\mathbf{P}, T, \mathbf{p_{goal}})$
- $\mathcal{D}_{wp} = \mathcal{P}$

Another behavior we would like to be able to express is an exploration behavior which can be expressed in terms of the information gain at a location. As we traverse the value might change however it is not necessarily the case. We assume that the information gain is represented as a grid in the state space of the vehicle and that we can predict the outcome using $action_{info}$ for planning purposes. The information gain is defined in terms of the following tuple:

$c_{info}(\mathbf{P}) = (g_{info}, f_{info}, M_{info}, a_{info}, D_{info})$

- $\mathsf{info}(\mathbf{P})$ is the information gain function. See Sec. 5.2.7 for details.
- $g_{info} = \nabla \mathsf{info}(\mathbf{P})$
- $f_{info} = \mathsf{info}(\mathbf{P})$
- $M_{info} = \min_{\forall \mathbf{p} \in D} \mathsf{info}(\mathbf{p})$
- $a_{info} = action_{info}(\mathbf{P})$
- $\mathcal{D}_{info}$ is the domain and resolution of the information gain grid.

During an approach the helicopter needs to fly along a certain path and only deviate from it if necessary. The closer the helicopter is to the endpoint the more it should follow the approach path. The goal is set based on the reference path $\mathbf{P_{ref}}$ and therefore we have a single minimum at $\mathbf{p}_{ref,n}$.

Approach Path:

$c_{app} = (g_{app}, f_{app}, M_{app}, a_{app}, D_{app})$

- $g_{app} = (\mathbf{p_{ref,i}} - \mathbf{p_i}) \max(0, (1.0 - \frac{dist(\mathbf{p_{ref,n}}, \mathbf{p_i})}{d_{approach}}))$
- $f_{app} = g_{app}^2$
- $M_{app} = \mathbf{p_{ref,n}}$
- $a_{app} = \emptyset$
- $\mathcal{D}_{app} = \mathcal{P}$

The last objective is a measure of how much energy is necessary to follow a given trajectory. This objective will compete with the other objectives since it resists motion and will try to straighten the path. The implementation of this objective depends on the specific vehicle and will be addressed in the next section.

$c_{energy} = (g_{energy}, f_{energy}, M_{energy}, a_{energy}, D_{energy})$

- $g_{energy} = \nabla \mathsf{energy}(\mathbf{P})$
- $f_{energy} = \mathsf{energy}(P)$
- $M_{energy} = \emptyset, a_{energy} = \emptyset, \mathcal{D}_{energy} = \mathcal{P}$

We defined a set of six generally useful objectives that form the basic vocabulary of operating a UAV close to obstacles. These base functions can be combined with the operations in the language defined above to form missions that can be performed. Together all these functions form a set of optimization problems that the planning algorithms will optimize.

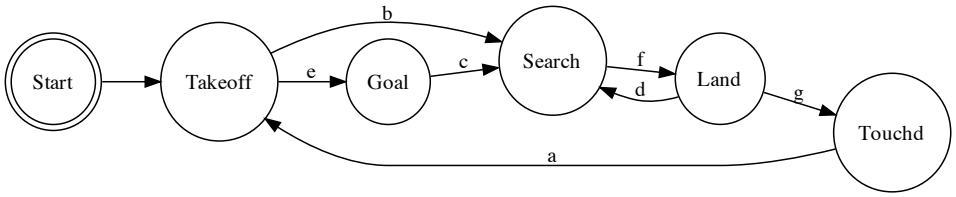


Figure 6.5: The state machine for the example unmanned helicopter. The available missions are takeoff, search for landing sites, land, and reach a goal point. The state transitions depend on events and the current cost.

S_{new}	S	M	C	T	Label	Description
Takeoff	Touchd	Takeoff			a	Initiate take off
Search	Takeoff	Search	$f_{to} < \lambda_{to}$		b	Search for LZ & Reached Min. Alt.
Search	Goal	Search	$f_{oa} < \lambda_{oa}$		c	Search for LZ & Close to goal
Search	Land	NoLZ			d	No valid LZ anymore
Goal	Takeoff	Goal	$f_{to} < \lambda_{to}$		e	Reach goal & Reached Min. Alt.
Land	Search	HaveLZ		$t > t_{minsearch}$	f	Have LZ: Approach next
Touchd	Land		$f_{lz} < \lambda_{lz}$		g	Close enough to LZ

Table 6.1: Finite state machine transition events with the guard $C \vee T$ for the state machine shown in Fig. 6.5.

6.2.5 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.

6.2.5.1 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. 6.5. The transition between the different states depends on the input string (events) given by the user and robot internal strings.

State	Obst	Info	Alti	Goal	Appr
Takeoff				1	
Touchdown				1	
Land	1				1
Search	1	1	1		
Goal	1		1	1	

Table 6.2: Activated sub-objective functions for different states in the state machine shown in Fig. 6.5.

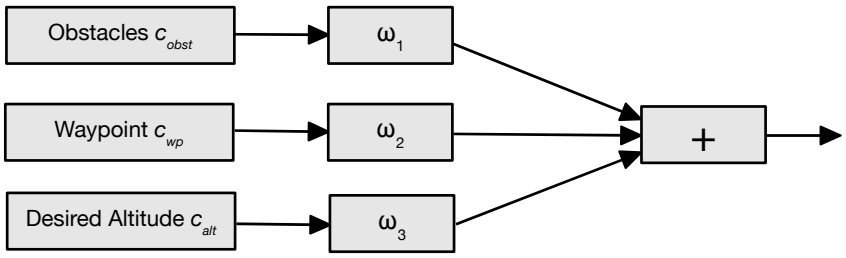


Figure 6.6: Graphical representation of the obstacle avoidance objective.

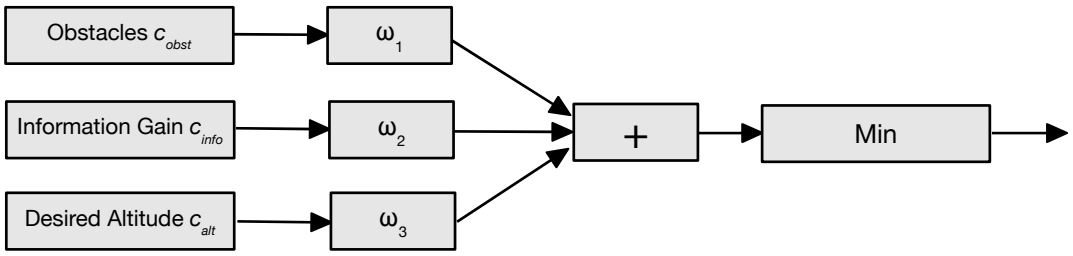


Figure 6.7: Graphical representation of the search for landing sites objective.

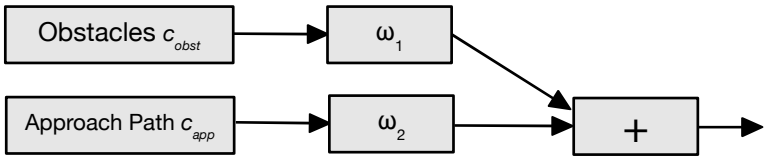


Figure 6.8: Graphical representation of the planning objective for approaching a landing site.

The availability of landing sites is received as an internal string of $HaveLZ$ or $NoLZ$. A $HaveLZ$ word is produced if a landing site is available by the landing zone evaluation algorithm. The state is then switched to land if enough search time has passed. If for some reason all the landing sites should be infeasible the algorithm can switch back to search. The complete state transition table is shown Tab. 6.1. For each state a different set of objective functions will be used (See Tab. 6.2).

For the autonomous helicopter there are several possible missions. We can try to land close to a goal location ($Search$), reach a goal point ($Goal$), and takeoff ($Takeoff$). For these three commands we can be in five states $\mathcal{S}$ with the corresponding cost functions. In summary one can define the state machine $H = (\Sigma, \mathcal{S}, \mathcal{X}, \mathcal{T}, \mathcal{C}, M, D)$ as

- $\Sigma :=$ {Takeoff, Goal, Search, HaveLZ, NoLZ},
- $\mathcal{S}$: = {Goal, Search, Land, Takeoff, Touchd},
- $\mathcal{X}$ is the state space of the dynamics,
- $\mathcal{T}$ is the time starting at 0,
- $\mathcal{C} := \{c_{oa}, c_{lz}, c_{app}, c_{td}, c_{to}\}$,
- $M := \{m_i, \ldots, m_m \in (\Sigma \times \mathcal{P})\}$ is the input string (events) of the mission defining the next desired state and position if required,
- $D := \mathcal{S} \times M \times \mathcal{C} \times \mathcal{T} \rightarrow \mathcal{S}$ is the state transition function and is shown in Tab. 6.1.

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} :

$$c_{s,} = \text{limit}(\text{smooth}(c_{combined}, \mathbf{W}), w_1) + \text{limit}(c_{energy}, w_2) \tag{6.8}$$

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:

$$c_{combined} = c[H] = \{c_{to}, c_{td}, c_{app}, c_{lz}, c_{oa}\}[H] \tag{6.9}$$

For each mission state the cost function expression that is optimized is different. For example, for reaching a goal point, one has to avoid obstacles c_{obst} while reaching a goal point c_{wp}. Usually for aerial vehicles it is also desirable to hold an altitude to minimize energy usage and therefore additionally the cost function includes a desired altitude above ground with c_{alt}. A graphical representation of the cost function is shown in Fig. 6.6 and it can be expressed as follows:

$$c_{oa} = w_1 c_{obst} + w_2 c_{wp} + w_3 c_{alt} \tag{6.10}$$

The next mission is to search for landing sites where we still want to avoid obstacles and keep an altitude. This time the altitude is set by the optimal height of sensing for the sensor which depends on the range and field of view. The main objective that steers the vehicle to different locations is the information gain c_{info} . It will have several minima and represents the knowledge about landing sites. Since we have to combine the cost of obstacles, altitude, and information we cannot gain an efficiency advantage in asking for the minima and have to use the min operation to calculate the actual cost function minima over the domain to generate the initial guess as shown in Fig. 6.7 and as an expression:

$$c_{lz} = \min(w_1 c_{obst} + w_2 c_{info} + w_3 c_{alt}) \tag{6.11}$$

During the approach the algorithms needs to reach a final goal and also follow a preplanned path that we know was obstacle free and will guide the vehicle smoothly to a hover location above the

ground. Therefore two objectives the c_{obst} obstacle objective and the c_{app} approach objective are combined to guide the vehicle as shown in Fig. 6.8 and as a combined cost function like this:

$$c_{app} = w_1 c_{obst} + w_2 c_{app} \tag{6.12}$$

During takeoff and touch down only the waypoint objective is activated since it is a controlled interaction with the terrain. It is assumed that the vehicle will try to slowly move towards a touch-down position or a final take-off hover location:

$$c_{td} = w_1 c_{wp},\ c_{to} = c_{td} \tag{6.13}$$

The consistent and compact representation of the problem using the cost function language enables the operator to specify flexible missions and enables further reasoning on the behavior of the system.

6.2.5.2 Vehicle Model

The dynamical model of the Unmanned Little Bird helicopter is too detailed and is therefore not suitable to be initialized by motion planning. Instead we developed a simplified idealized model that produces a behavior similar to the more detailed model when comparing resulting paths.

Dynamics The state and input to the forward are expressed as

$$\mathbf{x} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ v_x \\ v_y \\ v_z \\ \theta \\ \dot{\theta} \\ \dot{\gamma} \end{bmatrix},\ \mathbf{u} = \begin{bmatrix} v_{xc} \\ v_{yc} \\ v_{zc} \\ \dot{\theta}_c \end{bmatrix} \tag{6.14}$$

where $\mathbf{p}$ is the position, $\mathbf{v}$ is the body local velocity, θ, $\dot{\theta}$ are the heading and rate respectively, and $\dot{\gamma}$ is the roll rate. $\mathbf{v}_c$ is the command velocity, and $\dot{\theta}_c$ is the desired heading rate. Formally we can express the model as the ordinary differential equation (ODE)

$$\dot{\mathbf{x}}(\mathbf{x}, u, t) = f(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t) \tag{6.15}$$

and the controller can be expressed as

$$\mathbf{u}(\mathbf{x}, t, \mathbf{P_c}) \tag{6.16}$$

Since helicopters behave different for different speeds there is a switching point where the behavior of the control changes. At high speed (above 20 knots) the vehicle will execute coordinated turns and couple the roll and yaw commands. At low speed each axis is decoupled and the vehicle can be positioned anywhere in space. The position and velocity integration is common between the two models:

$$\dot{p} = Rv,\ \dot{v} = G_{v_n}(v_c - v) \tag{6.17}$$

At low speed the heading rate is directly commanded:

$$\ddot{\theta} = G_{\dot{\theta}}(\dot{\theta}_c - \dot{\theta}) \tag{6.18}$$

The high speed model assumes that the vehicle is performing a coordinated turn and therefore instead of integrating the heading rate directly the heading rate is converted to a bank angle and the operations are performed on a bank angle control:

$$\gamma_{des} = \tan(\frac{\dot{\theta}_c \cdot v_x}{g}),\ \gamma_{curr} = \tan(\frac{\dot{\theta} \cdot v_x}{g}) \tag{6.19}$$

γ_{des} is limited to 30 degree bank angle.

$$\ddot{\gamma} = G_{\dot{\gamma}}(\gamma_{des} - \gamma_{curr}) \tag{6.20}$$

and we convert the bank angle back to a heading rate change command to integrate:

$$\ddot{\theta} = \frac{g}{v_x} \cdot \tan(\gamma) \tag{6.21}$$

Overall the model can be summarized as follows:

$$f(\mathbf{x}(t), \mathbf{u}(\mathbf{x}, t, \mathbf{P}_c), t) = \frac{\delta}{\delta t}\begin{bmatrix} \mathbf{p} \\ \mathbf{v} \\ \theta \\ \dot{\theta} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} limit(Rv, v_{max}) \\ limit(G_v(v_c - v), a_{max}) \\ limit(\dot{\theta}, \dot{\theta}_{max}) \\ \begin{cases} \frac{g}{v_x}\tan(\gamma) & |v| \geq s_{max} \\ G_{\dot{\theta}}(\dot{\theta}_c - \dot{\theta}) & otherwise \end{cases} \\ \begin{cases} G_{\dot{\gamma}}(\tan(\frac{\theta_c v_x}{g}) - \tan(\frac{\theta v_x}{g})) & |v| \geq s_{max} \\ 0 & otherwise \end{cases} \end{bmatrix} \tag{6.22}$$

Based on this model we would like a controller that can line up with a sequence of waypoint segments. After finding the best segment the controller will adjust the sink rate to match the slope of the two selected waypoints $\mathbf{p}_{c,i+1}$and $\mathbf{p}_{c,i}$ and also control the sink to match the altitude of the waypoint segment. The forward speed v_x is set from the speed of the select point s_i. Two terms vary depending on the alignment with the waypoint segment: the strafe velocity v_y and the heading rate $\dot{\theta}$. If the heading difference between the desired heading θ_{des} and the current heading θ is too large only a heading change will be commanded. However if we have lined up the heading reasonably well two parts will mix together to reduce the cross track error and the heading error for the commanded heading rate. If the cross track error is large the vehicle will turn toward the segment to be able to faster reduce the cross track error. As the helicopter gets closer the absolute heading of the segment will dominate. The amount of mixing is determined in λ_{mix} which depends on the magnitude of the $|d_{ct}|$ cross track distance. The strafe velocity v_y is set to minimize the cross track error if the heading error is small enough.

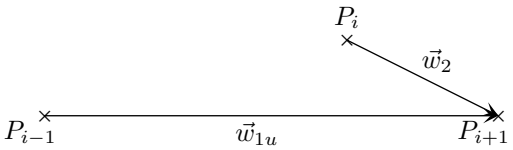
The input to the controller is a path $\mathbf{P_c} = \{\mathbf{p}_{c,0}, \ldots, \mathbf{p}_{c,i}, \ldots, \mathbf{p}_{c,n}\}$ and the current state $\mathbf{x}$. The output is the command $\mathbf{u}$. These are some short hand expressions to keep the overall controller compact:

$$i = \text{argmin}_{\forall i}|\mathbf{p}_{c,i} - \mathbf{p}|, \mathbf{v} = (\mathbf{p}_{\text{c,i+1}} - \mathbf{p}_{\text{c,i}})/|p - p_{\text{c,i}}|, \mathbf{v}_{ortho} = \begin{bmatrix} v_y \\ -v_x \end{bmatrix}/\sqrt{v_x^2 + v_y^2} \tag{6.23}$$

$$\theta_{des} = \arctan(v_y, v_x) \tag{6.24}$$

$$\lambda_{mix} = \min(1, \max(0, G_{mixd} - \frac{|d_{ct}|}{G_{mixd}}) \tag{6.25}$$

The overall waypoint controller can then be expressed as follows:



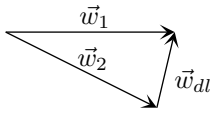


Figure 6.9: Illustration of the smoothness objective.

$$\mathbf{u}(\mathbf{x}, t, \mathbf{P_c}) = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} s_i \\ \begin{cases} limit(G_{ct} d_{ct}, s_{max,ct}) & \text{if} \angle(\theta_{des}, \theta) < \theta_{ct,max} \\ 0 & otherwise \end{cases} \\ G_z(P_{z,i} - z) + G_{z,fwd} \frac{(z_{i+1} - z_i)\sqrt{v_x^2 + v_y^2}}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \\ \begin{cases} (1 - \lambda_{mix})(-G_{y,ct} - v_y) + \lambda_{mix} G_{\dot{\theta},ct} \angle(\theta_{des}, \theta) & \text{if} \angle(\theta_{des}, \theta) < \theta_{max,ct} \\ G_{\dot{\theta},ct} \angle(\theta_{des}, \theta) \end{cases} \end{bmatrix} \tag{6.26}$$

This model enables us to simulate the motion of the vehicle for the trajectory optimization algorithm and also defines the range of motion for the initial guess algorithms.

Smoothness Objective A measure of trajectory energy ($\mathsf{energy}(\mathbf{P})$) is how much the trajectory is displaced from a straight line. Based on the displacement one can also calculate an energy gradient of the trajectory. The gradient is computed with respect to the straight line of the next point to preserve the global shape of the trajectory. Each segment is smoothed locally with respect to the next point. The energy function is defined as follows:

$$\vec{w}_{2u} = \mathbf{p}_{i+1} - \mathbf{p}_i, \vec{w}_2 = \frac{\vec{w}_{2u}}{|\vec{w}_{2u}|} \tag{6.27}$$

$$\vec{w}_{1u} = \mathbf{p}_{i+1} - \mathbf{p}_{i-1},\ \vec{w}_{1l} = \frac{\vec{w}_{1u}}{|\vec{w}_{1u}|}, \vec{w}_1 = |\vec{w}_2|\, \vec{w}_{1l} \tag{6.28}$$

$$\vec{w}_d = \vec{w_1} - \vec{w_2} \tag{6.29}$$

$$\mathsf{energy}(\mathbf{P}) = \min\left(1.0, \frac{d_{max}}{|\vec{w}_d|}\right) \vec{w}_d \tag{6.30}$$

Figure 6.9 illustrates how the terms are defined. The minimum energy is achieved when the trajectory is at a straight line.

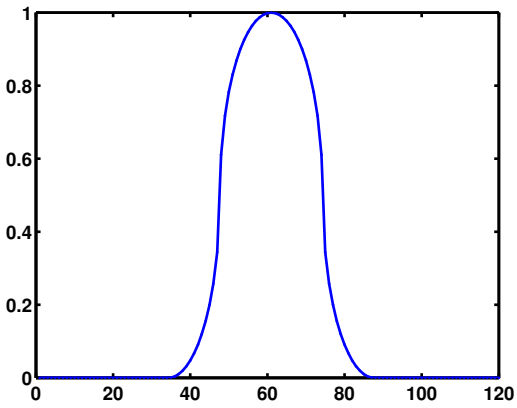


Figure 6.10: An illustration of the shape of the smoothing distribution function along one dimension. The function consists of three circles that smoothly rise from 0 to 1 and back down to 0.

Smoothing Matrix The objective function gradient is smoothed by distributing the update 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 [Ratliff et al., 2009]. 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 (See Fig. 6.10) 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:

$$\mathbf{W}(i,j) = \frac{1}{2} + \frac{r}{2} \begin{cases} \sqrt{d_{v1}} & d_{v1} \geq 0 \\ \sqrt{d_{v2}} & d_{v2} \geq 0 \wedge j \geq (i - 2r) \\ \sqrt{d_{v3}} & d_{v3} \geq 0 \wedge j \leq (i + 2r) \end{cases} \tag{6.31}$$

where

$$d_{v1} = r^2 - (j-i)^2,\ d_{v2} = r^2 - (j-i-2r)^2,\ d_{v1} = r^2 - (j-i+2r)^2. \tag{6.32}$$

6.2.5.3 Trajectory Optimization

The trajectory is discretized into 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 trajectory vector $\mathbf{P_c}$. 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 problem setup the non-linearity of the problem makes applying the Jacobian pseudo-inverse or transpose to the path mapping unstable in the optimization and we were able to achieve better results by directly updating the path.

Since we discretize the state as a set of 3-D points that are followed by the path controller $\mathbf{u}$ we need to calculate the gradient and cost across the segment to avoid skipping any important cost areas. Since our cost function domain is discretized we can check the gradients and cost in increments of the grid resolution. More formally, the gradient g of the objective function for a path segment is calculated as follows:

$$l_{s1} = |\mathbf{p}_{i-1} - \mathbf{p}_i|,\ l_{s2} = |\mathbf{p}_i - \mathbf{p}_{i+1}| \tag{6.33}$$

$$g(\mathbf{p}_i) = (l_{s1} \cdot \mathsf{gradLine}(\mathbf{p}_{i-1}, \mathbf{p}_i) + l_{s2} \cdot \mathsf{gradLine}(\mathbf{p}_i, \mathbf{p}_{i+1}))/(l_{s1} + l_{s2}) \tag{6.34}$$

The gradient is projected along the connecting line between the segments because the trajectory will change in length otherwise. This is undesirable because some segments will get contracted or can overlap. Furthermore, the gradient has to be computed along the line segment to allow a continuous update of the gradient.

$$\mathsf{gradLine}(\mathbf{p},\mathbf{q}) = \frac{1}{m}\sum_{i=0}^{m}\mathsf{gradProj}((\mathbf{p}-\mathbf{q})/|\mathbf{p}-\mathbf{q}|, \frac{i}{m}\mathbf{p} + (1-\frac{i}{m})\mathbf{q}) \tag{6.35}$$

$$\mathsf{gradProj}(\mathbf{d},\,\mathbf{g}) = \mathbf{g} - (\mathbf{d}\cdot\mathbf{g})\mathbf{d} \tag{6.36}$$

The new path $\mathbf{P}_c'$ is evaluated for the overall average trajectory cost and energy at each trajectory iteration. The cost is defined to enable computation over the grid resolution and for the line segments as

$$J(\mathbf{P_c}) = \frac{1}{n-1}\sum_{i=1}^{n}\mathsf{lineSegCost}(\mathbf{p}_{r,i-1}, \mathbf{p}_{r,i}) \tag{6.37}$$

$$\mathsf{lineSegCost}(\mathbf{p},\mathbf{q}) = \frac{1}{m}\sum_{i=0}^{m} c(\frac{i}{m}\mathbf{p} + (1-\frac{i}{m})\mathbf{q}) \tag{6.38}$$

The discretization of the parameter set into a set of line segments for the controller and discrete grid of the gradient and cost function allow efficient gradient descent on the trajectory.

6.2.5.4 Initial Guess Model

A good initial guess can be followed closely with the path controller. In the following we define two algorithms to create the initial guesses 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.

Simple Initial guess Since there is no sense of heading at the goal location $\mathbf{p}_{m,i}$ the initial guess just needs to connect the start to the goal location. The simplest algorithm that respects the turning radius constraint of the dynamic model is a one-sided Dubin's curve which is only consists of a circle and a straight line to the goal location without heading. The circle tangents are then defined as

$$\mathbf{t}_i = \mathbf{a}_i + R_{h,min}\begin{pmatrix}\cos\beta\\ \sin\beta\end{pmatrix} \tag{6.39}$$

$$\beta = s\cdot\arccos(\frac{r}{|\mathbf{x}-\mathbf{a}_i|}) + \angle(\mathbf{x}-\mathbf{c}_i) \tag{6.40}$$

$$\mathsf{simple}(\mathbf{x},\mathbf{p}) = \{circle(\mathbf{x},\mathbf{t}_i), line(\mathbf{t}_i,\mathbf{p})\} \tag{6.41}$$

where $\mathbf{a}_1, \mathbf{a}_2$ are the circle centers, $\mathbf{x}$ is the current position and $s = \{-1, 1\}$ are the sign of the tangents. The altitude component is interpolated between the start and end altitude. Since the climb rate of a helicopter is not constraint as it is the case for fixed-wing airplanes it is not necessary to add extra loops to the path to achieve a certain altitude.

Planning Graph Traditionally in 2D grid search an 8-connected graph 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. Instead we adopt the idea of [Hwangbo et al., 2007] 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 coarsely discretized and the allowable node expansions are set in conjunction with the cell resolution. The grid cell resolution is set to not violate the turning radius and climb rate constraint as

$$S_x = S_y = \frac{2}{3} R_{h,min},\ S_z = S_x \tan^{-1}(\mu_{max}) \tag{6.42}$$

where μ_{max} is the maximum climb angle. The parent expansion in the search graph prevents the planner from considering the same (x, y, z) location twice and do not allow looping back. However, since we also can move in the z-direction small changes in altitude allow us to produce overlapping routes.

6.3 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 three different objectives. The first experiment shows avoiding an unknown obstacle, the next experiment searches for landing sites, and finally we show results from planning a path that avoids obstacles on approach.

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 (See Sec. 6.2.5.2 for the dynamic model).

6.3.1 Reach a Goal Point

The first experiment uses the obstacle avoidance objective (See Fig. 6.6) 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. 6.11a 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 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.

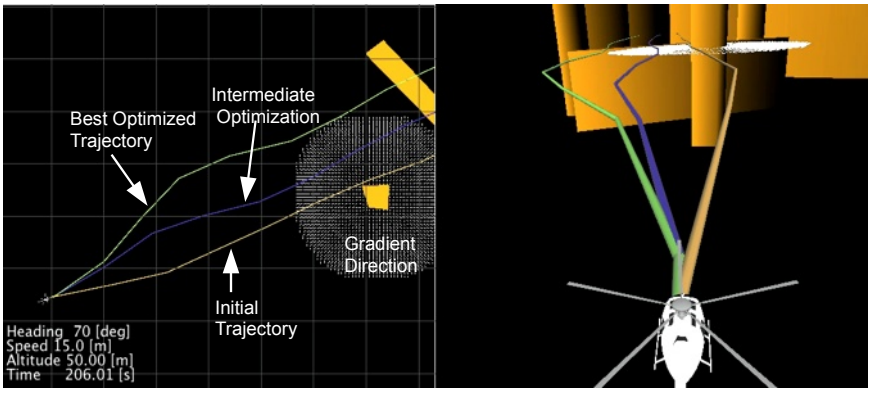
In Fig. 6.13 and Extension 8 one can see how the obstacle cost and energy cost evolve as the algorithm optimizes the trajectory. Initially since the trajectory intersects with an obstacle the cost is maximal. As the shape of the trajectory changes the cost decreases towards the minimum. The energy of 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. Figure 6.11b shows an example of the optimization changing the altitude of the trajectory to avoid the obstacle.

Next we enable the coarse initial guess planner which will avoid obstacles. However as shown in Fig. 6.11c 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.

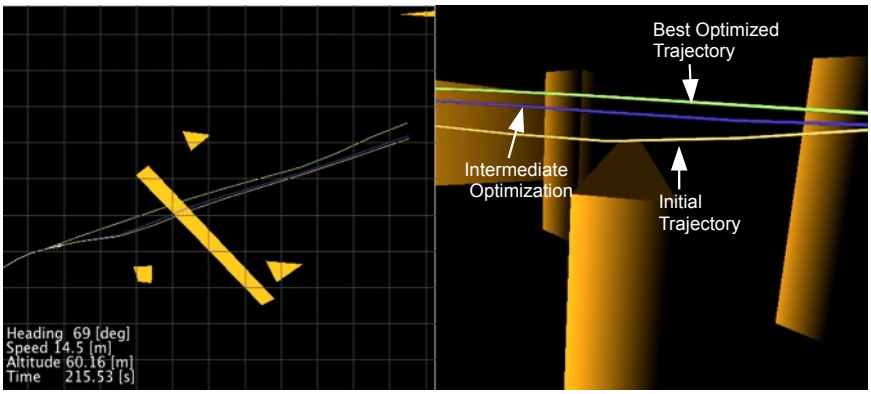
Since we cannot guarantee that the optimization will converge to a solution we want to optimize a set of initial guesses (Fig. 6.12a and Extension 9). One guess is always a simple guess that moves through an obstacle while the other initial guesses avoid the larger obstacles in the center either to the left or right. One can see that the alternative route planning algorithm picked a set of significantly different trajectories that can be optimized (Fig. 6.12b). If there are no better alternatives only one will be found as shown in Fig. 6.12c where only one planned and one simple initial guess were found.

6.3.2 Search for Landing Sites

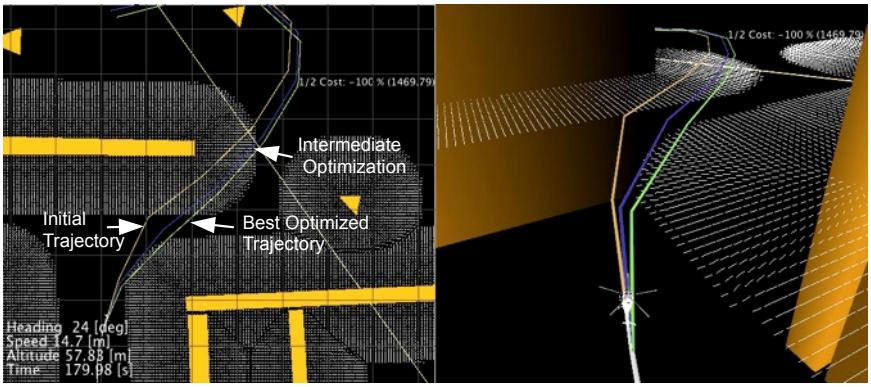
The problem considered now is the landing site search problem (Fig. 6.7) and approach planning problem (Fig. 6.8) where the planning algorithm needs to find a set of landing sites and approach the sites. The landing sites have to be reachable by a ground goal.



(a)

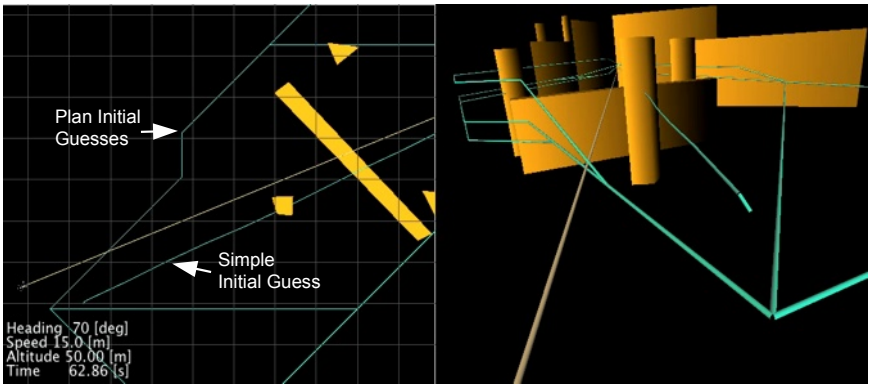


(b)

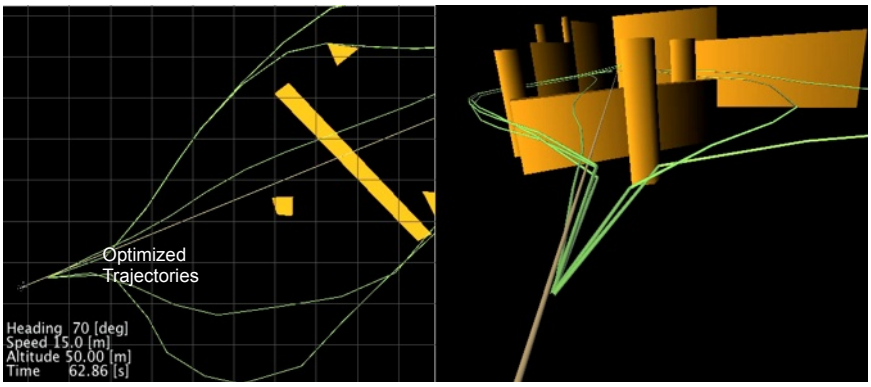


(c)

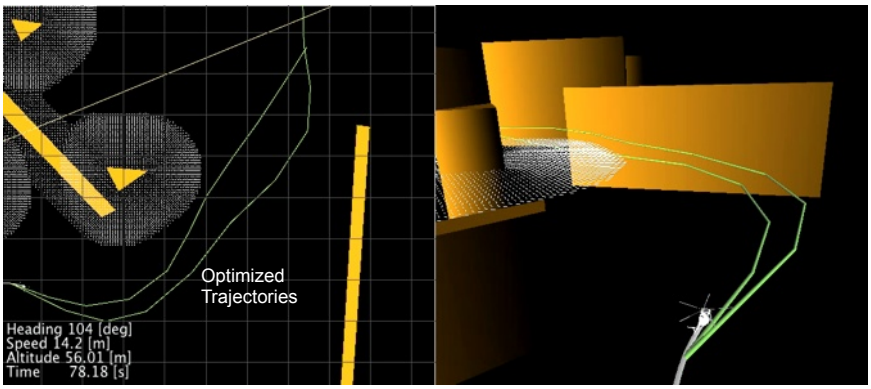
Figure 6.11: Examples of the trajectory optimization algorithm avoiding obstacles. In (a) the simple initial guess trajectory does not use the coarse planner and therefore only the optimization moves the trajectory to avoid collision. In (b) the optimization avoids collision with a vertical obstacle. In (c) the coarse planner is used to plan an initial guess, however the path gets close to some obstacles. After optimization the clearance is larger. The segments are discretized in 16 meter control intervals. (Orange = obstacles, green = best trajectory, orange = initial guess. purple = intermediate optimization step.)



(a)



(b)



(c)

Figure 6.12: Using the planned initial guess for trajectory optimization. (a) shows the simple and plan initial guesses that are used to find a set of optimized trajectories. (b) 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 (c) only the simple and one planned initial guess are optimized.

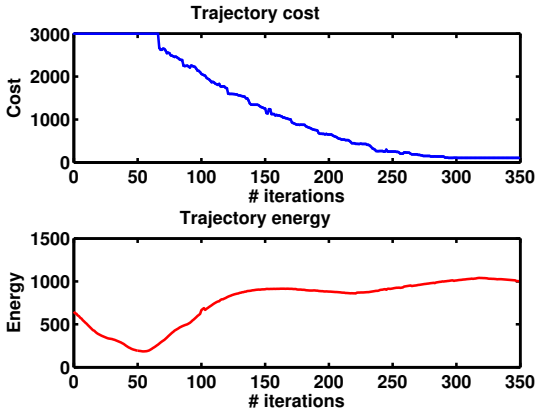


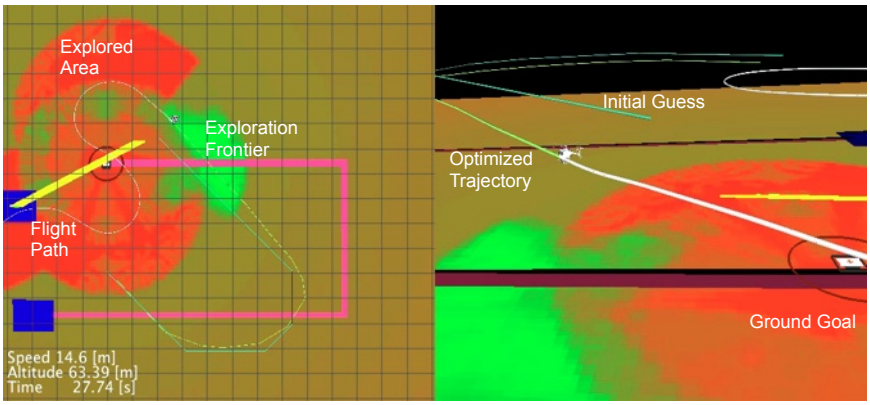
Figure 6.13: Cost and energy per iteration for one planning cycle. The initial guess is moved outside the obstacle during the optimization since the simple initial guess path intersected an obstacle. The optimization could be interrupted after the obstacle is cleared. The setup is shown in Fig. 6.11a.

In the next example no obstacles are present and the vehicle has to explore the environment to find a ground path to the goal location and find a set of reachable good landing sites (Also see Extension 10). In Fig. 6.14a the vehicle has explored some of the environment and has found some landing sites. However, the landing sites are not reachable and therefore the search continues. One can see that the trajectory is optimized to explore more of the terrain since it is pulled towards areas with higher information gain (green). In Fig. 6.14b the search is almost complete and the current minimum is close to the vehicle. If there was no action prediction the trajectory that was optimized would be too short to fly at the desired speed. Utilizing the action prediction allows optimizing a trajectory that is longer because the information gain prediction removed the close minimum after the first search step.

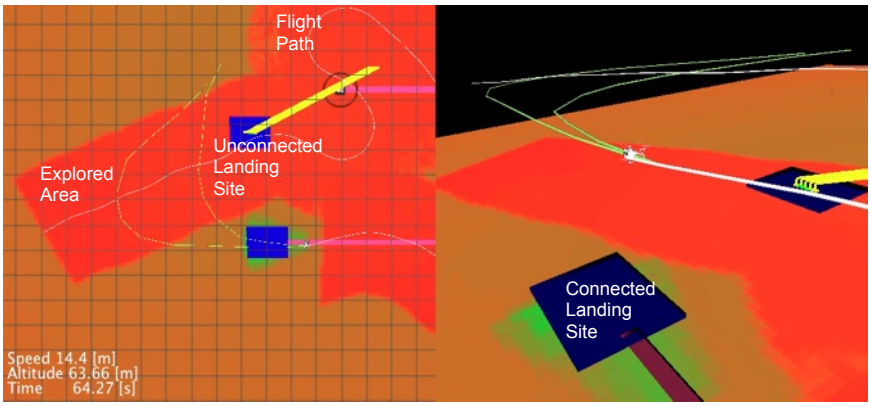
The information gain prediction increases the planning horizon and consequently the trajectory can be optimized over a longer distance. Figure 6.15 shows a sequence of a search (depth=2) with the action prediction to show the influence of the information gain prediction on planning. The benefit of the information gain prediction is that it increases the initial guess trajectory length. Since the trajectory length is longer the optimization has a larger trajectory space to optimize the path and the vehicle can fly faster since it does not need to come to a stop at the end of the trajectory. In Fig. 6.15a one can see the information gain before the action prediction. As the action is predicted the original minimum is removed and a new location becomes minimal. The plan to this new minimum is shown in Fig. 6.15c and then the action is predicted again in Fig. 6.15d.

Figure 6.16 and Extension 11 show an example of landing site search with obstacles that need to be avoided and that occlude the information that can be gained about landing sites. Initially the robot can follow the frontier of the information gain to explore along the path. However as it discovers new obstacles it needs to avoid the obstacles along the path either by lateral or climbing motion. In Fig. 6.16c a pole obstacle is in the approach path. The trajectory optimization moves the trajectory to enable the vehicle to approach using this path. The cost function for planning is shown in Fig. 6.8.

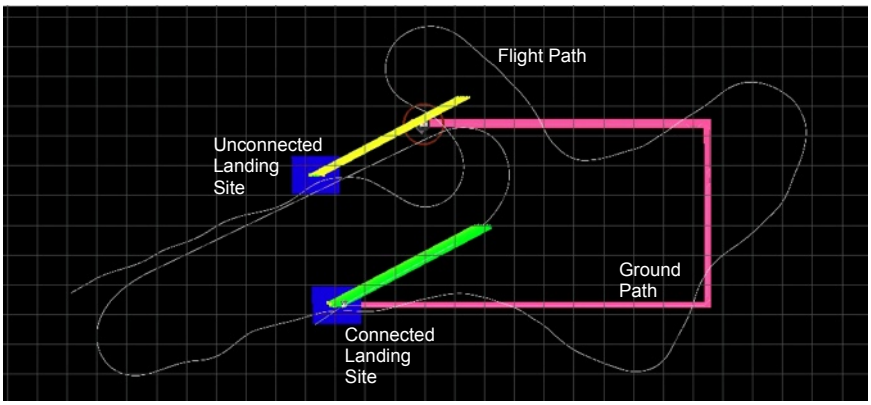
A challenging example for landing site search is shown in Fig. 6.17 and Extension 12. The vehicle has to find a good landing site and is only given the final ground goal shown in Fig. 6.17a and false prior knowledge of potential sites in dark purple. The obstacles, ground path and good landing zone are 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)



(b)



(c)

Figure 6.14: Simple landing site search. (a) shows the problem set up and a snapshot of the algorithm during the search. (b) shows progress just before a connected landing site is found. (c) shows the resulting search and approach path and discovered landing sites. Red= high cost, green = low cost. Discovered landing sites and approach paths are shown as yellow and green lines.

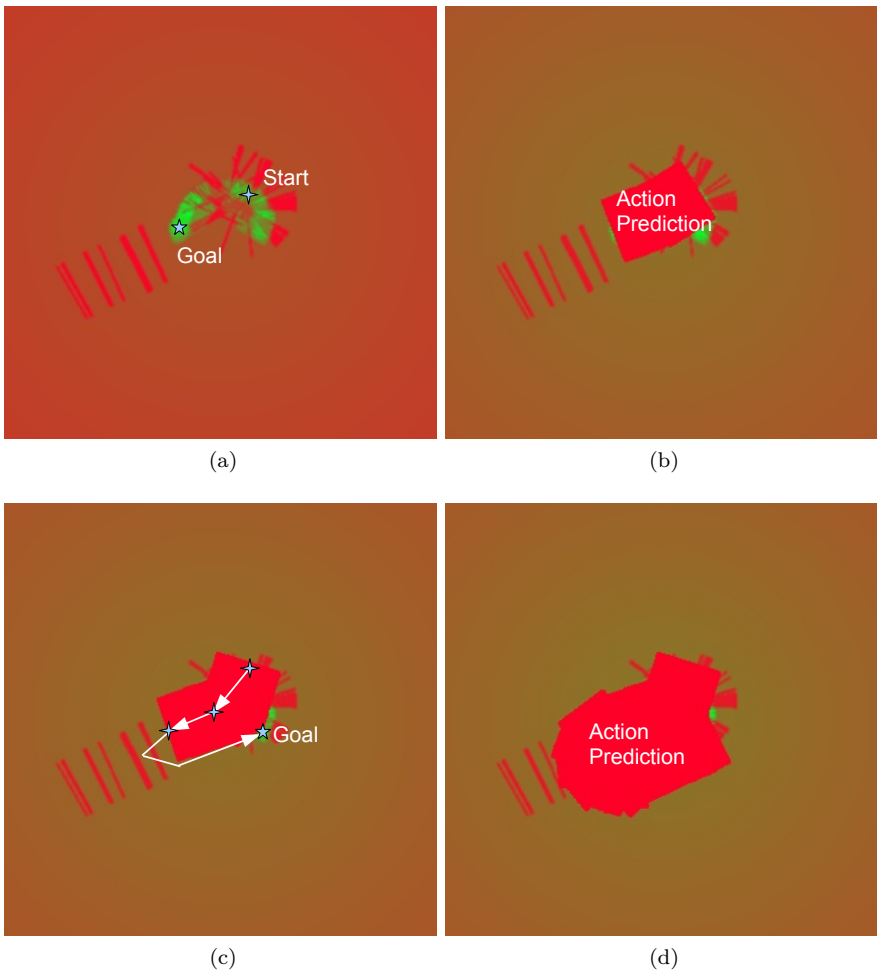
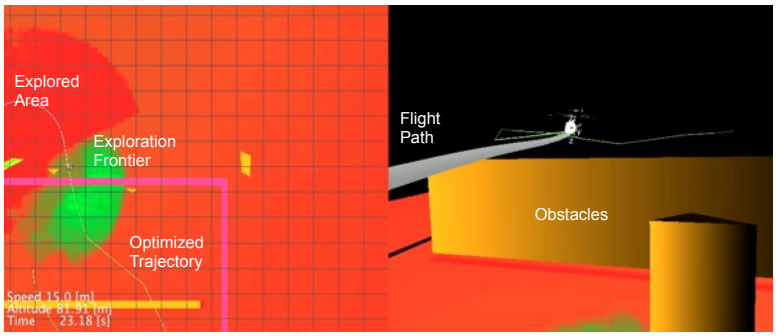
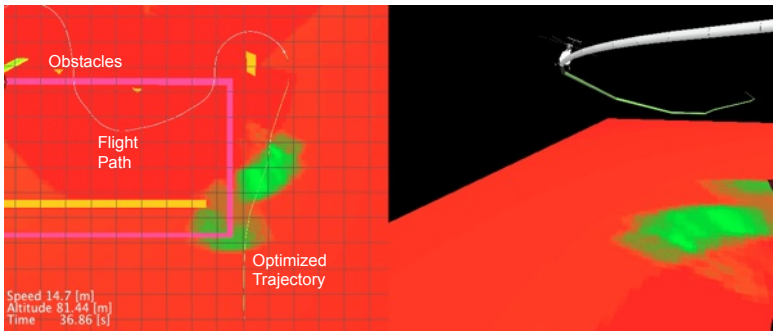


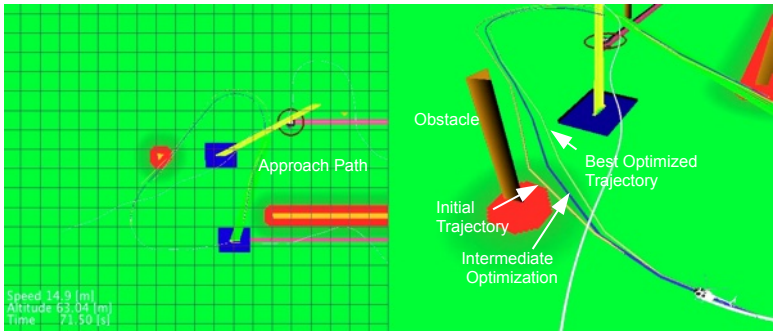
Figure 6.15: Action prediction example. This figure demonstrates the influence of the information gain prediction on the path planning horizon. In (a) the first minimum is found and a path is created. (b) shows the predicted information gain based on this path. In (c) a path to a new minimum is found and (d) shows the resulting combined action prediction. Red=low information gain, green= high information gain.



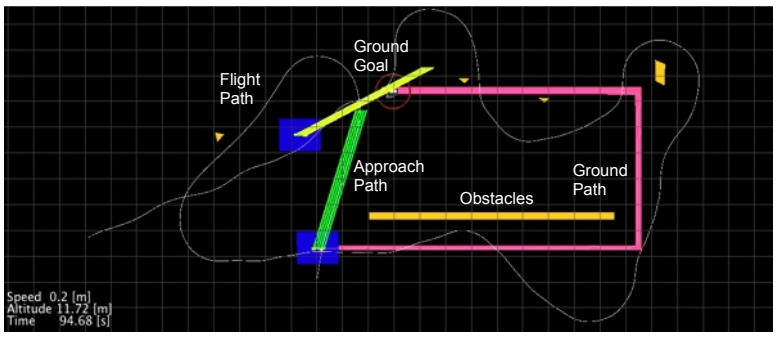
(a)



(b)



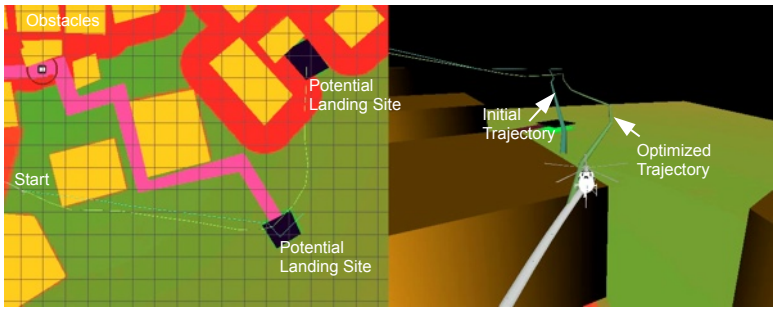
(c)



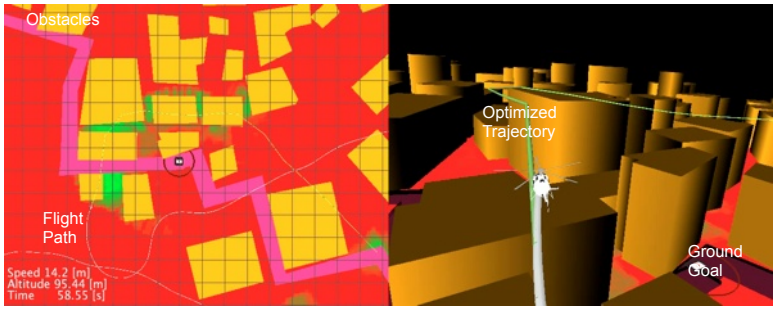
(d)

Figure 6.16: Landing site search with obstacles. (a) shows the algorithm avoiding obstacles in its flight path while search for landing sites. (b) is a later snapshot of the exploration. The vehicle is following the ground path frontier to find landing sites. In (c) one can see an example of avoiding an obstacle in the approach path. (d) shows the complete search and approach path taken by the vehicle.

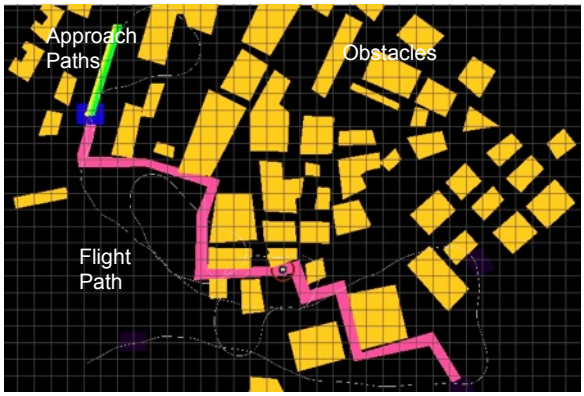
(a)



(b)

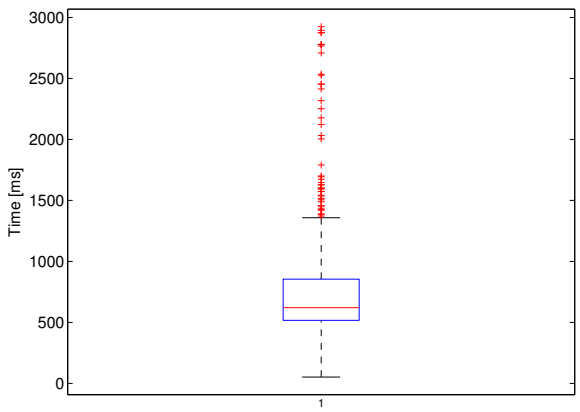


(c)

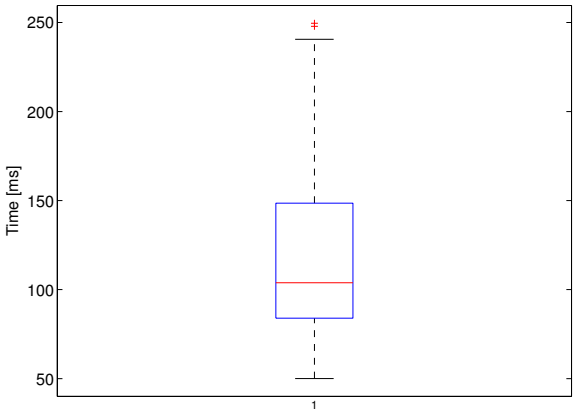


(d)

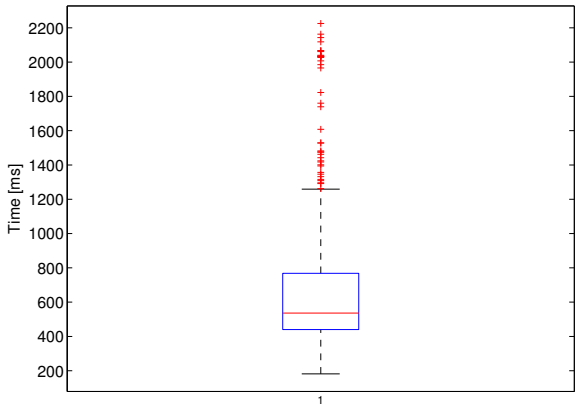
Figure 6.17: Manhattan environment landing site search. (a) shows the problem setup. A set of buildings similar to Manhattan presents obstacles and occludes the range sensor. (b) demonstrates the influence of prior knowledge on the trajectory. (c) demonstrates the occlusions. (d) shows the overall search and approach path.



(a) Complete algorithm calculation time.



(b) Trajectory optimization calculation time.



(c) Initial guess planner calculation time.

Figure 6.18: Calculation time and optimization cost for landing site search. Box and whisker plot legend for Fig. (a)-(c): The red line is the median, the blue box extends from the lower quartile to the upper quartile, and the whiskers extend to 1.5 of the interquartile range. Red crosses represent single run outliers.

Figure 6.17b shows how the information gain prediction increases the planning horizon. The vehicle is at the start location and has a minimum at the potential landing site since we gave potential landing sites as prior information. After reaching this minimum the action prediction will decrease the amount of information gain on the path up to the minimum and then look for the next minimum which is another area that is prior knowledge for a potential landing site.

A snapshot during exploration is visible in Fig. 6.17c. 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. 6.17d).

The calculation times for landing zone search are presented in Fig. 6.18 as box and whisker plots and Tab. 6.3. 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

Initial Guess		Optimization		Overall
0.53(±0.37)		0.10(±0.04)		0.62(±0.48)
Minima	Planner	Gradient	Dynamics	
0.20(±0.06)	0.33(±0.33)	0.02(±0.02)	0.08(±0.02)	

Table 6.3: Median and standard deviation computation time for landing site search in seconds.

optimized to perform the optimization more frequently while the coarse initial guess planner runs less frequently to reduce reaction time.

6.4 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.

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 parametrizations to test the generalization of the approach. We intend to formally define the set of admissible 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.

7 Conclusions and Future Directions

This thesis presented several approaches, and results of unmanned aerial vehicles operating in previously unknown environments with changing objectives. The reactive obstacle avoidance algorithm enables fast reaction to unknown obstacles for small aerial vehicles, while we remove a bottle neck for planning with the efficient C-space transform algorithm. Another problem we address is how to evaluate landing sites for full scale helicopters efficiently and with high fidelity. Since finding landing sites is a challenging problem, we developed an algorithm that generalizes to missions defined as an objective function and enables searching for landing sites. Next we summarize the approaches, review the contributions, and convey future research directions.

7.1 Summary

Obstacle Avoidance Our approach to obstacle avoidance for small UAV rotorcraft separates global planning from local planning and can therefore achieve the reaction time of a simple control law with the intelligence of a more deliberative approach. Our reactive approach addresses the issues of tuning algorithms to the desired behavior by learning the parameters based on piloted training examples and addresses safety with the use of a speed controller.

In over 1000 runs at obstacles, the uninhabited helicopter started with no prior knowledge of the environment and the straight line path between the waypoints often intersected obstacles. While we regularly ran collision avoidance close to buildings, trees and wires between 4-6 m/s, the system was tested at commanded speeds up to 10 m/s.

Efficient Calculation of the Obstacle Cost Function One bottleneck in planning paths in a grid map for aerial vehicles is the large expansion of obstacles required for safe routes away from obstacles. We have presented a completely incremental framework for planning paths in 3D that enables recomputation of obstacle costs an order of magnitude faster than current approaches. This speed up is made possible by using a novel limited incremental distance transform algorithm.

The algorithm exploits the local nature of cost function updates when obstacles are added or removed from the map and enables autonomous aerial vehicles to respond to newly observed obstacles (or obstacles that no longer exist) in real-time. We have provided results from simulation comparing various algorithms, and results from a quad-rotor micro aerial vehicle autonomously navigating in Pittsburgh, PA and an unmanned helicopter avoiding obstacles in Mesa, AZ.

Landing Site Evaluation We presented, to our knowledge, the first geometric model-based algorithm that gives estimates of available landing sites for VTOL aerial vehicles. In our algorithm we are able to incorporate many constraints that have to be fulfilled for a helicopter to be able to land. While the current state of the art is similar to our coarse evaluation algorithm, the fine evaluation allows us to consider more aerial-vehicle relevant constraints. Additionally, we include constraints based on the approach and ground paths to determine the feasibility of a landing sites. We also presented results based on real sensor data for landing site evaluation at vegetated and urban sites and demonstrated the first autonomous man-scale helicopter that is able to select its own landing sites and land in a casualty evacuation scenario.

Multiple-Objective Motion Planning For missions that require finding dynamically feasible routes for unknown goal points on real helicopters we presented a framework and algorithms that enables optimizing multiple objectives in motion planning. The algorithms approximate the problem by exploiting the additional information in the augmented cost functions to make the problem feasible.

The simulation results demonstrate the effectiveness of our algorithm for avoiding obstacles, performing landing zone search, and approaching a landing site.

7.2 Contributions

The following contributions have been presented in this thesis in the areas of perception, and planning for unmanned aerial vehicles operating at low-altitude:

1. A fast reactive obstacle avoidance algorithm for small unmanned helicopters and demonstration of the fastest (10 m/s) autonomous obstacle avoidance for small unmanned helicopters.

2. An efficient algorithm for obstacle cost function updates, and results demonstrating the effectiveness in simulation and a demonstration of the first full-size helicopter autonomously avoiding obstacles utilizing this algorithm.

3. A model-based landing site evaluation algorithm for unmanned helicopters that includes many of the constraints necessary for a successful landing. A demonstration of the first completely autonomous full-size helicopter performing autonomous landing site selection and landing.

4. An analysis of the problem of planning and acting for flexible missions at low-altitude for unmanned aerial vehicles and an algorithm for planning with multiple objective functions that is evaluated in simulation.

7.3 Future Directions

The area of operating autonomous aerial vehicles at low altitude is novel and there are many directions for future research. First, I will describe some new directions for future research and after that conclude with some specific enhancements to the current approaches.

7.3.1 New Research Topics

This thesis only considers a subset of the needed capability for a truly intelligent autonomous aerial vehicle. More advances in perception, planning, and verification are still needed to make intelligent aerial vehicles a reality. Next I describe some new research topics that need to be addressed:

Current mapping data structures are sufficient for small missions. However more efficient representations and algorithms are required to make plans in 3D for 100's of kilometers at low-altitude a possibility.

Planning algorithms need to be able to cope with emergencies such an engine failure. The landing zone evaluation can help in planning by guiding a pilot, however the planning algorithms need to be extended to account for the flight envelope by reacting to the change in capability.

Turbulence and wind are a large factor in effectively controlling an aircraft and pilots will read the environment to predict bad areas to be close too on the ground. To be at least as effective when navigating in turbulent environments a map of the turbulence has to be created and used to derive a cost for planning.

So far we have assumed that we are operating in a static environment. In future work, it will be necessary to account for other aerial vehicles during flight and other ground vehicles and people around landing sites. People or other vehicles can temporarily foul landing sites but can move to clear a site.

For commercial applications it must be possible to prove that the overall behavior of the system is safe and correct. There are some first steps in software model checking to verify the behavior such as in [Scherer et al., 2005] however it is difficult to scale to larger systems and to include the uncertainties of motion, and sensing in the verification.

So far we have assumed that a good pose sensor is available however especially for small vehicles with MEMS IMUs, integrated INS/GPS is not accurate enough at low altitude since occlusions, and

GPS multi-path cause jumps and outages. Furthermore, landing zone evaluation at distance in front of the aircraft is challenging because the algorithms are sensitive to misregistered points.

Active range sensors available today are too heavy for micro aerial vehicles to operate outdoors, and are too expensive to achieve the range required for full performance operation on manned helicopters at a viable cost. Novel sensors will be necessary to make sensing accessible to these platforms.

Finally, so far we have considered complete autonomy, however incorporating the human in an effective way in an autonomous VTOL (classifying landing site bearability, shared motion planning) is still an open problem.

7.3.2 Algorithmic Improvements

There are several future directions for research on the four problems that were addressed in this thesis.

The reactive obstacle avoidance algorithm is an efficient approach for smaller vehicles and responds fast to new situations. However currently the response to obstacles depends on the distance to obstacles instead of the time to collision. Also to guarantee that no collision is possible currently we are limited by stopping distance and in future work it needs to be shown that the behavior is intrinsically safe. Since the ground plane also is an obstacle we introduced a box constraint to be able to stay low. In future work, it would be desirable to have a more principled approach to stay low to the ground. Other goal interfaces to the reactive algorithm that are not goal positions from the global planner could be examined to combine the two algorithms.

The LIDT algorithm is efficient enough to update the obstacle cost function in real-time however in future work some efficiency can be gained by avoiding some expansions that are only required for certain distance metrics and the correctness of the algorithm also needs to be proven formally.

Landing zone evaluation for helicopters is a new problem and some generalizations and improvements to the current algorithm are possible. The weight vector that determines the cost of landing zones is currently set by hand. A more elegant method would be to use expert input to learn the weight vector. There are currently some false negative (grass) and some false positive (water) landing sites that cannot be distinguished with purely geometrical data. Some other sensor modality such as interpreting camera data will be required to improve the landing site evaluation in these environments.

The multi-objective motion planning algorithm currently has only been tested in simulation and results on a real platform will be necessary to demonstrate the practicality of the approach. Some work remains in improving the efficiency of the approach to increase the space of initial guesses that can be tested.

Sensor, computing and vehicle platform advances in recent years, as well as a new interest in bringing intelligent autonomous rotorcraft closer to earth has enabled the advances in this thesis. This thesis addressed a subset of the many exciting new problems that still need to be addressed in aerial robotics. I am excited that this research area will eventually lead to safer and more capable manned aircraft and intelligent autonomous aerial vehicles.

Appendix: Multimedia Extensions

Extensions	Description	Link
1	Obstacle avoidance on the RMax helicopter.	http://www.youtube.com/watch?v=8iYi_b1pQdI
2	Comparison of the effect of setting a soft ceiling for the planning algorithm.	http://www.youtube.com/watch?v=n6S8iD5qFJg
3	Flight of the quad-rotor vehicle avoiding obstacles.	http://www.youtube.com/watch?v=CLWhRtLSPAk
4	Obstacles avoidance on the Unmanned Little Bird Helicopter.	http://www.youtube.com/watch?v=fQ0XcCNuhsM
5	Landing site evaluation overflying a power plant.	http://www.youtube.com/watch?v=IXbBNyt6tz4
6	Landing site evaluation and landing of the Unmanned Little Bird Helicopter.	http://www.youtube.com/watch?v=BJ3RhXjucsE
7	Example information gain map created from the overflight in extension 6.	http://www.youtube.com/watch?v=Y5vIujepfPQ
8	Avoiding obstacles based on the trajectory optimization.	http://www.youtube.com/watch?v=G5AfpsGyP5E
9	Planning a path to avoid obstacles.	http://www.youtube.com/watch?v=snqHinG6LB8
10	Simple landing zone search without obstacles.	http://www.youtube.com/watch?v=oRMNGAeQMq0
11	Landing zone search wit obstacles.	http://www.youtube.com/watch?v=BJH7N0Fc1Q0
12	Manhattan landing zone seach.	http://www.youtube.com/watch?v=IflAI2W20eg

Table 7.1: Table of multimedia extensions.

Bibliography

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

[Amidi, 1996] Amidi, O. (1996). *An autonomous vision-guided helicopter*. PhD thesis, The Robotics Institute, Pittsburgh, PA.

[Amidi et al., 1998] Amidi, O., Kanade, T., and Miller, J. (1998). Vision-based autonomous helicopter research at carnegie mellon robotics institute 1991-1997. In *American Helicopter Society International Conference*.

[Andert et al., 2010] Andert, F., Adolf, F.-M., Goormann, L., and Dittrich, J. S. (2010). Autonomous Vision-Based Helicopter Flights Through Obstacle Gates. *Journal Of Intelligent & Robotic Systems*, 57(1-4):259–280.

[Andert and Goormann, 2007] Andert, F. and Goormann, L. (2007). Combined grid and feature-based occupancy map building in large outdoor environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2065–2070.

[Barber et al., 2005] Barber, D., Griffiths, S., McLain, T., and Beard, R. W. (2005). Autonomous Landing of Miniature Aerial Vehicles. In *Proceedings of the AIAA Infotech@Aerospace Conference*.

[Beyeler et al., 2006] Beyeler, A., Mattiussi, C., Zufferey, J.-C., and Floreano, D. (2006). Vision-based altitude and pitch estimation for ultra-light indoor microflyers. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2836–2841.

[Beyeler et al., 2007] Beyeler, A., Zufferey, J., and Floreano, D. (2007). 3D Vision-based Navigation for Indoor Microflyers. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1336–1341.

[Boeing, 2010] Boeing. http://www.boeing.com/rotorcraft/military/ulb [online]. (2010).

[Bosch et al., 2006] Bosch, S., Lacroix, S., and Caballero, F. (2006). Autonomous Detection of Safe Landing Areas for an UAV from Monocular Images. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5522–5527.

[Bouabdallah, 2007] Bouabdallah, S. (2007). *Design and control of quadrotors with application to autonomous flying*. PhD thesis, EPFL, Lausanne, Switzerland.

[Bresenham, 1965] Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30.

[Brock and Khatib, 2002] Brock, O. and Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 21(12):1031–1052.

[Brooker et al., 2003] Brooker, G., Sukkarieh, S., and Durrant-Whyte, H. (2003). Decentralised ground target tracking with heterogeneous sensing nodes on multiple UAVs. *Lecture Notes in Computer Science*.

[Byrne et al., 2006] Byrne, J., Cosgrove, M., and Mehra, R. (2006). Stereo based obstacle detection for an unmanned air vehicle. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2830–2835.

[Choset et al., 2005] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). Principles of Robot Motion: Theory, Algorithms, and Implementations. The MIT Press.

[Connolly et al., 1990] Connolly, C. I., Burns, J., and Weiss, R. (1990). Path planning using Laplace's equation. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 2102–2106.

[de Wagter and Mulder, 2005] de Wagter, C. and Mulder, J. (2005). Towards Vision-Based UAV Situation Awareness. In *Proceedings of the AIAA Conference on Guidance, Navigation, and Control (GNC)*.

[Deal and Huang, 2008] Deal, C. and Huang, P. (2008). Remote Control Hovering Ornithopter. In *46 th AIAA Aerospace Sciences Meeting and Exhibit*.

[Fabbri et al., 2008] Fabbri, R., Costa, L., Torelli, J., and Bruno, O. (2008). 2D Euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys (CSUR)*, 40(1).

[Fabri et al., 1996] Fabri, A., Giezeman, G., Kettner, L., and Schirra, S. (1996). The CGAL kernel: A basis for geometric computation. *Lecture Notes in Computer Science*.

[Fajen and Warren, 2003] Fajen, B. R. and Warren, W. H. (2003). Behavioral Dynamics of Steering, Obstacle Avoidance, and Route Selection. *Journal of Experimental Psychology: Human Perception and Performance*, 29(2):343–362.

[Ferguson and Stentz, 2006] Ferguson, D. and Stentz, A. (2006). Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2):79–101.

[Foessel, 2002] Foessel, A. (2002). *Scene Modeling from Motion-Free Radar Sensing*. PhD thesis, The Robotics Institute, CMU, Pittsburgh, PA.

[Frazzoli et al., 2002] Frazzoli, E., Dahleh, M., and Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance Control and Dynamics*, 25(1):116–129.

[Frazzoli et al., 2005] Frazzoli, E., Dahleh, M. A., and Feron, E. (2005). Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics and Automation*, 21(6):1077–1091.

[Frew and Elston, 2008] Frew, E. and Elston, J. (2008). Target assignment for integrated search and tracking by active robot networks. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2354–2359.

[Gonzalez and Stentz, 2008] Gonzalez, J. and Stentz, A. (2008). Replanning with uncertainty in position: Sensor updates vs. prior map updates. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1806–1813.

[Green and Oh, 2008] Green, W. and Oh, P. (2008). Optic-Flow-Based Collision Avoidance. *Robotics & Automation Magazine, IEEE*, 15(1):96–103.

[Griffiths et al., 2006] Griffiths, S., Saunders, J., Curtis, A., and McLain, T. (2006). Obstacle and Terrain Avoidance for Miniature Aerial Vehicles. *IEEE Robotics and Automation Magazine*.

[Grocholsky, 2002] Grocholsky, B. (2002). *Information-theoretic control of multiple sensor platforms*. PhD thesis, University of Sydney.

[Grzonka et al., 2009] Grzonka, S., Grisetti, G., and Burgard, W. (2009). Towards a Navigation System for Autonomous Indoor Flying. In *Proceedings IEEE International Conference on Robotics and Automation*.

[Hamner et al., 2006] Hamner, B., Singh, S., and Scherer, S. (2006). Learning obstacle avoidance parameters from operator behavior. *Journal of Field Robotics*, 23(11/12):1037–1058.

[Harris et al., 2000] Harris, F., Kasper, E., and Iseler, L. (2000). US Civil Rotorcraft Accidents, 1963 through 1997. Technical Report NASA/TM-2000-209597, NASA, NASA.

[He et al., 2008] He, R., Prentice, S., and Roy, N. (2008). Planning in Information Space for a Quadrotor Helicopter in a GPS-denied Environment. In *Proceedings IEEE International Conference on Robotics and Automation*.

[Hebert and Vandapel, 2003] Hebert, M. and Vandapel, N. (2003). Terrain classification techniques from ladar data for autonomous navigation. In *Proceedings of the Collaborative Technology Alliances Conference*.

[Henzinger, 1996] Henzinger, T. A. (1996). The Theory of Hybrid Automata. In *Proceedings of the 11th Annual Sysmposium on Locgic in Computer Science*.

[Hintze, 2004] Hintze, J. (2004). Autonomous Landing of a Rotary Unmanned Aerial Vehicle in a Non-Cooperative Environment using Machine Vision. *Master's Thesis (Brigham Young University)*.

[Howard and Kelly, 2007] Howard, T. M. and Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26(2):141–166.

[Hrabar and Gaurav, 2009] Hrabar, S. and Gaurav, S. (2009). Vision-based navigation through urban canyons. *Journal of Field Robotics*, 26(5):431–452. (Preprint).

[Hwangbo et al., 2007] Hwangbo, M., Kuffner, J., and Kanade, T. (2007). Efficient Two-phase 3D Motion Planning for Small Fixed-wing UAVs. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1035–1041, Rome.

[Jackson et al., 2005] Jackson, Sharma, Haissig, and Elgersma (2005). Airborne Technology for Distributed Air Traffic Management. *European Journal of Control*, 11(4-5).

[Johnson et al., 2005] Johnson, A., Montgomery, J., and Matthies, L. (2005). Vision Guided Landing of an Autonomous Helicopter in Hazardous Terrain. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 3966–3971.

[Joon Ahn, 2004] Joon Ahn, S. (2004). Least squares orthogonal distance fitting of curves and surfaces in space. *Lecture Notes in Computer Science*.

[Kalra et al., 2006] Kalra, N., Ferguson, D., and Stentz, A. (2006). Incremental Reconstruction of Generalized Voronoi Diagrams on Grids. In *Proc. of the International Conference on Intelligent Autonomous Systems*.

[Kanade et al., 2004] Kanade, T., Amidi, O., and Ke, Q. (2004). Real-time and 3D vision for autonomous small and micro air vehicles. *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, 2:1655–1662 Vol.2.

[Kavraki et al., 1996] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

[Khatib, 1986] Khatib, O. (1986). Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5(1):90–98.

[Kitamura et al., 1996] Kitamura, Y., Tanaka, T., Kishino, F., and Yachida, M. (1996). Real-time path planning in a dynamic 3-d environment. In *Intelligent Robots and Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, pages 925–931.

[Koenig and Likhachev, 2002a] Koenig, S. and Likhachev, M. (2002a). D* Lite. In *Eighteenth national conference on Artificial intelligence*.

[Koenig and Likhachev, 2002b] Koenig, S. and Likhachev, M. (2002b). Improved fast replanning for robot navigation in unknown terrain. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 968–975 vol.1.

[Kolter and Ng, 2009] Kolter, J. and Ng, A. Y. (2009). Task-Space Trajectories via Cubic Spline Optimization. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1675–1682.

[Kuwata, 2007] Kuwata, Y. (2007). *Trajectory Planning for Unmanned Vehicles using Robust Receding Horizon Control*. PhD thesis, MIT, Boston, MA.

[Lewis et al., 1993] Lewis, M., Fagg, A., and Bekey, G. (1993). The USC autonomous flying vehicle: an experiment in real-time behavior-based control. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 422–429 vol.2.

[Li and Bui, 1998] Li, Z. X. and Bui, T. D. (1998). Robot Path Planning Using Fluid Model. *Journal Of Intelligent & Robotic Systems*, 21:29–50.

[Liu and Cheng, 2002] Liu, X.-W. and Cheng, K. (2002). Three-dimensional extension of Bresenham's algorithm and its application in straight-line interpolation. *Journal of Engineering Manufacture*, 216(3):459–463.

[Martin and Moravec, 1996] Martin, M. C. and Moravec, H. (1996). Robot Evidence Grids. Technical Report CMU-RI-TR-96-06, The Robotics Institute, Carnegie Mellon, Pittsburgh, PA.

[Meijster et al., 2000] Meijster, A., Roerdink, J., and Hesselink, W. (2000). A general algorithm for computing distance transforms in linear time. *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 331–340.

[Mejias et al., 2006] Mejias, L., Campoy, P., Usher, K., Roberts, J., and Corke, P. (2006). Two Seconds to Touchdown - Vision-Based Controlled Forced Landing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3527–3532.

[Merrell et al., 2004] Merrell, P. C., Lee, D.-J., and Beard, R. W. (2004). Obstacle avoidance for unmanned air vehicles using optical flow probability distributions. *Mobile Robots XVII*, 5609(1):13–22.

[Michel, 2008] Michel, P. (2008). *Integrating Perception and Planning for Humanoid Autonomy*. PhD thesis, The Robotics Institute.

[Michelson, 2000] Michelson, R. C. (2000). The International Aerial Robotics Competition - a Decade of Excellence. *Unmanned Vehicles (UV) for Aerial, Ground and Naval Military Operations, NATO Research and Technology Organization Proceedings 52, Applied Vehicle Technology Panel (AVT), Ankara, Turkey*, pages SC–1 to SC–24.

[Nabbe, 2005] Nabbe, B. (2005). *Extending the Path-Planning Horizon*. PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[Nabbe and Hebert, 2007] Nabbe, B. and Hebert, M. (2007). Extending the path-planning horizon. *International Journal Of Robotics Research*, 26(10):997–1024.

[Niederreiter, 1988] Niederreiter, H. (1988). Low-discrepancy and low-dispersion sequences. *Journal of Number Theory*, 30(1):51–70.

[Oh, 2004] Oh, P. (2004). Flying insect inspired vision for micro-air-vehicle navigation. *Autonomous Unmanned Vehicles System International Symposium, Aug. 3-5, Anaheim USA*.

[Oh et al., 2004] Oh, P., Green, W., and Barrows, G. (2004). Neural nets and optic flow for autonomous micro-air-vehicle navigation. In *Proc. Int. Mech. Eng. Congress and Exposition*.

[Pivtoraiko et al., 2009] Pivtoraiko, M., Knepper, R. A., and Kelly, A. (2009). Differentially Constrained Mobile Robot Motion Planning in State Lattices. *Journal of Field Robotics*, 26(3):308–333.

[Preparata and Ian Shamos, 1985] Preparata, F. and Ian Shamos, M. (1985). *Compuational geometry: an introduction*. Springer.

[Proctor and Johnson, 2004] Proctor, A. and Johnson, E. (2004). Vision-Only Aircraft Flight Control Methods and Test Results. In *Proceedings of the AIAA Conference on Guidance, Navigation, and Control (GNC)*.

[Proctor et al., 2006] Proctor, A. A., Johnson, E. N., and Apker, T. B. (2006). Vision-only control and guidance for aircraft. *Journal of Field Robotics*, 23(10):863–890.

[Quinlan and Khatib, 1993] Quinlan, S. and Khatib, O. (1993). Elastic bands: connecting path planning and control. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2.

[Ratliff et al., 2009] Ratliff, N., Zucker, M., Bagnell, J. A., and Srinivasa, S. (2009). CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 489–494, Kobe.

[Richards, 2002] Richards, A. (2002). *Trajectory optimization using mixed-integer linear programming*. PhD thesis, MIT, Boston, MA.

[Saripalli and Sukhatme, 2003] Saripalli, S. and Sukhatme, G. (2003). Landing on a moving target using an autonomous helicopter. In *Proceedings of the International Conference on Field and Service Robotics*.

[Saripalli and Sukhatme, 2007] Saripalli, S. and Sukhatme, G. (2007). Landing a Helicopter on a Moving Target. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2030–2035.

[Scherer et al., 2010] Scherer, S., Chamberlain, L., and Singh, S. (2010). Online Assessment of Landing Sites. In *AIAA Infotech@Aerospace*, Atlanta.

[Scherer et al., 2009] Scherer, S., Ferguson, D., and Singh, S. (2009). Efficient C-space and cost function updates in 3D for unmanned aerial vehicles. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 2049–2054. IEEE Press.

[Scherer et al., 2005] Scherer, S., Lerda, F., and Clarke, E. (2005). Model checking of robotic control systems. In *Proc. of the 8th International symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS)*, Munich, Germany.

[Scherer et al., 2008] Scherer, S., Singh, S., Chamberlain, L., and Elgersma, M. (2008). Flying Fast and Low Among Obstacles: Methodology and Experiments. *The International Journal of Robotics Research*, 27(5):549–574.

[Schlemmer et al., 1995] Schlemmer, M., Finsterwalder, R., and Grubel, G. (1995). Dynamic Trajectory Optimization in Real Time for Moving Obstacles Avoidance by a Ten Degrees of Freedom Manipulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 72–77 vol.3.

[Serrano, 2006] Serrano, N. (2006). A Bayesian Framework for Landing Site Selection during Autonomous Spacecraft Descent. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5112–5117.

[Shim et al., 2006] Shim, D., Chung, H., and Sastry, S. (2006). Conflict-free navigation in unknown urban environments. *Robotics & Automation Magazine, IEEE*, 13(3):27–33.

[Sofman et al., 2006] Sofman, B., Bagnell, J., Stentz, A., and Vandapel, N. (2006). Terrain Classification from Aerial Data to Support Ground Vehicle Navigation. Technical Report CMU-RI-TR-05-39, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[Sprinkle et al., 2005] Sprinkle, J., Eklund, J., and Sastry, S. (2005). Deciding to land a UAV safely in real time. In *Proceedings of the American Control Conference (ACC)*, pages 3506–3511 vol. 5.

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

[Stentz, 1994] Stentz, A. (1994). The D* Algorithm for Real-Time Planning of Optimal Traverses. Technical Report CMU-RI-TR-94-37, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[Templeton, 2007] Templeton, T. (2007). Autonomous Vision-based Rotorcraft Landing and Accurate Aerial Terrain Mapping in an Unknown Environment. Technical Report USB/EECS-2007-18, University of California at Berkeley, Berkeley, CA.

[Templeton et al., 2007] Templeton, T., Shim, D., Geyer, C., and Sastry, S. (2007). Autonomous Vision-based Landing and Terrain Mapping Using an MPC-controlled Unmanned Rotorcraft. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1349–1356.

[Thrun et al., 2003] Thrun, S., Diel, M., and Hahnel, D. (2003). Scan Alignment and 3-D Surface Modeling with a Helicopter Platform. In *Proceedings of the International Conference on Field and Service Robotics*.

[Tisdale et al., 2009] Tisdale, J., Kim, Z., and Hedrick, J. (2009). Autonomous UAV path planning and estimation. *Robotics & Automation Magazine, IEEE*, 16(2):35–42.

[Tompkins et al., 2002] Tompkins, P., Stentz, T., and Whittaker, W. (2002). Mission planning for the Sun-Synchronous Navigation Field Experiment. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 3493–3500. IEEE.

[Tournier, 2006] Tournier, G. (2006). *Six Degree of Freedom Estimation Using Monocular Vision and Moire Patterns*. PhD thesis, MIT, Boston, MA.

[Tsenkov et al., 2008] Tsenkov, P., Howlett, J. K., Whalley, M., Schulein, G., Takahasi, M., Rhinehart, M. H., and Mettler, B. (2008). A System for 3D Autonomous Rotorcraft Navigation in Urban Environments . In *AIAA 2008*, page 23.

[Vandapel et al., 2005] Vandapel, N., Kuffner, J., and Amidi, O. (2005). Planning 3-D Path Networks in Unstructured Environments. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 4624–4629.

[Viquerat et al., 2007] Viquerat, A., Blackhall, L., Reid, A., and Sukkarieh, S. (2007). Reactive Collision Avoidance for Unmanned Aerial Vehicles using Doppler Radar. In *Proceedings of the International Conference on Field and Service Robotics*.

[weControl AG, 2006] weControl AG (2006). http://www.wecontrol.ch/. *Miscellaneous*.

[Welford, 1962] Welford, B. (1962). Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420.

[Whalley et al., 2008] Whalley, M., Directorate, A., Schulein, G., and Theodore, C. (2008). Design and Flight Test Results for a Hemispherical LADAR Developed to Support Unmanned Rotorcraft Urban Operations Research. In *American Helicopter Society 64th Annual Forum, Montreal, Canada, April 29 - May 1*.

[Whalley et al., 2005] Whalley, M., Freed, M., Harris, R., and Takahashi, M. (2005). Design, Integration, and Flight Test Results for an Autonomous Surveillance Helicopter. In *Proceedings of the AHS International Specialists' Meeting on Unmanned Rotorcraft*.

[Yu et al., 2007] Yu, Z., Nonami, K., Shin, J., and Celestino, D. (2007). 3D Vision Based Landing Control of a Small Scale Autonomous Helicopter. *International Journal of Advanced Robotic Systems*, 4(1):51–56.

[Zapata and Lepinay, 1998] Zapata, R. and Lepinay, P. (1998). Collision avoidance of a 3D simulated flying robot. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 113–120.

[Zapata and Lepinay, 1999] Zapata, R. and Lepinay, P. (1999). Flying among obstacles. In *Advanced Mobile Robots, 1999. (Eurobot '99) 1999 Third European Workshop on*, pages 81–88.

[Zufferey et al., 2006] Zufferey, J., Klaptocz, A., Beyeler, A., Nicoud, J., and Floreano, D. (2006). A 10-gram Microflyer for Vision-based Indoor Navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3267–3272.