Flying Fast and Low Among Obstacles: Methodology and Experiments

Sebastian Scherer Sanjiv Singh Lyle Chamberlain Mike Elgersma

Abstract

Safe autonomous flight is essential for widespread acceptance of aircraft that must fly close to the ground. We have developed a methodology of collision avoidance that can be used in three dimensions in much the same way as autonomous ground vehicles that navigate over unexplored terrain. Safe navigation is accomplished by a combination of online environmental sensing, path planning and collision avoidance. Here we outline our methodology and report results with an autonomous helicopter that operates at low elevations in uncharted environments some of which are densely populated with obstacles such as buildings, trees and wires. We have recently completed over 700 successful runs in which the helicopter traveled between coarsely specified waypoints separated by hundreds of meters, at speeds up to 10 meters/sec at elevations of 5-11 meters above ground level. The helicopter safely avoids large objects like buildings and trees but also wires as thin as 6 mm. We believe this represents the first time an air vehicle has traveled this fast so close to obstacles. The collision avoidance method learns to avoid obstacles by observing the performance of a human operator.

1 Introduction

Today the threat of low-altitude obstacles constrains fielded unmanned aerial vehicles (UAVs) to operate at high altitude, or under close human supervision at low altitude. This is because even a low-speed collision with the environment can be fatal. Safe autonomous flight is essential for widespread acceptance of aircraft that must fly 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 (Green et al., 2005). Likewise, UAVs performing reconnaissance for the police, news or the military must fly low enough that the environment presents 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 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

Figure 1: The helicopter platform we used to test collision avoidance is flying autonomously in between two poles 10 m apart. The rotor span of this helicopter is 3.4 m

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.

Our planner is based on an implementation of Laplace's equation that generates a potential function with a unique minimum at the goal. The advantage of this method is that it provides smooth paths that are equidistant from obstacles rather than getting close to them. 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 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 1) in two significantly different environments at speeds (up to 20 knots) significantly greater than those attempted by anyone to date. .

Next we show related work in section 2, then describe sensor processing in section 3. Section 4 explains our architecture and algorithm. In section 5 we describe the flight hardware and range sensor. Section 6 describes our simulator and in section 7 we show results from field tests and conclude in section 8.

2 Related Work

In this section we examine related work as well as explain the rationale for design decisions made in sensing and algorithms for the development of this platform. First we will show prior work on obstacle avoidance

algorithms and systems for aerial vehicles and then present related work in sensing for UAVs.

2.1 Collision Avoidance

Current UAV obstacle avoidance techniques can be categorized into two broad methods: Planning and Reactive. Planning paradigms use a world map and plan a trajectory for the vehicle to follow. Such approaches must find trajectories that not only geometrically avoid obstacles but also ensure that the dynamics of the vehicle are capable of following the paths. This can be prohibitively expensive to compute if the trajectory has to be continuously revised. In contrast, reactive methods overcome the real-time problem by using a simple formula to *react* to obstacles as they appear, but they cannot guarantee an appropriate solution to every possible situation.

Vision-based reactive methods have been popular because payload weight is a serious limitation for UAVs. For example, Merrel et al. (Merrell et al., 2004) proposed to use optical flow in order to compute how to avoid obstacles. The micro flyer developed by Zufferey and Floreano is expected to reactively avoid obstacles using very small 1-D cameras(Zufferey and Floreano, 2005). Zapata and Lepinay have proposed a reactive algorithm similar to ours in motivation, but we are aware only of simulated results (Zapata and Lepinay, 1999; Zapata and Liepinay, 1998). Hrabar et al. use vision in order to navigate in canyon-like environments using optical flow(Hrabar and Sukhatme, 2003; Hrabar et al., 2004). This method implicitly avoids the walls of the canyons by centering the vehicle such that optical flow from both sides of the canyon is equal. 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).

Larger helicopters such as our Yamaha RMax helicopter on the other hand can afford more resources to explore alternative plans. One approach by Vandapel et al. implements a planning algorithm that uses point clouds in order to determine a path tunnel network(Vandapel et al., 2005). While this work used real data taken from a helicopter, the path produced was only visualized and not executed. Kitamura et al. use an octree representation of the world to plan paths using a 3D A* implementation(Kitamura et al., 1996). Frazolli et al. on the other hand use a concatenation of motion primitives in order to construct valid paths that avoid obstacles and achieve a desired goal pose(Frazzoli et al., 2005). On our helicopter we use a Laplacian path planning algorithm that is similar to the approach used in (Jackson et al., 2005; Connolly and Grupen, 1993; Li and Bui, 1998).

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., 2005).

Our collision avoidance system combines reactive and planning algorithms and therefore enables travel at speeds limited only by the furthest distance at which the smallest obstacle can be avoided. At the same time, we know that our system can plan complicated paths to arrive at a difficult goal without risking a collision. A similar reactive algorithm in 2D was deployed previously on a ground patrol robot (Roth et al., 2005). The local planning for both the 3D and 2D obstacle avoidance is related to models of humans navigation among discrete obstacles (Fajen and Warren, 2003).

Initially intended to provide an empirical model of how humans get to a point goal in the presence of point obstacles, this method has been been used by Fajen et al. as a control scheme for mobile robots operating under some restrictions– only point obstacles are used to interrupt a nominally straight line path to the goal and the vehicle travels at sub-meter/sec speeds(Fajen et al., 2003). Recently Huang et al., have used a modified version of the model proposed by Fajen & Warren geared towards obstacle avoidance using a monocular camera (Huang et al., 2006). Since range to obstacles can not be measured directly, the width of obstacles (segmented in the image) is used instead of the distance. Below we report on the extensions we have made to the basic model of collision avoidance proposed by Fajen & Warren to allow implementation on an air vehicle. While the researchers who have used this methodology have found the parameters of the controller by inspection, Hamner et al. have developed a method to automatically find such parameters from observation of human operators, a method that has been separately discussed in detail (Hamner et al., 2006) for a ground vehicle application and extended here for use by air vehicles.

2.2 Sensing Technology

A robot must be able to reliably detect and recognize obstacles before it can avoid them. Previous work on UAV obstacle avoidance uses many different sensor modalities. Active sensors such as radar and ladar operate with and without ambient light; however, unlike passive sensors such as cameras, the energy they emit into the environment can betray their position to others.

Radar is a reliable sensor to detect and track other aircraft and also is used on cars to implement smart cruise control. Foessel reconstructed an environment using a motion-free scanning radar(Foessel, 2002). Using a radar sensor in our application however poses several problems. Small beamwidth radars are difficult to construct and tend to have large antennas. Furthermore, radar sensors also suffer problems from multi path returns. In our evaluation of sensors the sensitivity of radar was not sufficient to detect wires reliably at shallow angles because of specular reflection of the radar waves.

Computer vision is another popular approach to obstacle sensing, because cameras are relatively light and mechanically simple. Hartley and Zissermann present a structure from motion algorithm that uses the parallax generated by a single moving camera(Hartley and Zisserman, 2004). This approach can be problematic because the structure from motion technique does not solve for scale, so distance and size of detected obstacles remains unknown. Other groups such as Call et al. compute the instantaneous optic flow field, interpreting areas with greater flow magnitude to contain obstacles(Call et al., 2006). The scale problem can be solved by using stereo camera pairs to solve the structure of a scene; however, the baseline distance between the cameras limits the long-range accuracy of the stereo approach. In another approach, Strelow and Singh use inertial state information to estimate distance between two single-camera images(Strelow and Singh, 2002). This estimate solves the scale in the structure-from-motion problem.

Camera-based approaches have drawbacks that make use in a fieldable UAV for obstacle avoidance difficult. Since cameras are passive sensors that use ambient light there are problems if there is too much contrast, and too little, or too much light, because image sensors used in cameras have a limited dynamic range.

The size of detectable objects and measurement accuracy of camera-based systems decrease rapidly with distance. A camera may easily detect a tree, but an individual branch or a telephone wire may be virtually invisible from a distance. Stereo vision techniques have to rely on texture. Homogeneous regions within a scene (such as a sand dune or a uniform wall) will produce no range information.

Ladar systems have many advantages over machine vision for detecting obstacles because they are active sensors and do not rely on ambient lighting or texture. Obstacles are detected by scanning a laser beam in the environment and measuring distance through time of flight or interference. Especially small obstacles such as wires can be better detected using ladar. Detection accuracy is constant for all detectable distances and is instead a function of laser intensity and beam size.

Ladar does have a few problems as well, however. Specular reflection of the laser beam causes errors. A glossy finish on a car or a puddle of water remain invisible at shallow angles. Worse, the reflected light may bounce off another obstacle and return to the sensor, giving a false distance measurement that dangerously indicates an obstacle further away than the first surface. Also ladar may be temporarily blinded by the sun, and is more complicated mechanically which may make it less reliable and heavier than other systems.

Off the shelf systems primarily use a single-line scan that produces 2-dimensional information. A popular approach has been to construct 3-dimensional maps for ground vehicles by mechanically sweeping these sensors back and forth (Trepagnier et al., 2006); however, the mechanisms that sweep the ladar units are slow and prohibitively bulky for most air vehicles. Typically, a UAV overflies an area at a safe altitude with a 2D ladar to build a static map before venturing lower near obstacles. 2D ladar is ill-suited to real-time obstacle detection in 3D navigation problems. One could conceivably build a 3D map using the roll/pitch/yaw motions of a helicopter, but the aircraft would have to fly very slowly and pitch around awkwardly to get a suitable amount of information.

Kanade et al. use a combination of ladar and vision; however, they use vision only as a state-estimation input and employ a line scanning ladar for obtaining mapping data of the environment (Kanade et al., 2004).

Flash lidar saturates a scene with a bright light and computes time-of-flight for every pixel in an imager. The result is a dense 3D image that is updated very rapidly. The entire system is solid state with no moving parts, though a suitable light source requires a lot of energy and generates large amounts of heat. In our initial review of this technology we did not find a system suitable for our needs, though the technology does

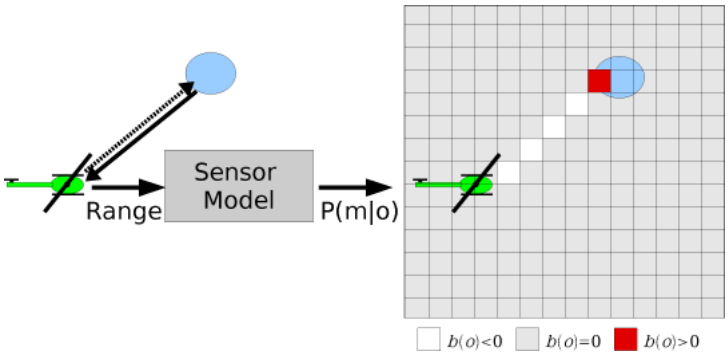

Figure 2: A laser rangefinder returns a signal indicating a hit. The signal is mapped to an occupancy probability $P(m|o)$. This probability is used to update the relevant belief cells in the evidence grid.

hold promise in the near future.

Since radar was not able to detect the necessary obstacles, passive camera systems are not reliable enough, and flash lidar was not available we used a two-axis scanning ladar because it was able to detect very small obstacles at a significant range. See Section 5 for more information about the range sensor we chose.

3 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. 2 summarizes the flow of data.

3.1 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 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:

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

however since $P(m) = 0.5$:

$$b'(m) = \frac{P(m|o)}{1 - P(m|o)} \cdot \frac{b(m)}{1 - b(m)} \tag{2}$$

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 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.

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

$$\overline{b}(m) = \ln \frac{b(m)}{1 - b(m)} = \ln b(m) - \ln(1 - b(m)) \tag{3}$$

Therefore the new update rule is

$$\overline{b}'(m) = \overline{b}(m) + \ln P(m|o) - \ln(1 - P(m|o)) \tag{4}$$

Our robot uses a laser rangefinder that scans in azimuth and elevation (Also see section 5). 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

$$\ln P(m|o) - \ln(1 - P(m|o)) = 127 \tag{5}$$

and

$$\ln P(m|\overline{o}) - \ln(1 - P(m|\overline{o})) = -1 \tag{6}$$

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 (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 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 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

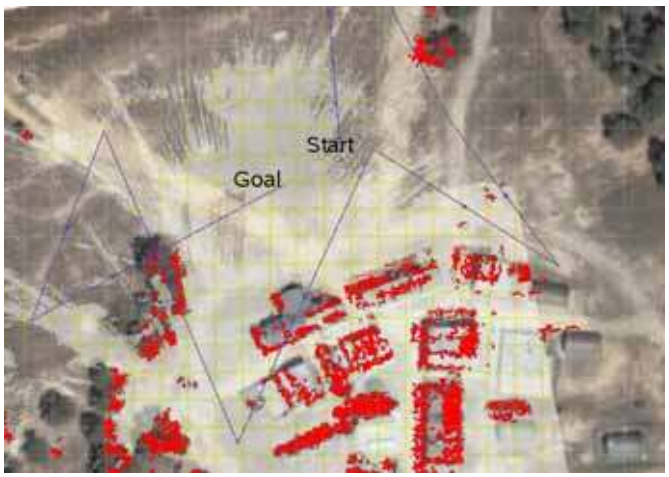

Figure 3: A 2D horizontal slice of an evidence grid from the McKenna military operations in urban terrain (MOUT) site at Ft. Benning, GA (USA). This model was created by flying part of the shown path. Dark(red) represents occupancy and white is evidence of free space. An aerial image is overlayed on 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.

amounts of wasted memory. As shown in Table 1 most cells are never touched. Only a small percentage(0.26% - 2.1%) of cells is occupied. 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. This model was created using the displayed flight path and is overlayed 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.

4 Obstacle Avoidance 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. 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

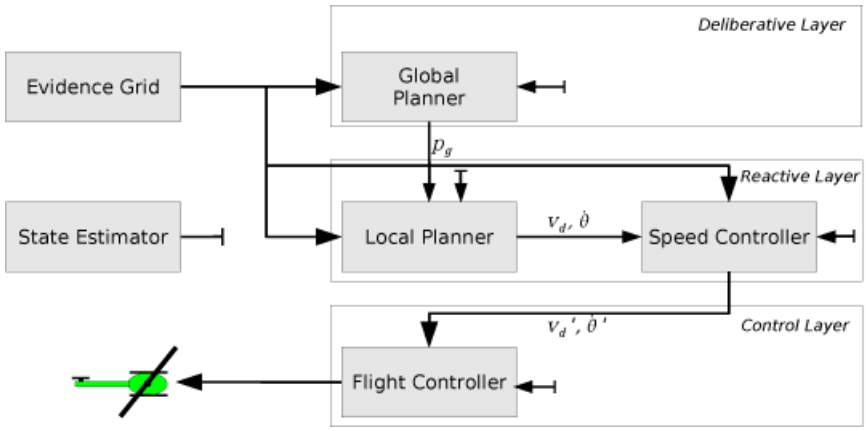

Figure 4: Overall software architecture of the algorithms. The higher the layer the lower the frequency of execution. A path p_g is produced in the planning layer and transmitted to the reactive layer. The reactive layer produces commands $(v'_d, \dot{\theta}')$ that are then executed by the flight controller.

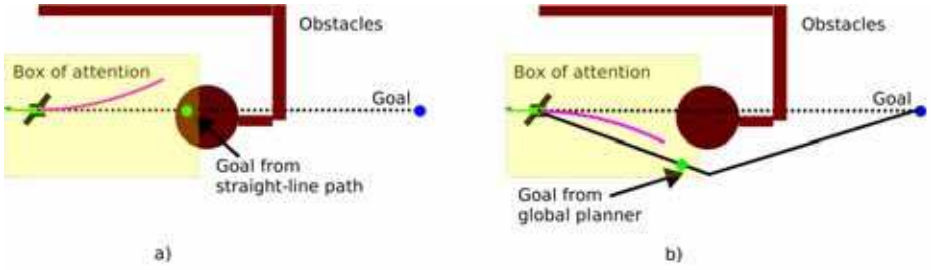

Figure 5: The benefit of combining local and global methods for navigation. In (a) the vehicle uses obstacles in the "box of attention" in its guidance. Since the horizon is short, this can lead the vehicle to a cul de sac from where it might have to either turn around or execute an expensive climb maneuver. In (b), the vehicle selects its temporary goal point along the path found by the global planner and ends up following a better route.

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. 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.

4.1 Global Planner

Our global planning algorithm uses a binary occupancy grid and goal point to assign boundary conditions for Laplace's equation. The solution of Laplace's equation then gives a potential and paths that follow the gradient of that potential, lead to the goal point. Boundary-condition values are chosen in a way that guarantees that the potential has no local minima or local maxima, so all paths lead to the goal.

Using three spatial coordinates, position is given by $r = [x, y, z]$, potential at each point is given by $\phi(r)$ and Laplace's equation is given by:

$$\frac{\delta^2}{\delta x^2}\phi(x,y,z) + \frac{\delta^2}{\delta y^2}\phi(x,y,z) + \frac{\delta^2}{\delta z^2}\phi(x,y,z) = 0 \tag{7}$$

A moving 3D box, containing a subset of the evidence grid is centered at the vehicle location. The algorithm assigns a potential value of $\phi(r) = -1$ for r on the goal region and a potential value of $\phi(r) = 0$ for r on any boundaries or obstacle. Obstacles are expanded by a sphere approximation to the volume of the vehicle and planning is performed in this configuration space. Dynamic range requirements make it necessary to use $\phi(r) = 0$ for r on obstacles, since long narrow passages between obstacles have large regions where $\phi(nearboundary) \approx \phi(boundary)$, due to averaging of nearby cells. If $\phi(boundary) = 0$, then $\phi(nearboundary) = 10^{-300}$ is still distinguishable from 0. While if $\phi(boundary) = 1$, then $\phi(nearboundary) = 1 + 10^{-300}$ would round to 1, giving no gradient through regions between obstacles.

Using the assigned values of potential on boundaries, obstacles and goal point, Laplaces equation is solved in the interior of the 3D region, guaranteeing no local minima in the interior of the domain. All minima and maxima must then lie on the boundary of the domain, which consists of its outer boundary, obstacles, and the goal point. Since there are only two values of the potential on these boundaries, one of those values is the only minimum and the other value is the only maximum, leaving no possibility for local minima or local maxima. A path from any initial point, $r(0)$, to the goal, is constructed by following the negative gradient of the potential $\phi(r)$:

$$\frac{\delta path}{\delta t} = -[\frac{\delta\phi}{\delta x}, \frac{\delta\phi}{\delta y}, \frac{\delta\phi}{\delta z}]/|||[\frac{\delta\phi}{\delta x}, \frac{\delta\phi}{\delta y}, \frac{\delta\phi}{\delta z}]|| \tag{8}$$

A physical analogy for paths obtained by Laplace's equation is to apply a voltage of 0 to all boundary and obstacle locations, and a voltage of -1 to the goal region. Mass-less positive charges will follow paths along the gradient of potential, from anywhere in the interior to the goal region. Laplace's equation sets the divergence of a potential to zero in the interior of a domain.

Numerical solutions of Laplace's equation are obtained by gridding the domain, then iteratively setting the potential at each interior point equal to the average of its nearest neighbors. Since the average of neighboring potential values cannot be larger than all neighboring values, no interior point can be a local maxima. This averaging procedure is justified as follows:

The discrete version of the spatial derivatives can be expressed as

$$\frac{\delta\phi}{\delta x} = \frac{\phi(x[i+1], y[i], z[i]) - \phi(x[i], y[i], z[i])}{x[i+1] - x[i]} \tag{9}$$

$$\frac{\delta\phi}{\delta y} = \frac{\phi(x[i], y[i+1], z[i]) - \phi(x[i], y[i], z[i])}{y[i+1] - y[i]} \tag{10}$$

$$\frac{\delta\phi}{\delta y} = \frac{\phi(x[i], y[i], z[i+1]) - \phi(x[i], y[i], z[i])}{z[i+1] - z[i]} \tag{11}$$

and therefore the corresponding discrete second derivatives in Laplace's equation are

$$\frac{\delta^2\phi}{\delta x^2} = \frac{\phi(x[i+1], y[i], z[i]) - 2 \cdot \phi(x[i], y[i], z[i]) + \phi(x[i-1], y[i], z[i])}{(x[i+1] - x[i])^2} \tag{12}$$

$$\frac{\delta^2\phi}{\delta y^2} = \frac{\phi(x[i], y[i+1], z[i]) - 2 \cdot \phi(x[i], y[i], z[i]) + \phi(x[i], y[i-1], z[i])}{(y[i+1] - y[i])^2} \tag{13}$$

$$\frac{\delta^2\phi}{\delta z^2} = \frac{\phi(x[i], y[i], z[i+1]) - 2 \cdot \phi(x[i], y[i], z[i]) + \phi(x[i], y[i], z[i-1])}{(z[i+1] - z[i])^2} \tag{14}$$

The discrete version of Laplace's equation can therefore be expressed as

$$\frac{\phi(x[i+1], y[i], z[i]) - 2 \cdot \phi(x[i], y[i], z[i]) + \phi(x[i-1], y[i], z[i])}{(x[i+1] - x[i])^2} + \tag{15}$$

$$\frac{\phi(x[i], y[i+1], z[i]) - 2 \cdot \phi(x[i], y[i], z[i]) + \phi(x[i], y[i-1], z[i])}{(y[i+1] - y[i])^2} + \tag{16}$$

$$\frac{\phi(x[i], y[i], z[i+1]) - 2 \cdot \phi(x[i], y[i], z[i]) + \phi(x[i], y[i], z[i-1])}{(z[i+1] - z[i])^2} \tag{17}$$

$$= 0 \tag{18}$$

$$\tag{19}$$

Since we assume the distance between the cells to be 1 we can rewrite this equation as

$$\phi(x[i], y[i], z[i]) = \tag{20}$$

$$(\phi(x[i+1], y[i], z[i]) + \phi(x[i], y[i+1], z[i]) + \phi(x[i], y[i], z[i+1]) \tag{21}$$

$$\phi(x[i-1], y[i], z[i]) + \phi(x[i], y[i-1], z[i]) + \phi(x[i], y[i], z[i-1]))/6 \tag{22}$$

which says that in order to fulfill Laplace's equation in the discrete case, each cell needs to be the average of it's neighbors. By iteratively setting each cell to the average of its neighbors, the solution eventually converges to a static solution satisfying this averaging condition.

The number of averaging iterations required to converge to the solution depends on the obstacle geometry, and the way that the grid size is chosen. The exact solution would require an infinite number of iterations, but we only need to iterate until the approximate solution has no local minima. This only takes a few iterations when multiple grid sizes are used, and paths between obstacles are not too long and narrow. It is easy to check for local minima to determine how many iterations are necessary.

By varying the grid size from the crudest that stills leaves paths between obstacles, to the finest that is required for smooth paths, the iteration can be made to converge in a time proportional to the number, N, of cells in the finest grid. The averaging iteration converges quickly on crude grids, since there are so few cells in the crude grid. The crude grid solution is then used to initialize the solution on finer grids. This multigrid technique is described in (U. Trottenberg and Schuller, 2001) and is applied to robotic path planning in (Li and Bui, 1998) and (Jackson et al., 2005).

After the Laplacian solution is computed, the gradient of the potential is used to compute a path, while the next Laplacian solution is computed. The next Laplacian solution has only slightly changed boundary conditions, since only a few new obstacles were sensed and put into the evidence grid during the last computational cycle. Therefore the potential solution will only have changed slightly, so the old potential solution is a good starting solution for the next iteration. The old solution is used as an initial solution for the coarse grid by smoothing and decreasing the resolution and then going from the coarser solution to the finer solution.

We use a multigrid process with grids of sizes $(\frac{n_x}{2^i}, \frac{n_y}{2^i}, \frac{n_z}{2^i})$ where $i = \ 0, 1, 2, 3$ and for $i = 0$ we have $N = n_x n_y n_z$ total cells. Let the grids of the potential values at resolutions i be defined as ϕ_i. The multigrid solution process then consists of the steps shown in Algorithm 1. It is important to flip the order of the smoothing procedure (Algorithm 2) if we do not perform an infinite number of iterations because even though the grid might converge to one local minimum at the goal it becomes biased towards one side. Therefore we flip the order of traversal for each axis on odd iterations.

Algorithm 1 Multi-resolution Laplacian Planning Algorithm

$i \leftarrow 0$
$\phi_i \leftarrow \phi'_i$ (Assign the old solution to new grid)
$\phi_i \leftarrow smooth_i(\phi_i)$
for all $i \in \{1, 2, 3\}$ **do**
  $\phi_i \leftarrow downSample_i(\phi_{i-1})$
  $\phi_i \leftarrow smooth_i(\phi_i)$
end for
for all $i \in \{2, 1, 0\}$ **do**
  $\phi_i \leftarrow upSample_i(\phi_{i+1})$
  $\phi_i \leftarrow smooth_i(\phi_i)$
end for

Algorithm 2 $smooth_i$

for $k = 0$ to some c **do**
  for All cells in the grid: $x = 0$ to $\frac{n_x}{2^i}$, $y = 0$ to $\frac{n_y}{2^i}$, and $z = 0$ to $\frac{n_z}{2^i}$. If k is odd flip order of traversal. **do**
    $\phi_i(x, y, z) \leftarrow (\phi_i(x+1, y, z) + \phi_i(x, y+1, z) + \phi_i(x, y, z+1)\phi_i(x-1, y, z) + \phi_i(x, y-1, z) + \phi_i(x, y, z-1))/6$
  end for
end for

Algorithm 3 $downSample_i$

for All cells in the grid: $x = 0$ to $\frac{n_x}{2^i}$, $y = 0$ to $\frac{n_y}{2^i}$, and $z = 0$ to $\frac{n_z}{2^i}$ **do**
  $\phi_i(x, y, z) \leftarrow (\phi_{i-1}(2x, 2y, 2z) + \phi_{i-1}(2x + 1, 2y, 2z) + \phi_{i-1}(2x, 2y + 1, 2z) + \phi_{i-1}(2x, 2y, 2z + 1))/4$
end for

Algorithm 4 $upSample_i$

for All cells in the grid: $x = 0$ to $\frac{n_x}{2^{i+1}}$, $y = 0$ to $\frac{n_y}{2^{i+1}}$, and $z = 0$ to $\frac{n_z}{2^{i+1}}$ **do**
  $\phi_i(x, y, z) \leftarrow (\phi_{i+1}(x/2, y/2, z/2)$
  $\phi_i(x + 1, y, z) \leftarrow (\phi_{i+1}(x/2, y/2, z/2)$
  $\phi_i(x, y + 1, z) \leftarrow (\phi_{i+1}(x/2, y/2, z/2)$
  $\phi_i(x, y, z + 1) \leftarrow (\phi_{i+1}(x/2, y/2, z/2)$
end for

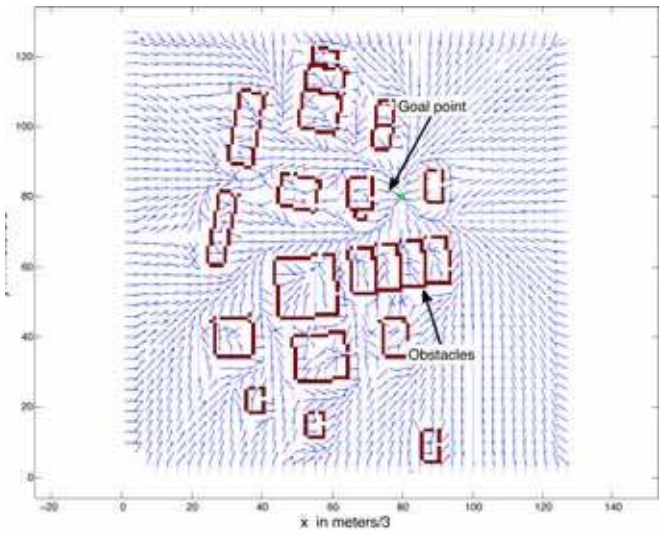

Figure 6: A vector field plot of a 2D slice of the potential gradient generated by the multigrid Laplacian algorithm for a model of the McKenna MOUT site. The boxes show obstacle boundaries while the arrows show the direction of the gradient. The gradient always points away from obstacles and leads to the goal point outside of an obstacle boundary.

The single sequence from finest to coarsest back to finest grid sizes, is referred to as a V cycle. Some applications use more complicated cycles, or multiple V cycles (W cycles). We use only one V cycle, since on the coarsest grid, resolution errors in representing boundary shapes introduce significant errors. These errors get corrected on finer grids during the last half of the V cycle, but if another V cycle were done, those errors would be re-introduced, causing no significant advantage in doing additional V cycles.

Textbook convergence proofs for empty domains give the total number of computations to be $c \cdot N$, where N is the number of cells in the finest grid and $c \approx 5$ is some small number of iterations at each grid size. Note that the empty-domain (no obstacles) convergence speed relies on being able to set the crudest grid cell size equal to the entire domain for the crudest solution. However, we have found that in a domain with obstacles, the number of needed iterations is given by $c \cdot \frac{path\ length}{path\ width}$, since path-width between obstacles limits the largest cell size of the crudest grid that still preserves the topology of the computed paths. With the largest grid cell size set equal to the path width, the number of grid cells along the path equals $\frac{path\ length}{path\ width}$. The iterative process of setting a cell's potential equal to the average of its neighbor's potentials, propagates a nonzero solution value a distance of one more grid cells along the path, at each iteration. So it takes $\frac{path\ length}{path\ width}$ iterations for a nonzero solution to propagate along the entire path length, when the crudest grid cell size is equal to the path width. After $c \cdot \frac{path\ length}{path\ width}$ iterations, with $c \approx 5$, the iteration converges on the crudest grid. Using this same number of iterations on each finer grid size results in the bulk of the work being done by the $c \cdot \frac{path\ length}{path\ width}$ iterations on the finest grid size, for a total number of operations of approximately $c \cdot \frac{path\ length}{path\ width} \cdot N$, where N is the total number of cells in the finest grid.

A C implementation of this procedure has a constant runtime of 0.07 seconds for $(n_x, n_y, n_y) = (64, 64, 32)$ and 0.6 seconds for $(n_x, n_y, n_z) = (128, 128, 64)$ on a 1.8 GHz Pentium M with 2 MB L2 cache. Since the algorithm was sharing processing with sensor processing and other algorithms the update of the solution was slowed to approximately 2.0 seconds for $(n_x, n_y, n_z) = (128, 128, 64)$ on a 1.6 GHz Pentium M with 1 MB L2 cache.

A 2D slice at $n_z = 2$ of an example region of size $(n_x, n_y, n_z) = (128, 128, 16)$ is shown in Fig. 6. The vector field shows the direction of the gradient at regular intervals. The gradient always points away from obstacles and converges to the goal point. A path generated from the potential will be between obstacles because this is were the potential balances. This is a desirable property because it increases the safety of the path. However at the same time if there are no obstacles present the path will be curved if the goal

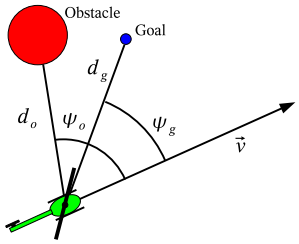

Figure 7: A diagram illustrating the terms used in the control law. The coordinates used are expressed in a robot centric reference frame.

is in a corner because the boundary of the grid is considered an obstacle and has the shape of a box. We alleviate this issue because we replace curvy segments in configuration space with the furthest line of sight line segments. Even though this reduces the smoothness of the path it removes unnecessary turns.

While the global planner produces smooth paths from the current point to the goal, it should be noted that the paths do not respect dynamics or any need for the vehicle to point in the direction of travel which is necessary for the vehicle to sense the environment. The planner is also not suitable for reaction to unanticipated obstacles while moving at high speeds. Instead, this planner is used to provide general advice to a faster collision avoidance method that operates on local information. This is done by using a point at a set distance ahead on the path produced by the global planner as the "carrot" for the local planner.

4.2 Local Planner

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.

4.2.1 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. 7. The attraction increases as the distance to the goal decreases and as the angle to the goal increases (Fig. 8.A-B), yielding the goal attraction function

$$attract_{FW}(g) = k_g \psi_g (e^{-c_1 d_g} + c_2) \tag{23}$$

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. 8.C-D). Then for each obstacle, there is a repulsion function:

$$repulse_{FW}(o) = k_o \psi_o (e^{-c_3 d_o})(e^{-c_4 |\psi_o|}) \tag{24}$$

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

$$\ddot{\phi}_{FW} = -b\dot{\phi} - attract_{FW}(g) + \sum_{o \in O} repulse_{FW}(o) \tag{25}$$

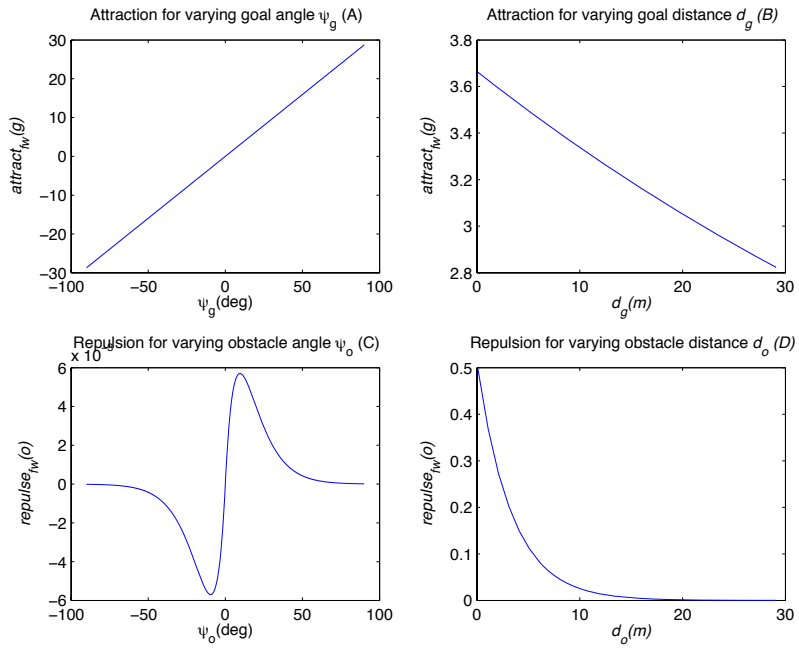

Figure 8: The attraction and repulsion for the original control law for varying goal angle (A), varying goal distance (B), varying obstacle angle (C), and varying obstale distance (D)

4.2.2 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. 9A-B). The vector of the two steering rates for goal attraction is therefore defined as

$$\overrightarrow{attract}_{3D}(g) = \overrightarrow{k_g} \begin{bmatrix} \theta_g \\ \phi_g \end{bmatrix} (e^{-c_1 d_g} + c_2) \tag{26}$$

Obstacles are also expressed in spherical coordinates. The repulsion increases exponentially with decreasing angles and decreases exponentially with increasing distance as shown in Fig. 9C-D. Also, larger angles from the other axis like ϕ for θ decrease the repulsion from obstacles. The repulsion function is

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

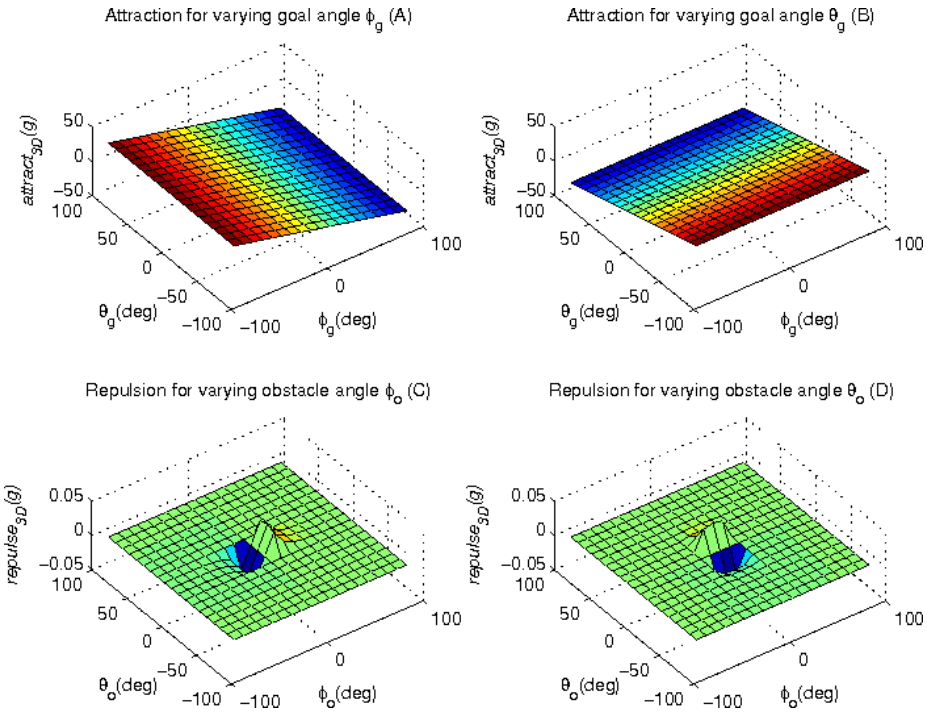

Figure 9: The attraction and repulsion surface for our modified control law. Plot (A) and (B) show the independence of the goal attraction on each axis. Since a sigmoid shapes the repulsion for the repulsion function in plot (C) and (D) is not uniform.

$$\begin{bmatrix} sign(\theta_o) \cdot sigmoid(s_1(1 - \frac{|\phi_o|}{s_2})) \\ sign(\phi_o) \cdot sigmoid(t_1(1 - \frac{|\theta_o|}{t_2})) \end{bmatrix} (e^{-c_3 d_o}) \begin{bmatrix} e^{-c_4|\theta_o|} \\ e^{-c_4|\phi_o|} \end{bmatrix} \tag{27}$$

where

$$sign(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x = 0 \\ -1 \; if \; x < 0 \end{cases} \tag{28}$$

and

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{29}$$

The resulting steering rate command sent is

$$\dot{\vec{\phi}}_{3D} = \overrightarrow{attract}_{3D}(g) + \sum_{o \in O} \overrightarrow{repulse}_{3D}(o) \tag{30}$$

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. 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

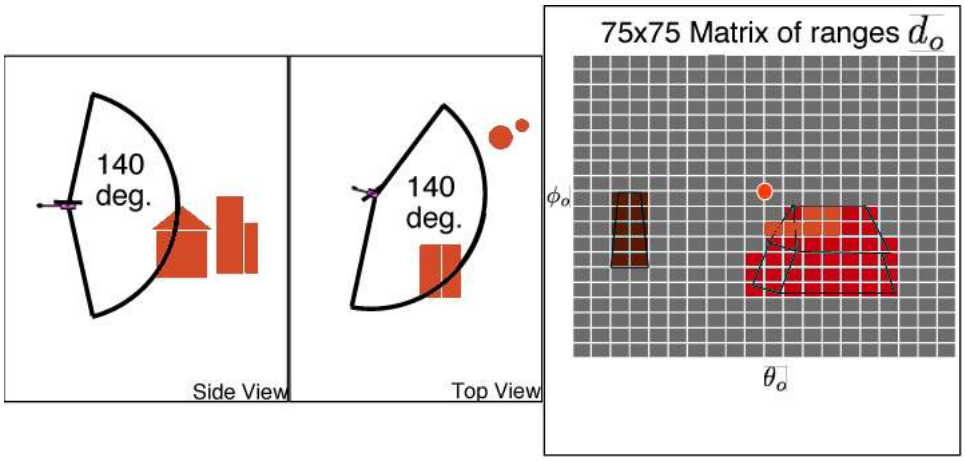

Figure 10: A diagram showing how an obstacle is projected into a grid-discretized spherical representation. The angle θ_o and ϕ_o determine the index into the grid. An obstacle at a similar angle is ignored because it is occluded by the closer obstacle.

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 $\left[\dot{\theta}, \dot{\phi}\right]$ 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 speed 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 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.

4.2.3 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. 10 to represent obstacles.

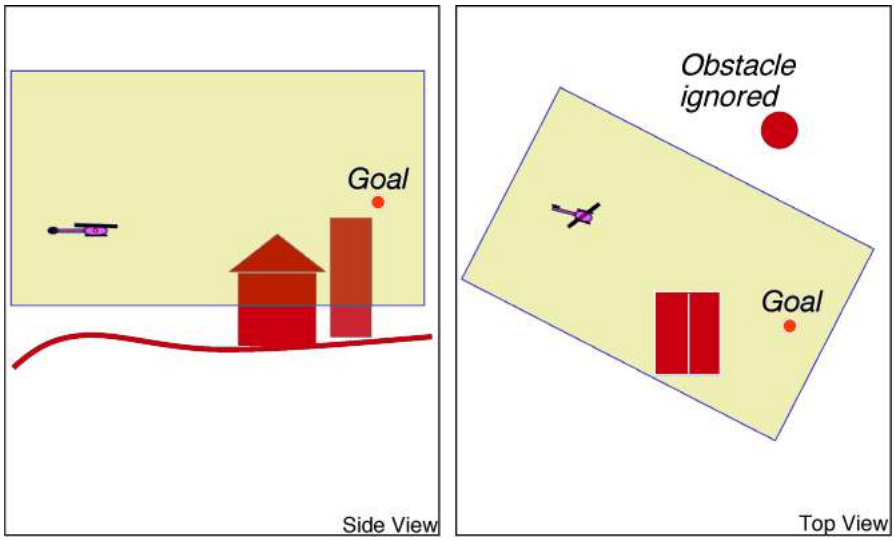

Figure 11: An example showing the box of attention. The box restricts the obstacles considered and is set between the goal point and the current location of the UAV.

The set of obstacles $o_i \in O$ where

$$o_i = \begin{bmatrix} \theta_i \\ \phi_i \\ d_i \end{bmatrix} \tag{31}$$

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. 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.

The obstacles considered by our reactive algorithm are additionally limited by a box-shaped constraint as shown in Fig. 11. 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 relevant 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.

4.2.4 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

$$\overline{u} = (k_g, c_1, c_2, s_1, s_2, t_1, t_2, k_{o,1}, k_{o,2}, c_{3,1}, c_{3,2}, c_{4,2}) \tag{32}$$

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 were determined

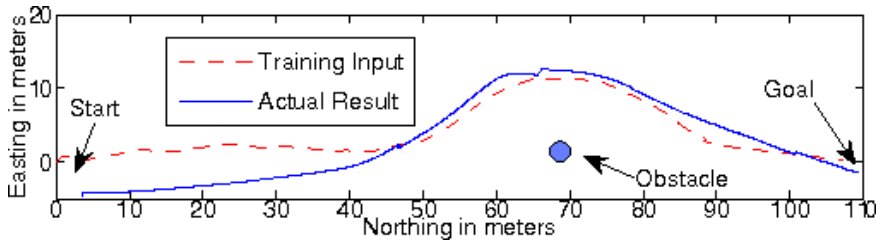

Figure 12: This figure shows the path used for training as a dashed line and the path taken by the RMax helicopter as a solid line. The path with the fine-tuned learned parameters still is a adequate match for our training path.

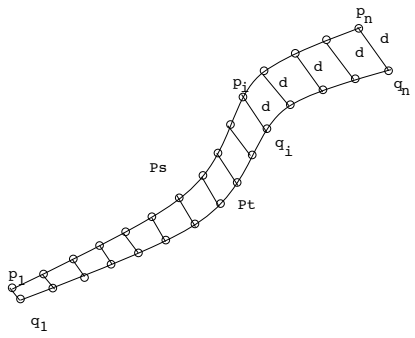

Figure 13: The difference between two path segments is used to optimize the parameter set $\bar{u}$. P_s is a recorded path segment while P_t was generated from a parameter set $\bar{u}$.

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. 12. 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:

$$u_e = (k_{o,1}, c_{3,1}, c_{4,1}) \tag{33}$$

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. 13: $d(\bar{u})_i = \sqrt{(k_i - x_i)^2 - (l_i - y_i)^2}$. 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 6 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.

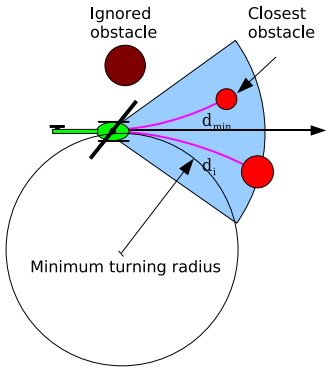

Figure 14: The speed controller slows the robot based on the closest reachable obstacle. Reachability is determined by the minimum turning radius based on the current velocity and the maximum turning rate of $30°/s$.

Figure 12 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.

The modified FW model appears to have enough parameters to be able to adjust the desired behavior but does not have too many degrees of freedom that it will overfit to a particular kind of obstacle. We fit the parameters 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 and the algorithm generalized to this different environment.

In prior work we evaluated a variant of the Fajen and Warren control law on a ground vehicle (reference removed for review). 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.

4.3 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. This assumes 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 power-on-descent and excessive load. The resulting command vector is scaled if these limits are exceeded.

Third and most importantly, the closest 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

$$d_b = vt_r + \frac{v^2}{2a_{max}} \tag{34}$$

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

The inverse can be calculated as

$$v_b = -a_{max}t_r + \sqrt{2a_{max}d_m + a_{max}^2 t_r} \tag{35}$$

Using the inverse one can determine a speed v_b at which it is possible to stop if the next command is a stop command. 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

$$v_b' = \frac{v_b}{cos(\theta)} \tag{36}$$

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. 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. 14 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

$$r_t = \frac{v_c}{\dot{\theta}_{max}} \tag{37}$$

where $\dot{\theta}_{max} = 30°/s$

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

$$\Delta\theta = \frac{r_t\pi - d_b}{2r_t} \tag{38}$$

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

$$d_o = 2 \cdot d \ \max(\phi, \theta)\frac{\sin(\frac{\pi}{2} - \max(\phi, \theta)}{\sin(2\max(\phi, \theta))} \tag{39}$$

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

$$d_m = min_{o \in O'}(d_o) \tag{40}$$

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. 15 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 $t_r = 1.1s$.

4.4 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

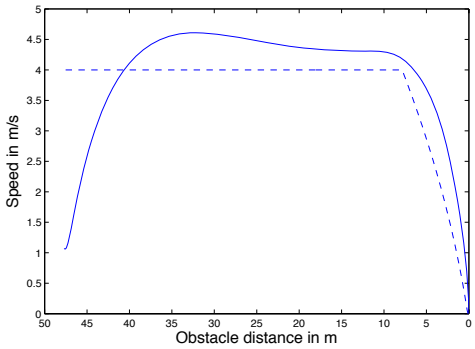

Figure 15: The solid line shows the actual speed of our physical simulation as the helicopter approaches a wall. In this example only the speed controller is avoiding obstacles because the local and global planning algorithm were disabled. The dashed line shows the commanded speed from the speed controller determined by the closest point to the wall. The helicopter stops without hitting the wall.

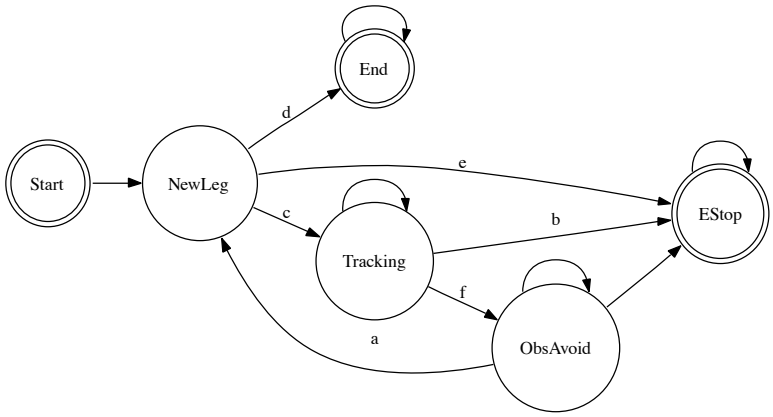

Figure 16: Finite state machine of the sequencing algorithm. The state machine begins execution in the "start" state and has two end states. "End" and "EStop." The conditions of the edge labels are shown in Table 2.

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 2: The conditions of the edge labels for the state machine shown in Fig. 16.

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. 16. Six labels a-f represent conditions on the transition between states described in Table 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.

5 Testbed

5.1 Helicopter and Flight Control

We fitted a Yamaha RMax helicopter (Fig. 17), 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 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+wePilot system at velocities up to 15m/s and weights up to the full payload capacity of the RMax. The wePilot performed flawlessly in every case, despite varying weight distributions.

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 *power-on descent*. 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 wePilot system has no knowledge

Figure 17: The 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	3.115 m
Tail Rotor Diameter	0.545 m
Overall Length (with rotor)	3.630m
Overall Width	0.720m
Overall Height	1.080m
Engine Type	Water-cooled, 2-stroke, 2-cylinder
Engine Displacement	246cc
Maximum Torque	2.6kgm
*(at 1 bar, 35^o C)	

Table 3: Specifications for the Yamaha RMax industrial remote-control helicopter.

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

Table 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.

of these constraints. We added a watchdog program between the output of our navigation system and the input of the wePilot that overrides commands that could cause a power-on descent.

5.2 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 figure 18. 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. 19. 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 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 could increase 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 airborn particles. If the helicopter is flying low in a dusty area, it will be effectively blinded by the particles it kicks up. The sensor could also see clouds of pollen when flying in the 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.

5.3 State Estimation

The Novatel state estimation system provides high-frequency estimates in six degrees of freedom. This estimate is crucial for accurate registration of the ladar data into a coherent world map. As shown in table 5, the state estimation provides position and attitude at 100Hz.

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. For obstacle avoidance,

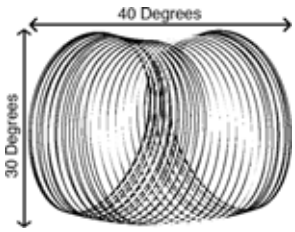

Figure 18: Ladar scan pattern. The small circle is scanned at 22 Hz while it it swept across the field in 0.75s. The laser rangefinder measures distance at a frequency of 64 kHz.

Figure 19: A scene generated from a series of static scans of the environment with buildings, trees and powerlines. The powerlines are visible at 170m.

regular GPS would probably suffice, especially given the range accuracy of the ladar. The map doesn't have to be perfect 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.

5.4 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.

6 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 4.2.4 for details on the training procedure.

6.1 Model

The flight control system described in section 5.1 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

	Update Rate	Accuracy
Position	100 Hz	1 cm
Attitude	100 Hz	1.2 deg
Acceleration	20 Hz	
Covariance	1 Hz	

Table 5: Performance of the Novatel state-estimation system.

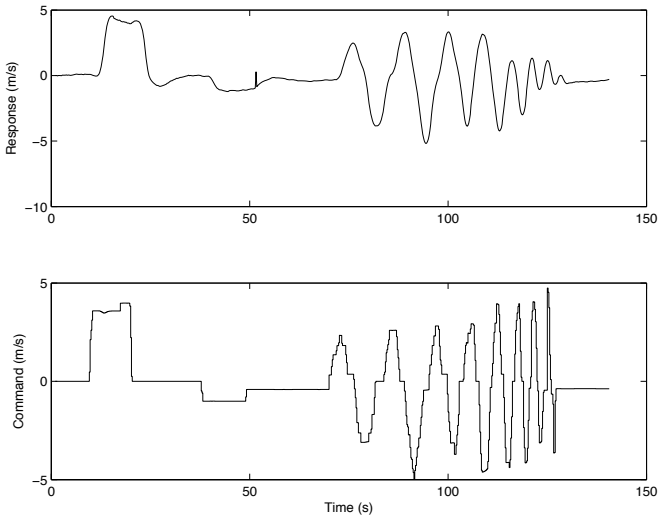

Figure 20: Example result of a frequency sweep experiment for dynamics characterization. This data is used to fit a dynamic model of the system for simulation.

degrees of freedom and we can therefore use a single input single output (SISO) frequency-domain model. The model is given by

$$H_i(s) = \frac{b_0 s^k + b_1 s^{k-1} + \cdots + b_k}{s^k + a_1 s^{k-1} + \cdots + a_k} e^{-st_d}, \tag{41}$$

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.

6.2 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:

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. 20. All of the SISO dynamics could be well approximated by a 2^{nd} order system with two poles and no zeros. Table 6 shows the resulting systems. The final column, FPE, is the Akaike *Final Prediction Error* for the fit.

6.3 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 21 shows the inputs and outputs for each degree of freedom during an actual flight and a flight recreated in simulation using the same

	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 6: Parameters of the SISO dynamic model of the velocity-controlled helicopter

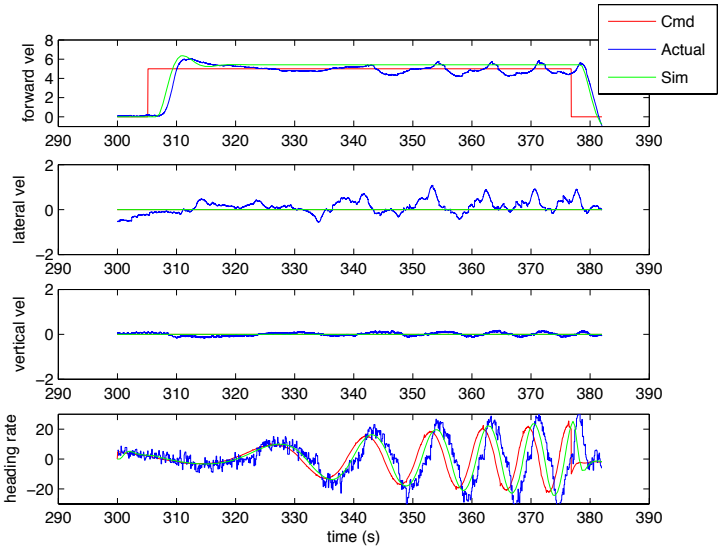

Figure 21: Comparison of simulated velocity to actual telemetry. These graphs show coupling between forward velocity and turn rate with altitude and lateral dynamics. The trajectories for this experiment are shown in Fig. 22.

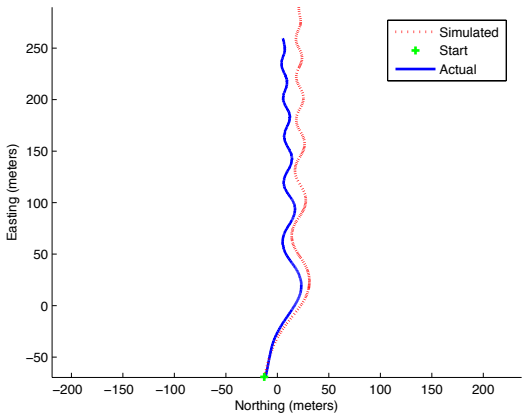

Figure 22: Comparison of a simulated and actual path flown by the helicopter. The general shape of the two trajectories are quite similar, though errors add up over time. The velocities and commands that generated these paths are shown in Fig. 21.

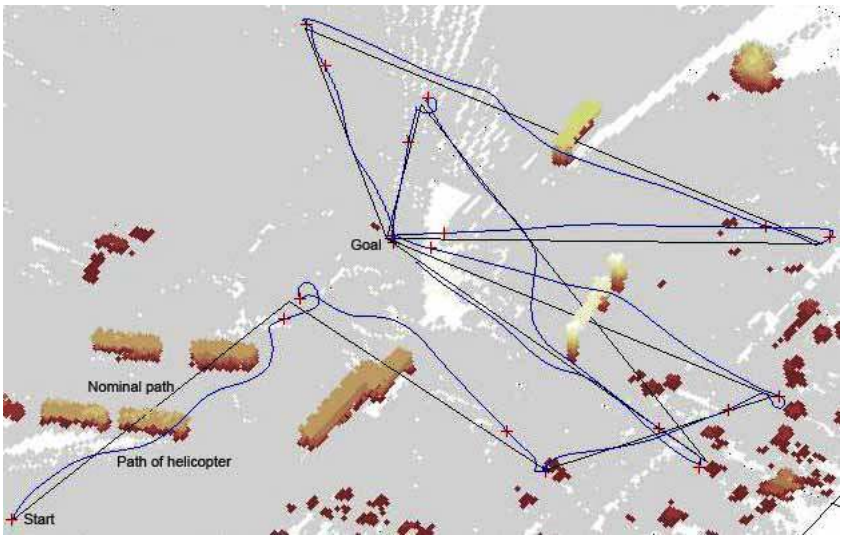

Figure 23: A long flight sequence over various obstacles at the Phoenix test site at $6m/s$. The environment had a series of containers, three poles and a wire as obstacles. In the beginning of the sequence two containers are stacked resulting in an overall height of 17 feet. The last stack of containers is stacked to a height of 25.5 feet.

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 22 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.

Another problem with the modelling 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.

7 Experimental 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. The UAV built the map incrementally as it was avoiding obstacles. The minimum speed was $4m/s$ and the maximum speed was $10m/s$. The connection

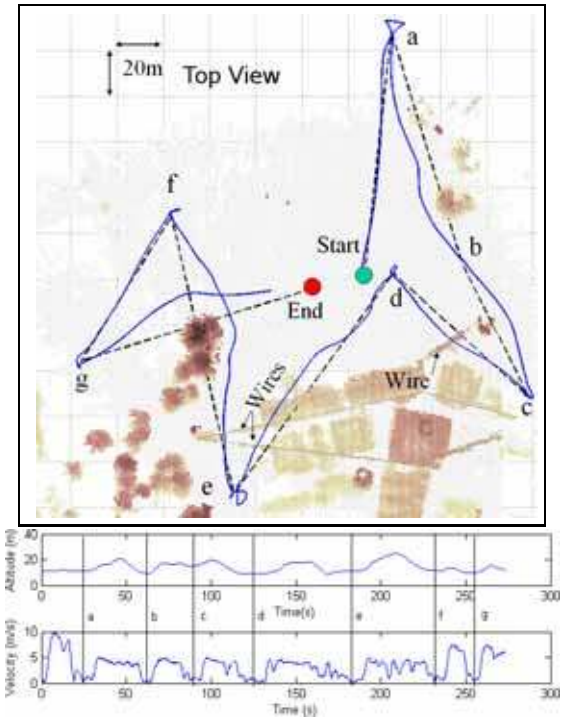

Figure 24: A flight through the McKenna MOUT site at Ft. Benning, GA at 4 m/s. 8 m/s was commanded at the first and 6 m/s at the last two segments of the path.

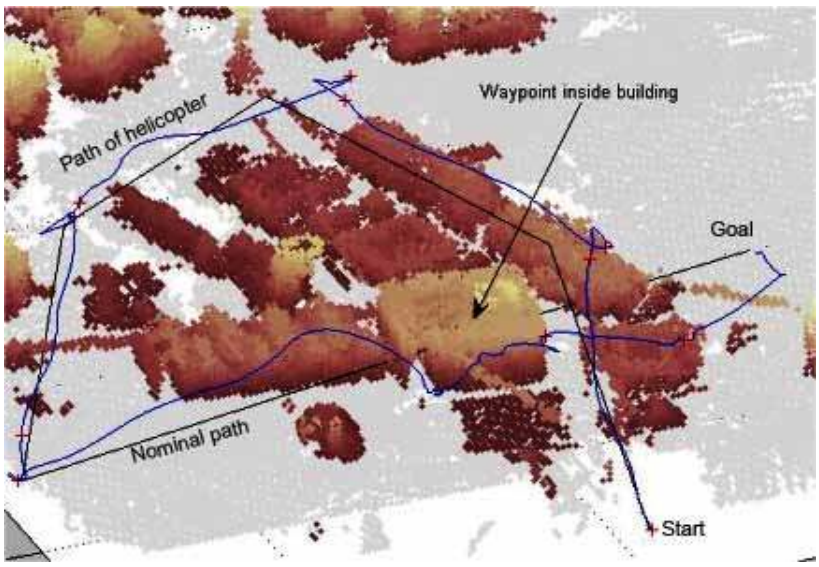

Figure 25: Another McKenna MOUT site demonstration at $4m/s$. This one shows the behavior of the system with an unachievable waypoint located inside a building.

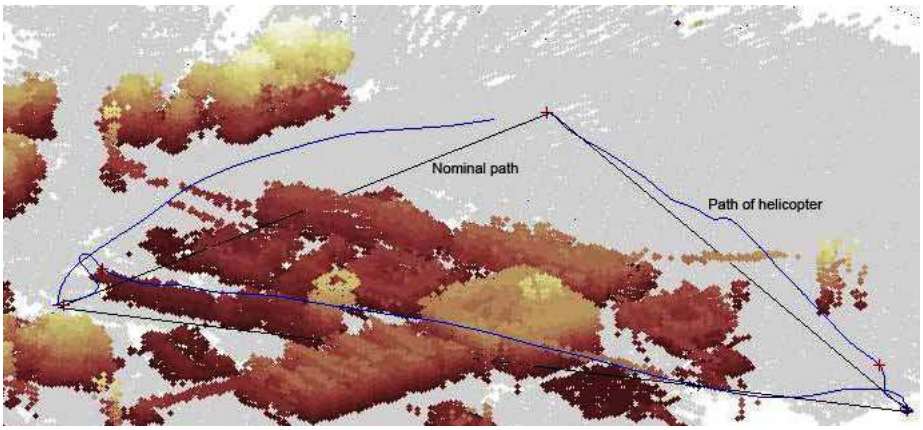

Figure 26: A third party provided a previously unknown waypoint sequence. The plot shows a flight of this waypoint sequence at the McKenna MOUT site at $4m/s$.

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 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. For the training of the local planning algorithm we collected several training paths for our fitting procedure. Even though the parameters found by our fit would have been sufficient to avoid obstacles we tweaked the parameters for a duration of about one day. 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. 23 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 provide 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. 24 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 ($\tilde{8}$ m above ground) is so low that the helicopter is not visible nor audible from the start of the path. Please also see Extension 1 for a video of this waypoint sequence.

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 27, 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. 24, 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. 25 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. 26.

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 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 fast spatial 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. 27. Extension 2 shows a replay of the flight data for both situations.

Figure 28 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 goalpoint 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.

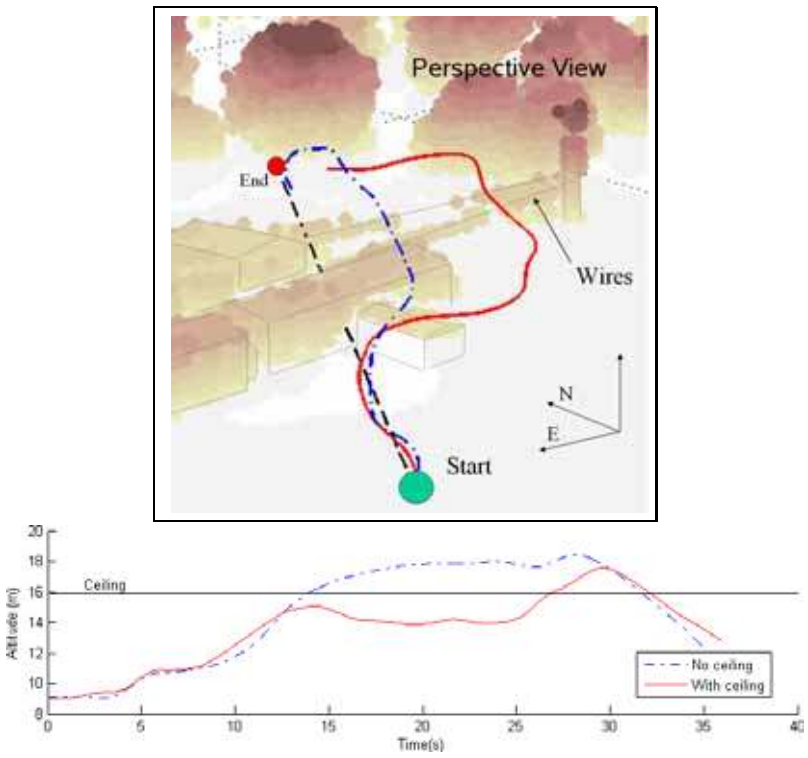

Figure 27: Flying with and without ceiling when specified path is directly through a building. If a ceiling has been specified, the vehicle tries to stay below a specified altitude. On the run with ceiling, the system begins by flying around buildings, but then ignores altitude constraint and climbs to safely clear power lines. Key: Dash-dot→No ceiling. Solid→With Ceiling.

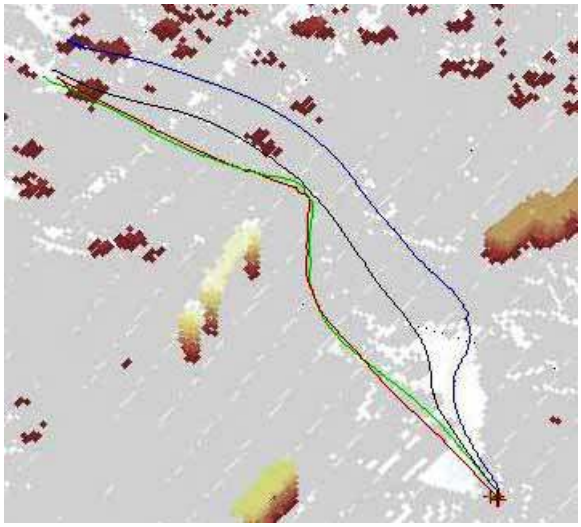

Figure 28: A comparison of the paths taken with different combinations of the local and global planner.Algorithms used from right to left path: Global planner only(blue), local + global planner(black), local + optimized global planner(green), and local planner only(red).

8 Conclusions and Future Work

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 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.

Appendix A: Index to Multimedia Extensions

The multimedia extensions to this article are at: http://www.ijrr.org.

Extension	Type	Description
1	Video	In this experiment the helicopter follows a sequence of waypoints. Obstacles in the path are trees, wires and buildings.
2	Video	A comparison of the effect of setting a soft ceiling for the planning algorithm.

Table 7: Table of Multimedia Extensions

Acknowledgments

The authors gratefully acknowledge the contribution of Samar Dajani-Brown for implementing the Laplacian algorithm; Gary Stevenson from Fibertek for providing the ladar; Henele Adams for his help with interfacing to the Ladar; Alan Touchberry, Tara Schesser and Robert Schley for help with experimentation; Mark Delouis for extraordinary piloting and maintenance skills; and Brad Looney for leadership in the collision avoidance program. This document has been approved for public release (DISTAR Case 7710), Distribution Unlimited.

References

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

Byrne, J., Cosgrove, M., and Mehra, R. (2006). Stereo based obstacle detection for an unmanned air vehicle. In *Proc. IEEE International Conference on Robotics & Automation*.

Call, B., Beard, R. W., Taylor, C., and Barber, B. (2006). Obstacle avoidance for unmanned air vehicles using image feature tracking. In *Proc. AIAA Conference on Guidance, Navigation, and Control*, Keystone, CO.

Connolly, C. I. and Grupen, R. A. (1993). The application of harmonic functions to robotics. *Journal of Robotic Systems*, 10(7):931–946.

Fajen, B. and Warren, W. (2003). Behavioral dynamics of steering, obstacle avoidance, and route selection. *Journal of Experimental Psychology: Human Perception and Performance*, 29(2).

Fajen, B. R., Warren, W. H., Temizer, S., and Kaelbling, L. P. (2003). A dynamical model of visually-guided steering, obstacle avoidance, and route selection. *International Journal of Computer Vision*, 54:13–34.

Foessel, A. (2002). *Scene Modeling from Motion-Free Radar Sensing*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

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

Green, W. E., Sevcik, K. W., and Oh, P. Y. (2005). A competition to identify key challenges for unmanned aerial robots in near-earth environments. In *Proc. IEEE International Conference on Advanced Robotics*, pages 309–315, Seattle, WA.

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

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision, second edition.* Cambridge University Press.

Hrabar, S. and Sukhatme, G. (2003). Omnidirectional vision for an autonomous helicopter. In *Proc. IEEE International Conference on Robotics & Automation*, volume 1, pages 558 – 563.

Hrabar, S., Sukhatme, G. S., Corke, P., Usher, K., and Roberts, J. (2004). Combined optic-flow and stereo-based navigation of urban canyons for a uav. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, pages 3609–3615, Alberta.

Huang, W., Fajen, B., Fink, J., and Warren, W. (2006). Visual navigation and obstacle avoidance using a steering potential function. *Robotics and Autonomous Systems*, 54(4):288–299.

Jackson, Sharma, Haissig, and Elgersma (2005). Airborne technology for distributed air traffic management. *European Journal of Control*, 11(4-5).

Kanade, T., Amidi, O., and Ke, Q. (2004). Real-time and 3d vision for autonomous small and micro air vehicles. In *Proceedings 43rd IEEE Conference on Decision and Control*.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98.

Kitamura, Y., Tanaka, T., Kishino, F., and M.Yachida (1996). Real-time path planning in a dynamic 3-d environment. In *Proc. IEEE/RSJ International Conference on Intelligent Robots & Systems*.

Li, Z. X. and Bui, T. D. (1998). Robot path planning using fluid model. *Journal of Intelligent and Robotic Systems*, 21:29–50.

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, M. C. and Moravec, H. (1996). Robot evidence grids. Technical Report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

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.

Roth, S. A., Hamner, B., Singh, S., and Hwangbo, M. (2005). Results in combined route traversal and collision avoidance. In *International Conference on Field & Service Robotics (FSR '05)*. to appear.

Shim, D., Chung, H., Kim, H. J., and Sastry, S. (2005). Autonomous exploration in unknown urban environments for unmanned aerial vehicles. In *Proc. AIAA GN&C Conference*.

Strelow, D. and Singh, S. (2002). Optimal motion estimation from visual and inertial data. In *IEEE Workshop on the Applications of Computer Vision*, Orlando, Florida.

Trepagnier, P., Nagel, J., Kinney, P., Koutsougeras, C., and Dooner, M. (2006). KAT-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain. *Journal of Field Robotics*, 23(8):509–526.

U. Trottenberg, C. O. and Schuller, A. (2001). *Multigrid*. Academic Press.

Vandapel, N., Kuffner, J., and Amidi, O. (2005). Planning 3-d path networks in unstructured environments. In *Proc. of the IEEE International Conference on Robotics & Automation*.

weControl AG (2006). http://www.wecontrol.ch/.

Zapata, R. and Lepinay, P. (1999). Flying among obstacles. In *Workshop on Advanced Mobile Robots (Eurobot)*, pages 81–88, Zurich, Switzerland.

Zapata, R. and Liepinay, P. (1998). Collision avoidance of a 3d simulated flying robot. In *Proc. International Symposium on Robotics and Automation*, pages 113–120, Saltillo, Coahuila, Mexico.

Zufferey, J. and Floreano, D. (2005). Toward 30-gram Autonomous Indoor Aircraft: Vision-based Obstacle Avoidance and Altitude Control. In *Proc. IEEE International Conference on Robotics & Automation*.