Auton Robot (2009) 27: 147–164
DOI 10.1007/s10514-009-9134-y

Online world modeling and path planning for an unmanned helicopter

Franz Andert · Florian Adolf

Received: 2 February 2009 / Accepted: 3 August 2009 / Published online: 19 August 2009

Abstract Mission scenarios beyond line of sight or with limited ground control station access require capabilities for autonomous safe navigation and necessitate a continuous extension of existing and potentially outdated information about obstacles. The presented approach is a novel synthesis of techniques for 3D environment perception and global path planning. A locally bounded sensor fusion approach is used to extract sparse obstacles for global incremental path planning in an anytime fashion. During the flight, a stereo camera checks the field of view along the flight path ahead by analyzing depth images. A 3D occupancy grid is built incrementally. To reduce the high data rate and storage demands of grid-type maps, an approximated polygonal world model is created. For a compacted representation, it uses prisms and ground planes. This enables the system to constantly renew and update its knowledge about obstacles. An incremental heuristic path planner uses both a-priori information as well as incremental obstacle updates to assure a collision-free path at any time. Mapping results from flight tests show the functionality of onboard world modeling from real sensor data. Path planning feasibility is demonstrated within a simulation environment considering world model changes inside the vehicle's field of view.

Keywords Unmanned aerial vehicle · Stereo vision · Mapping · World modeling · Path planning · Obstacle avoidance

F. Andert (✉) · F. Adolf
German Aerospace Center (DLR), Institute of Flight Systems, Lilienthalplatz 7, 38108 Braunschweig, Germany
e-mail: franz.andert@dlr.de

F. Adolf
e-mail: florian.adolf@dlr.de

1 Introduction

Operations of unmanned vehicles in urban environments are of high interest for surveillance and reconnaissance tasks. However, autonomous and safe operations in such environments with plenty of obstacles are not simple tasks. Many small VTOL UAV systems that are suitable for urban operations still rely heavily on the situational awareness of the remote operator. Since urban scenarios may require the vehicle to fly without line of sight to the operator, it becomes necessary that the vehicle has a sufficient situational awareness to avoid collisions with unanticipated obstacles. To achieve the goal of autonomous flight at low altitude in the vicinity of a-priori unknown obstacles, a collision-free path towards a desired target position must be maintained.

Flying in unknown terrain is related to obstacle avoidance that is often tackled with reactive behaviors. Especially on small aircraft, lightweight vision sensors and optical-flow methods are used (Green et al. 2004; Ruffier and Franceschini 2005; Zufferey and Floreano 2005; Garratt and Chahl 2008). Other research activities improve the obstacle detection capabilities with stereo vision (Hrabar 2008) or laser scanners (Griffiths et al. 2006). Most reactive obstacle avoidance methods yield insect-like behaviors with high accelerations and sharp curves. This is not suitable when carrying fragile payload and requires high performance image stabilizers to capture good quality video sequences.

Larger aerial vehicles allow complete autonomous control due to increased capabilities for carrying more precise avionics and other payload. Smoother and more useable trajectories in obstacle-prone environments are achieved using model-predictive control (Shim et al. 2006), collision cone approaches (Watanabe et al. 2007), or a combination of occupancy maps and behavioral dynamics of steering (Scherer et al. 2007).

However, reactive and local obstacle avoidance methods cannot ensure that a desired target is really reached. This can be solved with global path planning algorithms based on environmental maps. Aerial applications in known environments use probabilistic road maps (Pettersson and Doherty 2006) or rapidly-exploring random tree search (Saunders et al. 2005; Bruce and Veloso 2007). In unknown environments, obstacles have to be detected while flying and to be considered by the path planner. This requires environmental sensing, mapping, and fast path replanning in real-time. A combined approach is presented by Hrabar (2008) where the flight path is initialized based on environmental data and updated with the help of an occupancy grid map built incrementally during flight.

This paper presents an approach to fly autonomously in partially or completely unknown environments. Section 2 introduces the application domain and specifies the requirements that lead to the presented approach. Section 3 covers a mapping procedure based on stereo vision to generate and update a world model in real-time. To maneuver around newly detected obstacles, an anytime path planning approach is presented in Sect. 4. To underline the effectiveness of this work, Sect. 5 presents mapping tests under real operational conditions performed with an unmanned helicopter, as well as a feasibility demonstration of the proposed path planner. The work is concluded in Sect. 6.

2 Autonomous flights in a changing environment

2.1 Context and basic idea

An autonomous flight is initially based on a-priori knowledge of a three-dimensional environment that can contain obstacles. A collision-free path can be specified by the UAV operator that comprises of a set of *waypoints* at which the vehicle has to pass through or stop. Such a set of waypoints defines a *task*, e.g. to explore a specified area. UAV operators have an intrinsic need for reduced mission planning complexity when specifying cost-optimal, collision-free paths. A path planning system automates this translation of user specified sets of tasks into a sequence of waypoints while guaranteeing a collision-free path.

While the vehicle is flying along this initial path, it is gathering new information about obstacles. Probable collisions can be detected and in this case, the flight path has to be updated. Here, the challenge of generating initial and updated paths in potentially unknown areas is divided into two problems:

1. *Mapping*: Interpret sensor data to update the world model.
2. *Path planning*: Plan a path based on a world model and if the world model is updated, repair the path plan efficiently.

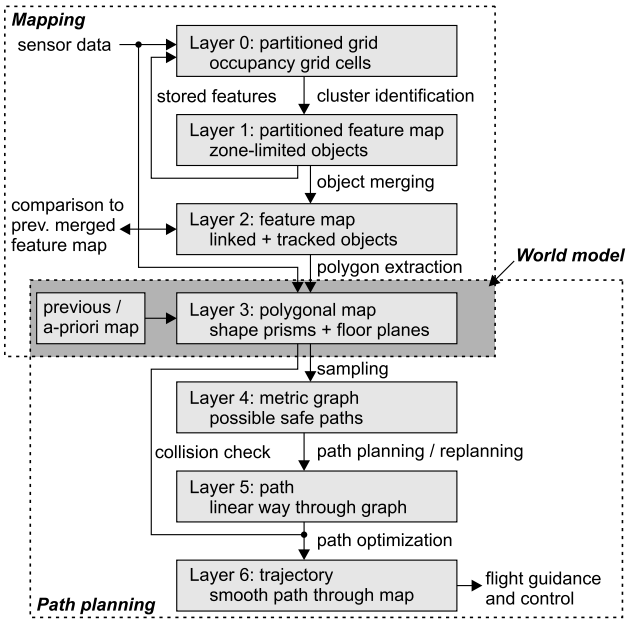

Fig. 1 The general sensor-based path planning process divided into a mapping and a path planning step and different layers of environmental representation

Figure 1 illustrates the overall process that consists of a mapping step that identifies objects, and a path planning step that uses the objects that have been identified or are initially given. Different types of environment representations optimized for specific tasks are introduced throughout this paper. The world model itself (layer 3 in the figure) acts as an interface between the two main steps mapping and path planning. Eventually, the output is a regularly updated and smoothed path that acts as the set point in flight control.

2.2 Representing the environment

The world model presented in this paper must fulfill requirements given by the mapping and the path planning step. As already presented in Andert and Goormann (2008), a primary requirement is to determine the required level of detailing to represent the objects. For path planning in outdoor applications, the important criteria are:

1. small details and knowledge of visual object appearance are not needed,
2. map boundaries must not be limited to a pre-defined area, and
3. real-time updates are important for online replanning to avoid obstacles.

The first criterium implies that it is sufficient to segment the environment into safe and danger areas that must be avoided. Array structures like occupancy grids are very common, but they do not cope with the second criterium since the memory usage is very high in comparison to other

representations. However, grid representations are used for some local map parts as discussed later. Compressed grid structures like octrees or polygonal bounding shapes reduce the memory requirements, and it was chosen to use a polygonal feature-based world model to cope with all of the given constraints. To improve calculation speed as required in the third criterium, the polygonal complexity must be minimized.

The world model is designed towards a compact representation of urban canyons. In a real scenario like a city, objects can have any ground shape from the top view, but it is likely that they have vertical walls. This assumption is made in a lot of mapping approaches, see e.g. Iocchi et al. (2000). For this reason, prism shapes are used here to define danger areas as a kind of bounding box around obstacles. They can deal with any ground shape without dramatically increasing the complexity. If walls are not vertical like roofs, the prism's volume will be much larger than the real object. To improve the modeling of objects, they can be subdivided vertically and represented by multiple prisms. In other words, the world is split into layers in different heights, and each layer has its own polygonal 2D obstacle map. This is sufficient for applications like flight trajectory planning in urban scenarios (Adolf et al. 2007). Additional complexity reduction is achieved by separating the floor from other objects like buildings. The ground is modeled with horizontal planes and can be understood as a height profile map with reduced resolution.

2.3 Adaptive path planning

Numerous approaches transform a path search into a graph search problem using sampling-based metric graphs. However, the sampling-based free space representation affects the path search completeness which is the chance to find a path whenever a solution exists. An appropriate sampling strategy has to maximize this chance. For the sampling itself and when detecting obstructed path segments, a rapid test against collisions with polygons is important. Essentially, the three computationally critical planning steps are the collision detection with polygonal obstacles, the sampling strategy and the graph search. This yields the following requirements:

- Sampling quality: The world needs to be sampled in such a way that path search is approximately complete. Especially in urban terrain it must be possible to find a path through likely occurring narrow passages.
- Anytime path search results: Due to limited deliberation time, the planner must provide a path whenever updates become necessary. A replanning step must always output a collision-free path. Hence, path optimality has a lower priority compared to a rapid path computation.
- Execution time: The configuration space must be updated rapidly.
- World expandability: The configuration space must be expandable for waypoints that exceed the boundaries of the currently known world.

Numerous sampling based approaches use global search graphs representation of free space. The work in Vestka et al. (1996) uses probabilistic road maps (PRMs). In theory, PRMs are probabilisticly complete for random samples. With an increasing number of samples, the probability to find a path converges towards one if at least one solution exists. The work in Plaku et al. (2005) presents a sampling-based road map of trees (SRTs) as hybrid of both, fast one shot path planners using Rapidly-exploring Random Trees (RRTs) and global path planning using a PRM. It guides the sampling of the RRT and is more decoupled than sole road map and sampling-based tree planners. However, this approach introduces a set of seven additional parameters in order to make use of existing knowledge about the world. Since the benefit of SRTs depends on these parameters, it remains unclear whether this set can be generalized for urban scenarios where little is known about the details beforehand. Hybrid approaches like SRTs tend to add the burden of an increased parameter space. Thus, this work focuses on the global, road map based path planning.

As a result, an online path planning approach is presented that is based on polygonal obstacle information and maintains a global road map for anytime heuristic graph searches. A fast cubic polynomial interpolation is used for path smoothing to optimize graph search results by minimizing a heuristic-based arrival time estimate.

A-priori obstacle information is considered during the construction phase of a metric search graph. Using a quasi-random sampling strategy (Branicky et al. 2001) for initial road map construction, a minimum number of vertices is used and path search completeness can be shown, provided that a knowledge of a minimum corridor width exists. During flight, a vertex grid structure is used as a search index for all known obstacle polygons such that rapid, locally bounded search graph updates are achieved. The connection strategy from Hrabar (2006) is used to search for neighbor vertices and to determine feasible edge costs to these vertices. An efficient collision detection (Langer 2006) is used to perform rapid collision checks for path segments and graph vertices.

To account for the inherent trade-off between either longer computation time for optimal paths or instant yet suboptimal paths, *Anytime Dynamic A** (Likhachev et al. 2005) is used to search for paths online. If computation time and world model update frequency permit, the graph search yields an optimal solution over time.

3 Online world model updating

This section describes the process of how the world model for path planning is updated. A geodetic map is automatically built onboard the vehicle using stereo-based range measurements and GPS/INS-based vehicle localization. Output of the presented algorithm are environmental updates that fit with the presented world model definition.

3.1 Combining grid and feature maps

Optimal sensor data fusion requires an environmental representation that is suitable for fast incremental updates and error handling like noise reduction. Occupancy grids have turned out to be an efficient way to handle that task. To profit from their advantages in the world model update procedure, grid and polygonal representations are combined and the mapping process works as follows:

1. Represent the area around the vehicle by an occupancy grid in the map.
2. If a new grid is allocated or the grid is extended, check whether previously stored features can be inserted.
3. Insert the actual sensor data information into the grid.
4. Find clusters of occupied grid cells and mark them as single obstacle features.
5. Find out which obstacle features are new and which are updates of objects that have already been identified in the previous loop cycle. Preexisting objects may also be removed.
6. Calculate the shape of each new or updated feature.
7. To insert the next sensor data, go back to step 1.

This approach uses different map types as already illustrated in Fig. 1, Sect. 2.1. The occupancy grid map for sensor inputs (layer 0) is partitioned into zones and each zone is searched for obstacles separately (layer 1). Next, a map with separate obstacles, but without zone separation is generated (layer 2), and finally, the prism shapes are extracted out of them (layer 3). Grid resolution and zone sizes are user-defined but may not change over time when processing image series.

3.2 Grid mapping

The occupancy grid is created incrementally with classical methods using the sensor data and the vehicle's position and attitude. The world is rasterized with a 3D grid where each cell stores the log odds probability of being occupied.

3.2.1 Sensor data interpretation

Basically, every depth image pixel acts as single distance measurement that gives information about objects and free space. The pinhole camera model and perspective transformations are used to project depth image pixels into object point coordinates relative to the camera. Object points are transformed into geodetic coordinates by using a manually measured camera alignment on the vehicle and the global vehicle position. The vehicle is localized using differential GPS, inertial and magnetic sensors that are combined with an EKF-based navigation solution. The navigation filter provides a relative localization accuracy of estimative 0.2 m horizontal and 0.5 m vertical in open terrain. Here, the vehicle is flown in urban terrain without narrow or deep canyons where the risk of possible satellite signal dropouts as depicted by Morales and Tsubouchi (2007) is assumed to be low. To compensate for GPS failures and eventually to fly without any satellite navigation, vision-aided flight state estimation based on optical flow is developed (Wu et al. 2005). Such optical navigation is not used here, but it is of high importance for future research. As illustrated by Fig. 2, three frameworks are used for the required coordinate transformations.

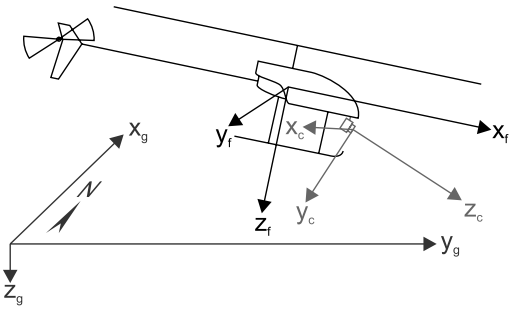

Fig. 2 Camera-fixed (index: c), vehicle-fixed (index: f), and geodetic (index: g) coordinate systems

In the measurement model described here, a depth image pixel acts as a measurement along a ray through the camera center and the object coordinate, defined by the pixel coordinate, its depth value, and the geodetic camera pose. To draw a line with occupancy values into the grid map, a 3D version of the method from Bresenham (1965) is used. Along this line, grid cells are filled with log odds of occupancy probabilities, dependent on the depth z and parameterized by the measured depth z_c. It is

$$l(z, z_c) = \ln\left(\frac{p(z, z_c)}{1 - p(z, z_c)}\right) \tag{1}$$

with occupancy probabilities

$$\begin{aligned} &p(z, z_c) \\ &\quad = p_{\text{occ}}(z, z_c) \\ &\qquad + \left(\frac{k}{\Delta z_c \sqrt{2\pi}} + 0.5 - p_{\text{occ}}(z, z_c)\right) e^{-\frac{1}{2}\left(\frac{z - z_c}{\Delta z_c}\right)^2} \end{aligned} \tag{2}$$

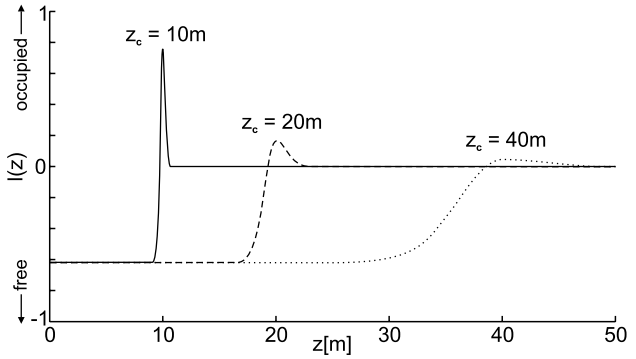

Fig. 3 Measurement Model. Occupancy values with the parameters $k = 0.1$ and $p_{\text{free}} = 0.35$, showing measurements of 10 m (*solid*), 20 m (*dashed*) and 40 m (*dotted*)

where

$$p_{\text{occ}}(z, z_c) = \begin{cases} p_{\text{free}} & \text{if } 0 < z \leq z_c; \\ 0.5 & \text{if } z > z_c. \end{cases} \tag{3}$$

Stereo-based measurements offer an error $\Delta z_c \propto z_c^2$ that is considered in the sensor model with depth-dependent occupancy significance and object thickness. The plot in Fig. 3 shows the resulting occupancy function $l(z, z_c)$ for different measurements.

Depth image processing takes advantages from image pyramids to improve calculation speed. Especially for near measurements, neighboring pixels with similar depth values are probably leading to points of the same grid cell. There is no need to draw such overlapping lines into the map. Reducing the resolution will provide fast image processing and a good approximation in comparison to a processing of all depth image pixels. Result is a temporal occupancy map dependent on a single image.

Incremental map building from image sequences starts with an empty map. Occupancies are initially unknown and all cell values set to zero, meaning an occupancy probability of 0.5. Now, the temporal map from a single image is inserted by adding the cell values. With the fusion of multiple sensor samples, the cell probabilities of being occupied increase if an obstacle is measured multiple times. Vice versa, the measure of free space will decrease the occupancy probability value of a cell. Obstacles as well as free space become more significant and noise is reduced. This method is very common and well-documented in the literature (Moravec and Elfes 1985; Konolige 1997; Thrun 2002).

3.2.2 The temporary zone-partitioned occupancy grid

Since grids are usually stored as continuous data blocks in the computer memory, the boundaries of the map must be known a priori. As already mentioned, one requirement for the world model is unlimited boundaries so that the overall size can only be assumed and an extension of the map must be provided. If the vehicle moves outside these preliminary boundaries, extensive reallocation or shifting methods must be applied. To avoid shifting a large number of map cells when moving, the implementation divides the global map into cuboidal zones. Their position is fixed. Map extension is done easily by adding new zones.

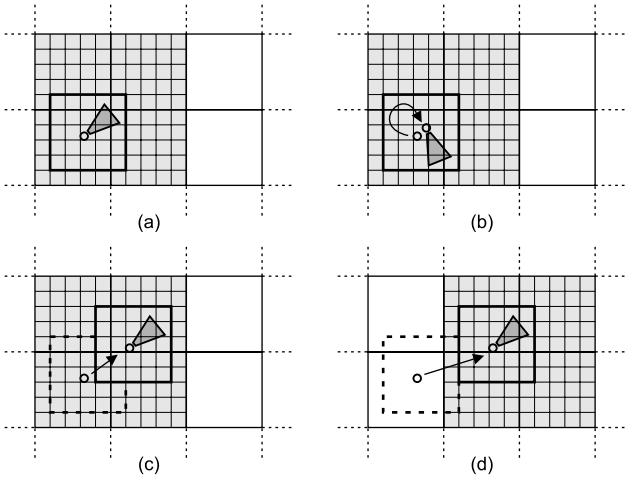

Fig. 4 2D view of the global map that is divided into zones. For zones around the actual camera position, an occupancy grid representation exists

Appending zones will be limited to the available memory. In the presented approach, the occupancy grid is only used for sensor fusion and will be converted to a feature map as described in Sect. 3.3. Hence, a temporary grid map is sufficient and the memory limitation is negligible. The data fusion of the map with one image will only affect the grid cells in the sensor range, i.e. inside a small environment of the actual position. There is no need to store grid-based information outside this environment. This is implemented by simply removing zones if not needed anymore.

Figure 4 illustrates how the zone partitioning works. The actual camera and local map position is shown in (a), the other graphics show possible effects on the map, caused by the next measurement. Often, rotations (b) or movements (c) will not have an effect on the zone boundaries. But if the movement is large enough so that boundaries are crossed (d), new memory is allocated for these zones. Grid information is discarded for zones that fall outside the immediate vicinity. The size of each zone is chosen dependent on the sensor range so that new sensor data can change a maximum of eight zones inside a $2 \times 2 \times 2$ zone cube of the global 3D map. Only these zones are represented by an occupancy grid array.

3.3 Feature mapping

3.3.1 Extracting and inserting features

In every map zone where a grid exists, features are detected by segmenting the grid into occupied and free areas, applying a threshold of minimal occupancy probability to each cell. Here, the threshold is set to a log odds value of 1.0 which implies that unknown space is not included into the object map. A single object is a set of occupied cells that are connected in a 6-neighborhood of the 3D array. These objects are recognized with a flood fill algorithm. By saving the minimal and maximal values of the x, y, and z-coordinates of cells belonging to the object, the bounding box is calculated and put into the global map as it is illustrated in Fig. 5. Unlike the cell array, these boxes are saved when the occupancy grid moves and the features are available to further applications, independent of the existence of a grid in that zone. The features are detected separately for each map zone, so that a bounding box will always be inside the boundaries of one zone. Objects that belong physically to more than one map zone are represented by multiple features.

Saving the bounding boxes leads to a great data reduction since they do not contain the object shapes. But they will only be a useful object representation in rather unoccupied environments, e.g. outdoor scenarios with large free spaces and single obstacles like trees. Free paths inside the bounding boxes of tunnel or canyon walls will not be found in the feature map since the whole box is regarded as occupied in the feature map. As an improvement, polygonal shape detection algorithms are applied, and the shape of each object is stored together with the box.

To complete the exchange of elements between grid and feature-based map representations, single objects of the global map must be inserted into the grid. This is done when a new grid zone is allocated and some world map objects exist there (Fig. 6). First, the global map is searched for bounding boxes. Then, those grid cells that intersect with an object shape are marked as occupied by giving them a log odds probability value equal or greater than the segmentation threshold. The other cells remain unexplored. When removing and re-allocating a map zone, the essential grid-based information about objects is recovered while discriminable occupancies and free space information are removed.

3.3.2 Merging and tracking objects

Single objects found in the temporary grid zones are limited to the zone boundaries since each zone is processed separately. To build an output map with a reduced number of objects, the features of different zones are merged if a connection exists as illustrated in Fig. 7. First, it is checked whether the bounding box of an object is located at a zone boundary. If another object box is located at the same boundary from a different zone, the cell shapes of both objects are tested for connection. Connected objects are linked together, and the linkage is not limited to only two objects.

After an update step where sensor data is inserted, the zones are checked for obstacles again without reference to the previous state. Since it is useful to know which specific object has been updated with the actual sensor information, the linked objects are tracked. Differential updates of the polygonal world model become available throughout the system.

Object tracking is done as follows:

1. Calculate the center of mass of each linked object.
2. For each linked object, try to find a matching object in the map with linked objects of the previous state. This is the object with the nearest center of mass determined by the Euclidean distance (Prassler et al. 2000). To avoid the matching of objects that are too far away, distances above a threshold will not result in a match.

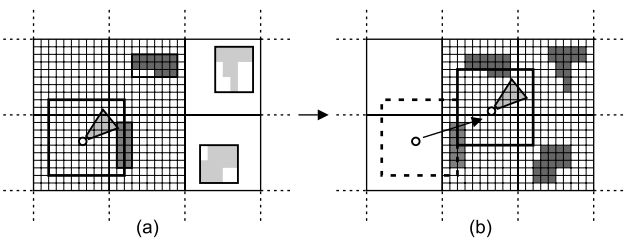

Fig. 6 Inserting features to the grid if there are objects inside a new zone

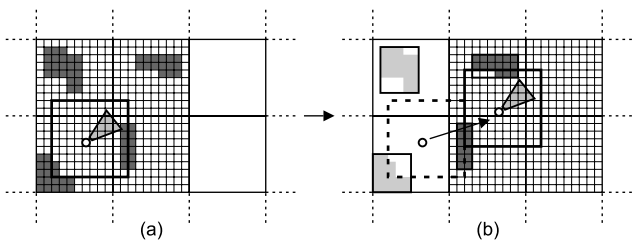

Fig. 5 Features are stored when the grid data is discarded due to helicopter movement

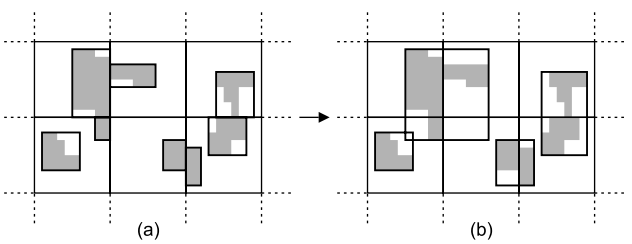

Fig. 7 Simplified 2D view of merging tangent objects in the partitioned map (**a**) to linked objects (**b**) that are not partitioned into zones

3. If no match exists for an object of the actual state, give it a new unique ID. Otherwise, copy the ID from the matched object.

With object tracking, each linked obstacle in the world gets a unique ID that remains constant over time, if the tracking is successful. The merging and tracking step can be skipped for linked objects whose coordinates are completely out of the zones where a grid exists since there will be no update.

Due to sensor updates it is possible that a linked object becomes split. Usually, one of the new objects gets the old ID, the others receive the new IDs. If multiple linked objects are merged together over time, one ID is kept and the others removed.

3.3.3 Horizontal shape slicing

The approach presented in this paper models the shape of each linked object with a prism (see Sect. 3.3.4) with horizontal bases. Since a bounding prism of complex shapes may be too rough, the cell-based shape $s(x, y, z)$ is partitioned into horizontal slices with the height of one cell. Similar slices are merged and a polygonal shape prism is calculated for each multi-slice.

A single slice in height z is denoted as $s_z(x, y)$ with $s_z(x, y) = s(x, y, z)$. Without loss of generality, z is valid from 1 to n. Similar consecutive slices are put together to multi-slices S using Algorithm 1.

The result is a set of multi-slices $\{S_1, \ldots, S_m\}$, see Fig. 8. The user-specified threshold t used in Algorithm 1 controls the maximum degree of similarity to merge similar successive single slices. The extreme case $t = 0$ puts every single slice into a separate multi-slice and $t = \infty$ merges all single slices together. In the tests presented in this paper, a value of $t = 100$ is used which provides suitable results.

Algorithm 1 Create m multi-slices from n single slices

set $i = 1, z = 1$
create first multi-slice S_i: $S_i = \{s_z\}$
for $z = 1$ to $n - 1$ **do**
  if

$$\sum_x \sum_y |s_z(x, y) - s_{z+1}(x, y)| < t$$

  then
    append slice to the current multi-slice $S_i = S_i \cup \{s_{z+1}\}$
  else
    $i = i + 1$
    create a new multi-slice S_i: $S_i = \{s_{z+1}\}$
  end if
end for
return $m = i$ and $\{S_1, \ldots, S_m\}$

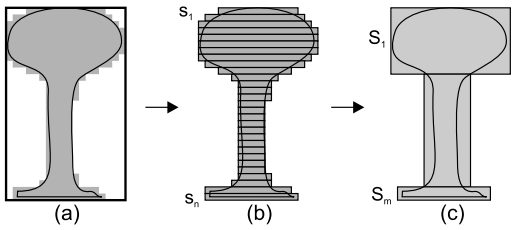

Fig. 8 Grid shape and bounding box of linked feature (**a**), single slices (**b**) and merged multi-slices (**c**). Side view of a tree as an example

In the next step, the 2D shape is calculated for each multi-slice S_i $(1 \leq i \leq m)$. Occupied cells of a multi-slice are calculated through the logical disjunction of all single slices s_z inside S_i. It is

$$S_i(x, y) = \begin{cases} 0, & \text{if } \sum_{s_z \in S_i} s_z(x, y) = 0; \\ 1, & \text{otherwise.} \end{cases} \quad (4)$$

3.3.4 Extraction and approximation of the polygonal shape

Now, algorithms developed for binary images can be applied to a multi-slice S_i since its shape is represented by a binary 2D array. Figure 9 illustrates the process. The polygonal extraction is done in two steps. First, the contour is calculated with a tracing algorithm (Ren et al. 2002). The main idea is to start at one edge pixel and search incrementally for the next edge pixel until the whole contour is covered. Its output is a list of pixels sorted counterclockwise for outer contours and clockwise for inner contours. Multiple lists are possible. Each contour pixel can be interpreted as a polygon vertex by using the corresponding grid cell center coordinate.

The second step is for data compression and to accelerate later applications. A polygon with a lower number of vertices is calculated here with the approximation algorithm presented by Ramer (1972). It is parameterized by a maximal distance value that specifies the accuracy of the algorithm. Every point of the original shape has this maximal distance to the lines defining the approximated shape. In contrast to the initial contour polygon with all traced pixels, the resulting polygon includes only corner pixels of the contour.

This 2D polygon acts as the ground shape of the right prism that is calculated for each S_i. The prism's height and vertical position are determined by the span of single slices $s_z \in S_i$. As seen in Fig. 9, the prism shapes can be smaller than the grid-based shapes since the center coordinates of the grid cells are used for the shape vertices and an approximation is performed. In obstacle avoidance applications, a safety distance to the objects must be larger than half a cell size plus the approximation accuracy to ensure that the grid cells of each object are completely covered by the polygon hull.

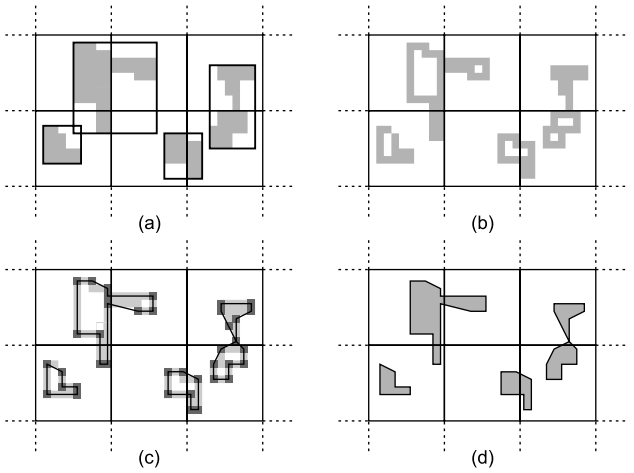

Fig. 9 Shape Extraction, top view. Cell-based shapes (**a**), contours (**b**), approximated polygons with its vertices (**c**) and final ground shapes (**d**)

3.4 Ground planes

In addition to the prism-based shape detection of obstacles, the ground profile is extracted out of the sensor data. This simplifies the resulting world model since the floor will not be a shape prism that has to be tracked.

To account for non-flat terrains, a locally bounded ground plane estimation is performed. These ground plane boundaries are defined by the map partitioning as described in Sect. 3.2.2 but without the vertical separation. This leads to rectangular 2D zones in the x-y-plane where each zone can have a different height. The floor plane detection is similar to classical Hough Transform based approaches, e.g. Okada et al. (2001), where only the horizontal planes are searched.

The actual sensor data leads to a cloud of points (x, y, z) in global coordinates. The vehicle's position must be known and a calibrated pitch and roll angle measurement is assumed to ensure that a horizontal ground will lead to a horizontal plane in map coordinates. A sampled histogram of all z-values of these points is built. It acts as a geodetic 1D occupancy grid created separately for each zone where points are inserted according to the sensor model of section Sect. 3.2.1. The sampling should be more precise than the 3D occupancy grid. Since all measurements of a horizontal plane have approximately the same geodetic z-value, planes lead to significant peaks in the histogram and can be detected there. If more than one plane is found, the floor is the one with the lowest height, i.e. with the largest z-value. There will be no ground plane if no peak with a high confidence is found.

A height histogram is calculated for each zone by accumulating these sensor data elements that lead to object points in that zone. The histogram is built incrementally over time so that multiple sensor updates can be stored in one zone if the vehicle stays there. Like the prism shapes, the ground plane map is initialized empty and built incrementally during flight. The resulting rectangular planes act as a polygonal obstacle for path planning in the same way as the prism shapes do. Ground plane extraction is done before occupancy grid mapping directly from depth image data. To improve the 3D map calculation speed, image pixels that lead to the ground level or below are not taken into account for 3D mapping since they are already interpreted as objects.

4 Online path planning

The following sections present a global, road map based path planning and discuss its sampling strategy, a feasible graph search method and a fast method for collision checks.

4.1 Road map sampling strategy

The effectiveness of a road map depends crucially on its coverage and connectivity properties. A good sampling aims for a good representation of the free configuration space. The objective is to fly through narrow passages in free space (e.g. road corridors). Effectively, the sampling has to guarantee the completeness for path searches. Moreover, during online planning, new obstacles may cause new narrow passages through which no path can be found. Even worse, it may break the road map's connectivity when the finish point lies in a different graph component than the vehicle's current position. That's why the sampling is of a high importance for the overall path search.

Common PRM implementations use random machine functions to sample the configuration space. The problem with a randomization is that any machine implementation generates a sequence of pseudo-random numbers. This can lead to relatively large areas of free space that contain no samples, while other areas are overcrowded with samples (e.g. Fig. 10b).

A useful measure for the sampling quality are two metrics, dispersion and discrepancy. The discrepancy measures the largest volume estimation error. The dispersion (here: Euclidean metric) can be considered as the radius of the largest sphere that fits into free space between samples. A low dispersion can be used to ensure that any corridor of a certain width (e.g. street width) will contain samples.

It can be shown that a regular grid has an optimal dispersion but poor discrepancy and is known to be prohibitive for path search in high dimensions ($\geq$3) (LaValle 2006). A PRM achieves better search performance in high dimensions due to its irregular sampling neighborhood structure. According to visibility properties in Hsu et al. (2006), this results in suboptimal visibility. The machine random function used yields to a non-optimal dispersion and a non-optimal discrepancy. The path planner based on a Quasi-random

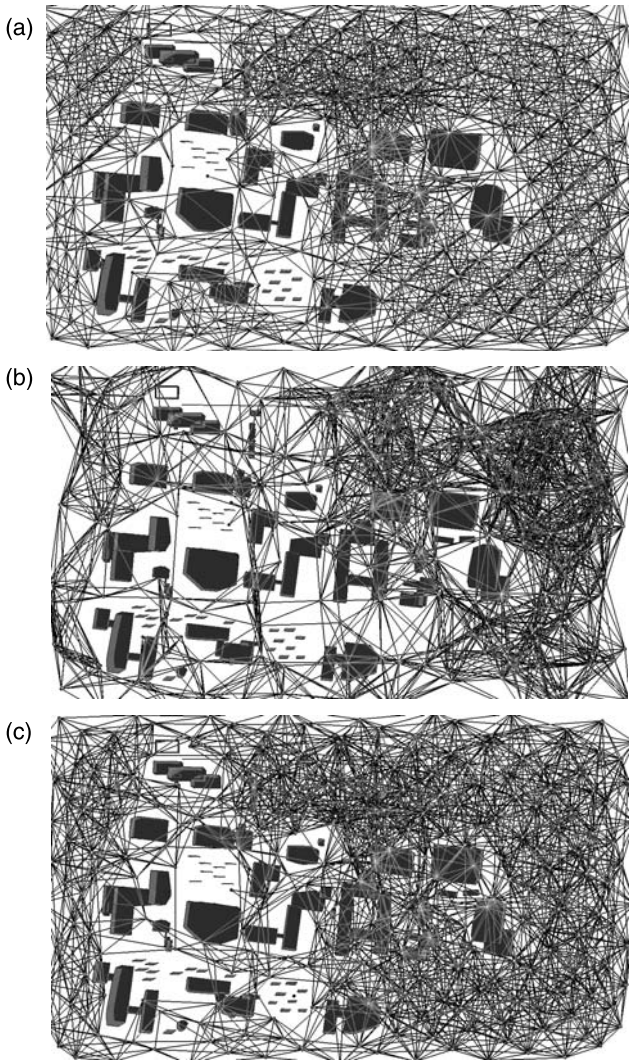

Fig. 10 Sampling strategies for the test scenario in top view, (**a**) LRM, (**b**) PRM, (**c**) QRM-Hammersley

Roadmap (QRM) in Branicky et al. (2001) distributes sample points using a quasi-random distribution of graph nodes. It provides an optimal dispersion and an optimal discrepancy. Furthermore, for the QRM it was empirically observed to provide similar or better performance in terms of the number of nodes required to generate successful plans. The same work also highlights the use of Halton/Hammersley sequences to produce a regular neighborhood structure along non-orthogonal axis. This yields to an optimal dispersion with near-optimal discrepancy.

Literature shows different comparison results between deterministic and probabilistic sampling sources. In Hsu et al. (2006) it is concluded that high computational costs for a high number of dimensions eliminate the advantages of deterministic sources over pseudo-random sources. Compared to path planning in robot manipulation tasks, three dimensions can be considered to mark the lower boundary search space complexity. Thus both, LRM and QRM can be suitable sampling strategies to sample nodes for path search.

The next section evaluates suitable graph search approaches that can cope with changes in a graph.

4.2 Graph search

Searching in a road map attempts to find a sequence of state transitions through a graph from the initial state to a goal state, or it determines that no such sequence exists. As discussed before, global path search algorithms have the advantage to be generally complete and to be able to provide optimal paths. In this context, the path is optimal if the sum of the transition/edge costs is minimal across all possible sequences through the graph. If during the traverse of the path one or more edge costs in the graph are outdated, the remaining fraction of the previously planned path may need to be updated. As stated before, the immediate output of a feasible result is more important than cost optimality.

From a graph search perspective, numerous methods exist that solve path replanning problem using an incremental heuristic search (*assumptive planning*).

The basis for many recent approaches is still A* (Hart et al. 1968) which provides good runtime complexity and is theoretically well understood.

Further optimizations of A* towards anytime path results have been developed. Anytime path search algorithms operate under time constraints and improve their solution over time. *Anytime Repairing A** or *ARA** (Likhachev et al. 2003) provides sub-optimality bounds for each successive search and has been shown to be efficient. *Anytime Dynamic A** or *AD** (Likhachev et al. 2005) combines the anytime property of ARA* and the plan repair concept from D*-Lite. It provides bounded solutions in partially-known or dynamic environments by reusing previous solutions in the same way as ARA*. It repairs invalidated solutions in the same way as D* Lite (Ersson and Hu 2001), while being able to provide anytime solutions.

A number of graph search algorithms exists that addresses dynamic changes in the edge cost of a graph. Furthermore, anytime capable solutions exist that repair plans fast over time. However, ARA* does not handle changing environments. For example, if the vehicle holds incomplete or imperfect initial information, then the solution will need to be repaired. Currently that can be made only by highly-inflated versions of D* or D* Lite. AD* tends to be about 10 percent more costly than the optimal path. However, it can cope with time constraints and imperfect initial information. It is able to improve and repair its solution over time.

4.3 Approach

The findings in the previous sections yield the Q-PRM/LRM method for sampling and road map construction. Furthermore A* is used during the initial path search phase. Since

the changes to the graph are happening within the field of view of the vehicle's obstacle sensor, the effects on the graph are limited to be close to the vehicle. Hence, using an incremental replanner will be more efficient than planning from scratch. Additionally, an anytime planning is desired such that an anytime incremental replanner such as AD* is the plan search and repair method of choice. The next sections describe the collision checking method that is used throughout all stages of the path planner. The successive sections describe the path search enhancements for rapid world model updates, support for non-linear path geometries (e.g. splines) and waypoints exceeding the boundaries of the known environment.

4.3.1 Collision checking with polygons

Besides the sole path computation, collision checks are known to be computationally complex operations. During the road map construction (learning phase) and with new polygons at world model updates, the path planning has to deal with the cost which is incurred when connecting nodes. Since the world model uses an arbitrary polyhedral obstacle model, each polygonal face of an obstacle is decomposed into triangles. Thus and without loss of generality, obstacles are constituted by a set of plane triangles, arbitrarily oriented in three-dimensional space. All polyhedral obstacles are assumed to be closed surfaces, and also assumed not to overlap. The helicopter is modeled as a sphere around its center. The sphere radius is the safety distance to be maintained during planning (here: twice length of the rotor blades plus expected worst case path deviation).

The collision detection presented by Langer (2006) is used which comprises two different detection methods, *TestNodeInFreeSpace* (Algorithm 2) and *TestPathIsCollisionFree* (Algorithm 3). *TestNodeInFreeSpace* draws a straight line from configuration point Q to a point Q_n exceeding the bounds of the current polygon set, e.g. at an altitude above the maximum allowed for flights. This line is tested for connection with nearby triangles. If this line crosses none or an even number of triangles then Q lies in free space, otherwise it is contained within an obstacle. Based on work in Eberly (1999), the methods *distanceTri*, *distanceLines* and *TestTriangleIntersection* reformulate a triangle $T(P_A, P_B, P_C)$ into barycentric coordinates, in order to minimize the squared distance function of Q to a point in the triangle. Note that for the method *TestTriangleIntersection* a full minimization is not necessary when the minimum lies in the interior of the function domain. If their interiors cross each other and this distance is zero, the shortest distance between a point in the line segment and another in the triangle is the one between points in their interiors.

Algorithm 2 Test if node Q is in free space

Require: $Q.dist = \infty$
  $Q_n = Q$
  $Q_n.z = maxHeight(C) + 1$
  $num_c = 0$
  for all $triangle(P_A, P_B, P_C)$ *IN* $neighborhood(Q)$ **do**
    if $TestTriangleIntersection(Q, Q_n, P_A, P_B, P_C)$ **then**
      $num_c = num_c + 1$
    end if
  end for
  if num_c *is even* **then**
    for all $triangle(P_A, P_B, P_C)$ *IN* $neighborhood(Q)$ **do**
      if $distanceTri((P_1, P_2), (Q, Q_n)) < d_s$ **then**
        return false
      else if $distanceTri((P_1, P_2), (Q, Q_n)) < Q.dist$ **then**
        $Q.dist = distanceTri((P_1, P_2), (Q, Q_n))$
      end if
    end for
    return true
  else
    return false
  end if

Algorithm 3 Collision test of path segment $[Q, Q_n]$

Require: $dist_{min} = min(Q.dist, Q_n.dist)$
  for all $triangle(P_A, P_B, P_C)$ IN $neighborhood(Q)$ **do**
    if $TestTriangleIntersection(Q, Q_n, P_A, P_B, P_C)$ **then**
      return false
    else
      for all $segment(P_1, P_2)$ IN $triangle(P_A, P_B, P_C)$ **do**
        if $distanceLines((P_1, P_2), (Q, Q_n)) < dist_{min}$ **then**
          $dist_{min} = distanceLines((P_1, P_2), (Q, Q_n))$
        end if
      end for
    end if
  end for
  if $d_s < dist_{min}$ **then**
    return true
  else
    return false
  end if

4.3.2 Initial path search

The initial path plan is found using a straight forward implementation of a road map based path planner. In addition to the original road map approach, this implementation is enhanced with a world model index and a spline-based path smoothing. The index structure bounds the search for affected regions in the world in case of adding a polygon to

the world model. The path smoothing assures that the path geometry does not contain unnecessary waypoints and sub-optimal heading changes. Thus, the road map-based path planning has three main phases:

1. A pre-computation step during which the road map is generated.
2. A query step that performs a graph search in the generated road map and outputs a sequence of consecutive, linear path segments.
3. A post-processing step that smooths the linear path segments.

In phase 1, an a-priori map is loaded into the world model. If available, known obstacles (e.g. buildings, line of trees) are represented as set of approximately dimensioned polygonal objects, including "no-fly" zones of infinite height. The graph is stored as a node set N and a weighted edge set E. First, N is filled with n quasi-randomly selected (e.g. Halton/Hammersley sequenced) nodes in 3D space. Figure 10 shows how different sampling sequences affect the coverage for a given urban test scenario. This top view shows the difficulty that there are street corridors and wide open areas on the right. The low maximum flight height of 25 m results in sparse sample distribution and fewer connections compared to the open areas. The maximum allowed flight height is approximately at the height of the buildings such that few buildings can be crossed.

Hence, for the same number of samples, LRM and QRM spread and connect more samples in areas between buildings. Again, this underlines that urban scenarios necessitate an efficient sampling.

These nodes are connected generating the set E. For each node Q a set of neighbor nodes is chosen and each pair of nodes (Q_n, Q) is connected using a local, straight line (see Algorithm 3). The collision tests for the local path planner are performed only for the nodes within a predefined maximum distance. The nodes in our representation do not have a heading information so that straight line connections are sufficient. If the local path has no collisions, both nodes and the path cost (Euclidean distance) are added to E. To optimize the search for graph edges to be updated, a vertex grid is added to the world model that has coarse resolution of 10 meters per cell. This extension is used to accelerate the search for affected graph edges, in case an obstacle is added or altered during the online path planning phase. If a polygon intersects a cell in this grid, it is added to a list of polygons covered by this cell. Note that the list remains empty if there is no polygon and thus no memory is occupied for free space.

In phase 2, the current position of the vehicle is added as a node and connected to the search graph. Additionally, the set of user-defined waypoints has to be connected. Any set of waypoints within a task that cannot be connected within a maximum neighborhood is ignored. Moreover, a necessary condition for exploration-type mission is the path planning into the a-priori unknown space of the world model. The example in Fig. 11 shows how task B exceeds the current world boundaries. First, the nearest collision-free node to every task node is determined. From this node a new sample set C is added at the maximum neighborhood distance. If possible, these nodes are connected to other nodes in the existing search graph. Additionally, the vertex grid is extended such that the new task sets the new boundaries of the known world's vertex grid. This operation is performed without noteworthy computational effort since no obstacles are known for the unexplored space and thus all polygon lists are an empty set. This second phase ends with the path planner's output of a set of linear path segments.

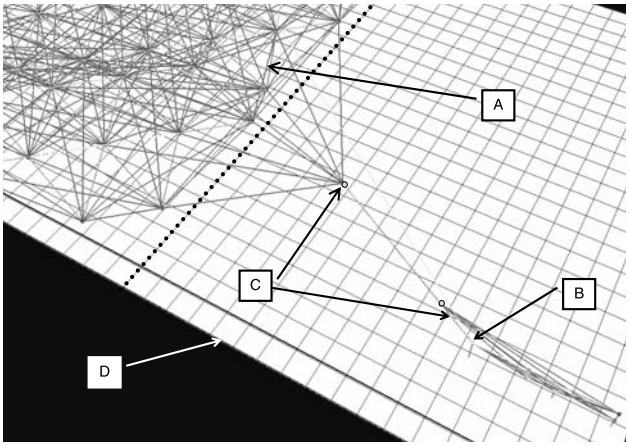

Fig. 11 Handling tasks exceeding the boundary (*dotted line*) of the known environment: (*A*) offline search graph, (*B*) task outside the a-priori known world, (*C*) additional samples connecting the task to the search graph, (*D*) extended spatial index structure (here: vertex grid)

With phase 3 this initial path is smoothed using a canny pruning technique and spline segments. The next section describes this optimization process.

4.3.3 Path smoothing

Path smoothing is necessary, since the road map consists of straight lines that can render a path unachievable due to instant changes of the vehicle's heading. First, the path output from the path planner is likely to be simplified using the visibility domain. A given node Q is removed, if its predecessor Q_p and successor Q_s may be connected by a collision-free path. That way, shorter and smoother linear paths are generated, with a minimized number of collision checks.

The second step is of more complex nature. The contributed work in Praxedes (2007) addresses path optimization of non-linear path geometries. The proposed approach determines which nodes can be removed from the effective path ("shortcuts") and where path segments need additional nodes ("smoothing"). The implementation uses cubic

splines. Since the road map only guarantees collision-free linear paths between its nodes, the splines need further collision checks. Hence, a spline is discretized using equidistant samples along the spline path. Each sample is connected using a linear segment. These linear segments are then tested against collisions.

However, for collision-detecting purposes this may yield more points than necessary. Thus, a subset of points is used during an initial, coarse collision check, and the test space is refined as necessary. This is implemented as a recursive function that receives an initial and a final position. This function determines the maximum deviance between the spline and the straight line that connects the two given points. If this maximum deviance is greater than a safety distance, the function calculates a spline point between the two initial positions and makes a recursive call. Effectively, two new sub-intervals have to be analyzed. One between the start position and the middle spline point, and another one between the middle spline point and the final position. This algorithm works well since the implemented collision detection algorithm (Algorithm 3) also computes the distance to the closest obstacle if there is no collision. Thus, if the distance from a line to the closest obstacle is smaller than the maximum deviance of the spline, a collision is also detected. This adaptive method returns a final result within few iterations. The algorithm scales with the safety distance. It can be visualized as the radius of a cylinder whose axis is the straight line between two spline points. If the safety distance is increased, the algorithm speeds up since the number of points needed to represent the spline is decreased.

Once a collision has been found, the spline is modified within a limited number of iterations by "pushing" it towards the straight line path that is known to be collision-free (Fig. 12a). In case the number of iterations is exceeded, the affected spline segment between two graph nodes is substituted with the original straight line from the search graph (Fig. 12b). After that, two splines are connected to this straight line segment. The boundary condition for the new splines is the direction of this new straight line. This approach is feasible in cases where narrow passages require an extensive number of iterations to fit a spline onto a straight line (Praxedes 2007).

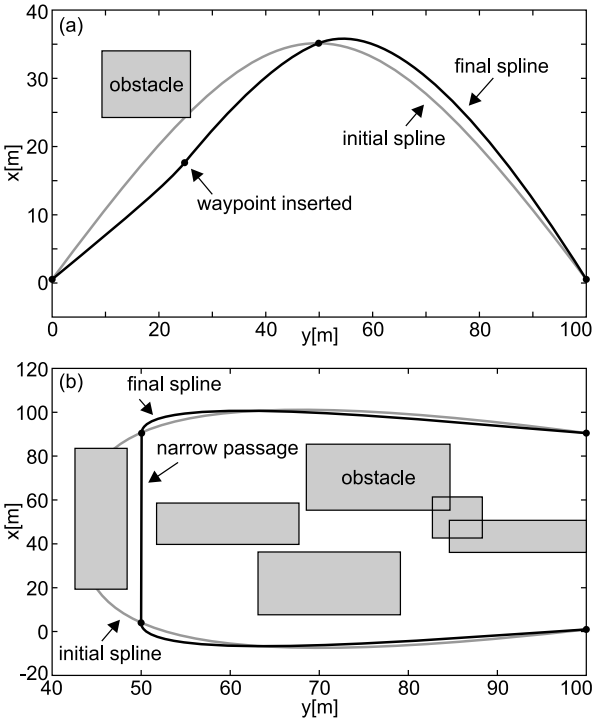

Fig. 12 Repairing spline trajectories by (**a**) inserting an additional spline support point in the middle of the linear connection of the spline segment's start and end point, (**b**) substituting a spline segment with a straight line

4.3.4 Road map updates

The road map covers the a-priori known free space. The current world model comprises no-fly zones and static obstacles for which it is assumed that a "disappearing" is not allowed. Thus, the focus is on detected obstacles, added to the world model online. The vertex grid-based spatial index structure generated during the offline learning phase is the basis for graph updates. Once a new set of polygons is added to the world model, the space partitioning allows a fast search of affected graph nodes. First, the bounding intervals on the x, y, z-axes are determined for a given polygon. In case it intersects a cell in the vertex grid, it is added to a list of polygons that affect this cell. After this step, a graph update comprises a search through all graph edges that are affected by the respective cell change. It is likely that one or more graph edges have to be marked as non traversable. This is done by assigning an infinite edge cost, a special threshold that is known throughout the planning system. If any edge exceeds that value it is ignored during graph search. Due to the computational efforts of connecting nodes to the graph, no graph edges are removed from the graph. Moreover it is possible that the added obstacle will be removed after some time. Thus, the edges can be assigned the original costs again.

Furthermore, the vehicle's current position is added as a node to the search graph (*sampling agent*). This node is a special case in two ways. First, it remains an instantaneous, temporary node which is removed from the graph after the path search. This way the vehicle position does not flood the search graph, but more importantly, it does not slow down the graph search due to a set of nodes with too short edge lengths. Second, for this node the current heading is used with a local planner (Hrabar 2006) that accounts for heading changes in cost such that sharp turns are only preferred if necessary. The path query favors the first successive node in the road map which is within a heading delta $\leq 90°$. Note

that neighbor search is performed within the radius of the maximum edge length defined by the sampling dispersion. Using the vertex grid, the affected graph edges are found quickly.

4.3.5 *Online path planner*

A single update to the road map triggers a single iteration of the online path planner. Three consecutive phases are executed:

1. a reconnect of the vehicle's state $[x, y, z, \psi]$ to the search graph,
2. a query step, which consists of a search within the updated road map, and
3. a post-processing for "smoother" paths is performed.

After the free space representation has been updated in the road map, the current position of the vehicle needs to be connected to the graph. Since the vehicle is expected to change the position between consecutive replanning steps, this implementation chooses the nearest, collision-free neighbor node of the previous path plan. This step avoids costly computations of the corresponding path segment and the exact position on this segment. Especially with spline segments, this step can become computationally costly.

As described in the beginning of this section, the AD* algorithm is applied for the path search and plan repair problem. As long as AD* yields suboptimal results, successive iterations of this online planning process are executed. AD* has a potential to waste computational time on plan repairs although a complete replan would be faster. Hence, it is important to choose a proper method to decide when replanning from scratch is favorable over plan repairing. In this implementation, two factors influence the decision. First, the time that has passed by since the last search tree update (e.g. $\leq$ 10 seconds). Second, it is taken into account how much of the search tree has become inconsistent (e.g. $\geq$ 30 %). Furthermore, AD* is optimized according to the recommendations in David Ferguson and Stentz (2005) that limits the expense of reordering the list of unexplored states (*OPEN* list) using a priority queue containing states with low key values and one unordered list containing the states with very large key values. The states from the unordered lists need to be considered only if the element at the top of the priority queue has a larger key value than the state with minimum key value in these lists.

The online path planning step uses the same set of path smoothing techniques as described in Sect. 4.3.3. Note that in the context of this work, the assumption from Hrabar (2006) is made as well: The helicopter is considered to be holonomic in the commandable degrees of freedom (x, y, z, ψ) at low velocities (e.g. $\leq$ 4 m/s).

As a result, an example of the effective path planning is illustrated in Fig. 13. The first snapshot (a) shows the initial mission planned through the set of waypoints from A to B. In the second scene (b) a newly detected obstacle set is added to the world model which triggers a replan of the path segment between the start position and A. Note that the online path planner does not consider subsequent mission tasks like the path from A to B. In (c) the spline path was modified due to new boundary conditions: The previous replan from start to A yields a different path tangent approaching A resulting in different boundary conditions for the spline between A and B. Snapshot (d) shows the effective path plan after the vehicle adapted the paths when it detected previously unknown obstacles.

5 Experimental results

The presented world modeling and path planning approach is developed within the ARTIS (Autonomous Rotorcraft Test bed for Intelligent Systems) research project that deals with unmanned helicopters. Flying test platforms were introduced by Dittrich et al. (2003) and follow a modular avionics and simulation concept inspired by the Georgia Tech GTMax UAV (Johnson and Schrage 2003).

The vehicle used in this work is a model helicopter (Fig. 14) with a main rotor diameter of 3 meters and a maximal take-off weight of 25 kg. It enables the evaluation of a variety of approaches to adaptive flight control, vision-based navigation, environment perception and onboard decision making. Sensor equipment comprises GPS, IMU and magnetometer for navigation, sonar altimeter for landing, wireless data links, manual R/C and video data links, a telemetry module and a flight control computer. The image acquisition and processing equipment are a 30 cm baseline stereo camera (Videre Design STOC) and a separate vision computer (Intel Pentium 4 Mobile). Camera field of view is approximately 50° × 40° and range measurements between 6.50 m and 40 m are evaluated. The camera system provides a 30 Hz VGA depth image calculation onboard a FPGA processor including camera calibration and depth confidence checks. An additional speck removal filter is implemented to improve the depth image quality.

5.1 Mapping in flight test

The mapping approach is tested in outdoor flights under real operational conditions. In the presented example, the 3D grid resolution is set to 0.5 m and zones of each 128 × 128 × 32 cells are built. Height histogram resolution for ground plane detection is set to 0.1 m. Figure 15 shows an example how image data is converted to an occupancy grid and eventually to a polygonal map according to the method described

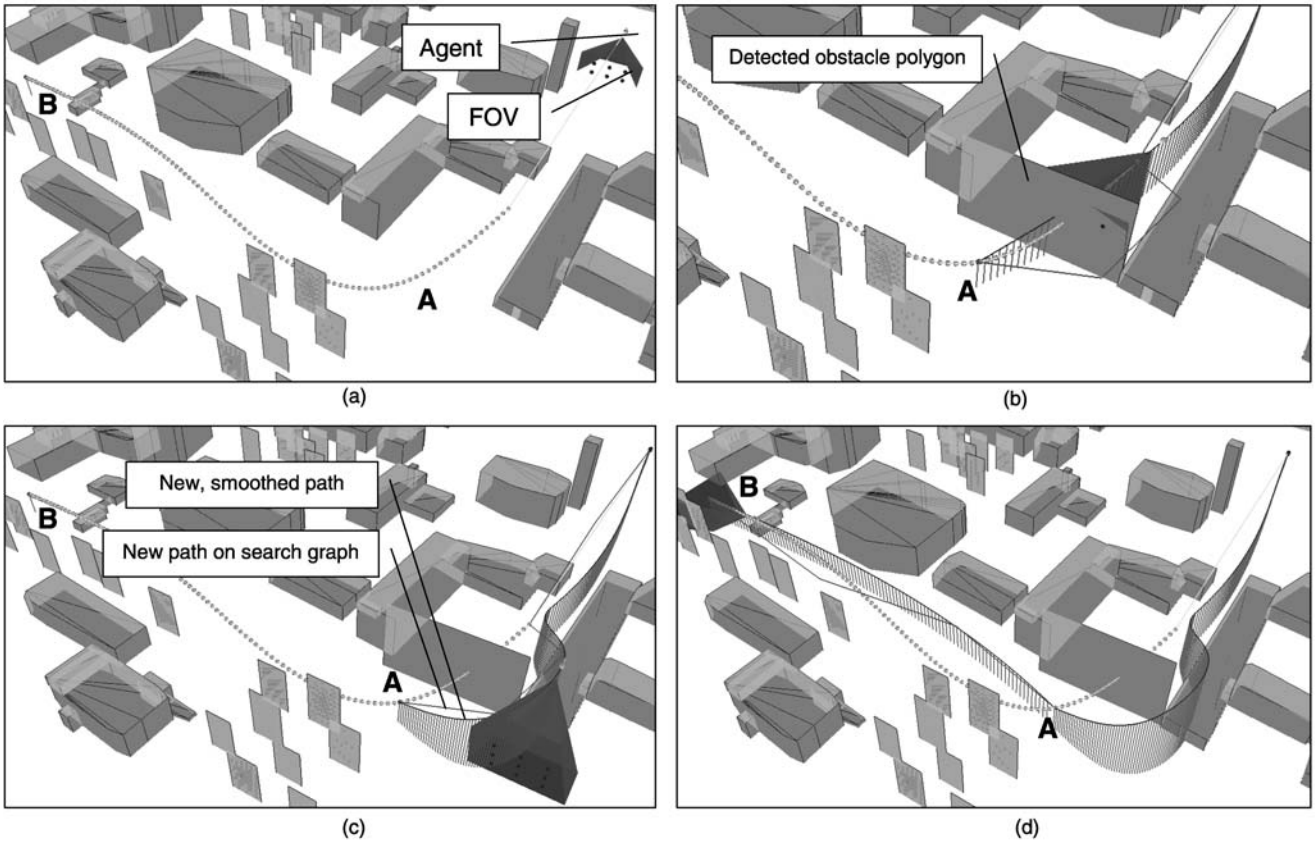

Fig. 13 Online path planning simulation test scenario: Obstacles are detected within field of view (FOV). Initial path through waypoints A–B (**a**), detected new obstacle obstructing the initial path (**b**), adapted spline segment through towards A (**c**), and effective path performed through A and B (**d**)

Fig. 14 The flight test carrier *ARTIS*

in Sect. 3. Figure 16 illustrates the incremental creation of the world model with prism shapes and ground planes. Subfigure (a) shows the helicopter in an urban terrain in the beginning of the flight. Mapping starts with onboard image acquisition as seen in (b) where confident obstacles and ground planes of the adjacent zones are already detected. While the helicopter is directed to fly a path along the houses with a speed of up to 4.5 m/s (c, d), additional objects and ground planes of other zones are added to the map until the helicopter reaches its ending position (e). By comparing the resulting map with ground truth data derived from aerial images, it is shown in (f) that visible walls, trees, ground and the gap between the buildings are identified and represented in the world model. Calculation speed in the given resolution is between 15 and 20 Hz on a 3 GHz computer, which is suitable for real-time use. The tests also show that the quality of vision and inertial navigation data on a vibrating vehicle is good enough to perform robotic mapping approaches on a helicopter system.

5.2 Path planning performance

The overall planning runtime in the test scenario was assessed using the environment setup described for Fig. 10 in Sect. 4.3.2, the test mission is shown in Fig. 13. The scenario includes possible world model changes inside the sensor's field of view. Obstacles are set into the world model if they are 40 meters or closer to the vehicle. The vehicle velocity is limited to 4 m/s. 20 test variations were performed, and in each of these tests, a subset of two waypoints was randomly chosen out of a set of predefined, reachable waypoints. The path planner always found a valid path.

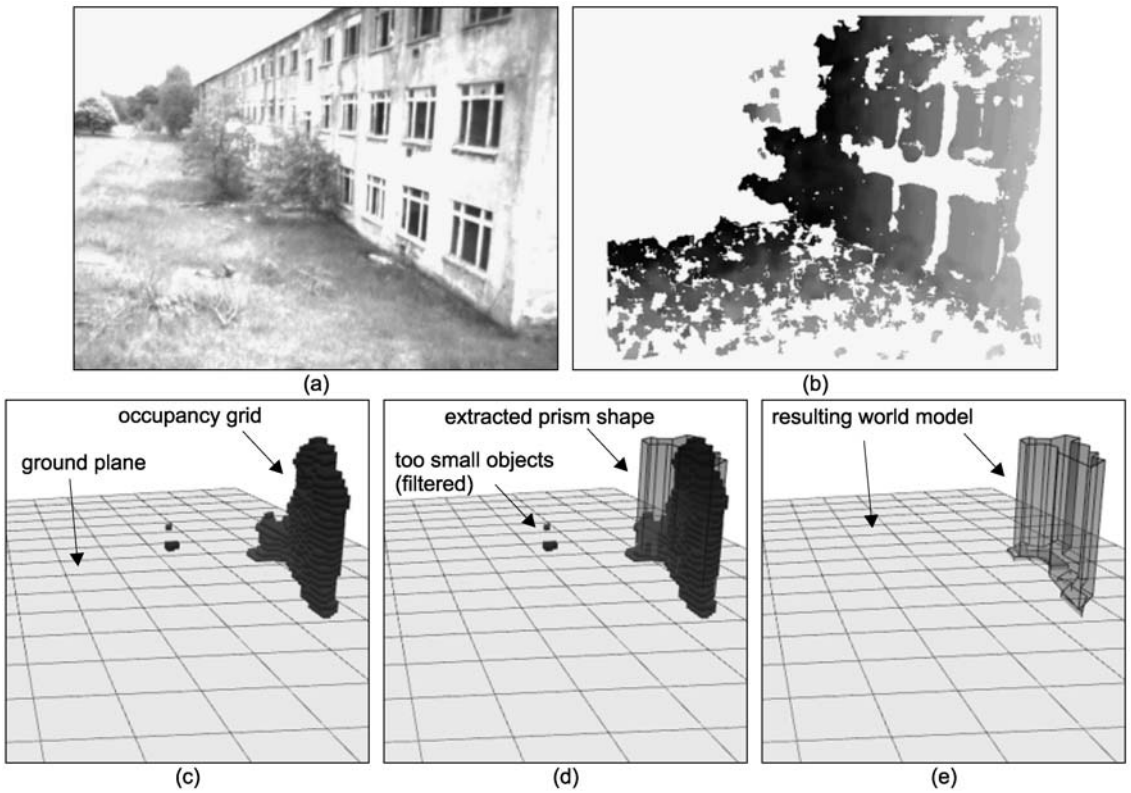

Fig. 15 Example of an intensity (**a**) and depth image (**b**) taken within a helicopter flight. In the depth image, darker areas refer to far distances, missing or filtered depth values are white. Occupancy grid mapping and ground plane estimation results (**c**), overlain prism shapes (**d**) and final map output without grid (**e**). Very small grid cell clusters are filtered during the shape extraction process in order to remove noise

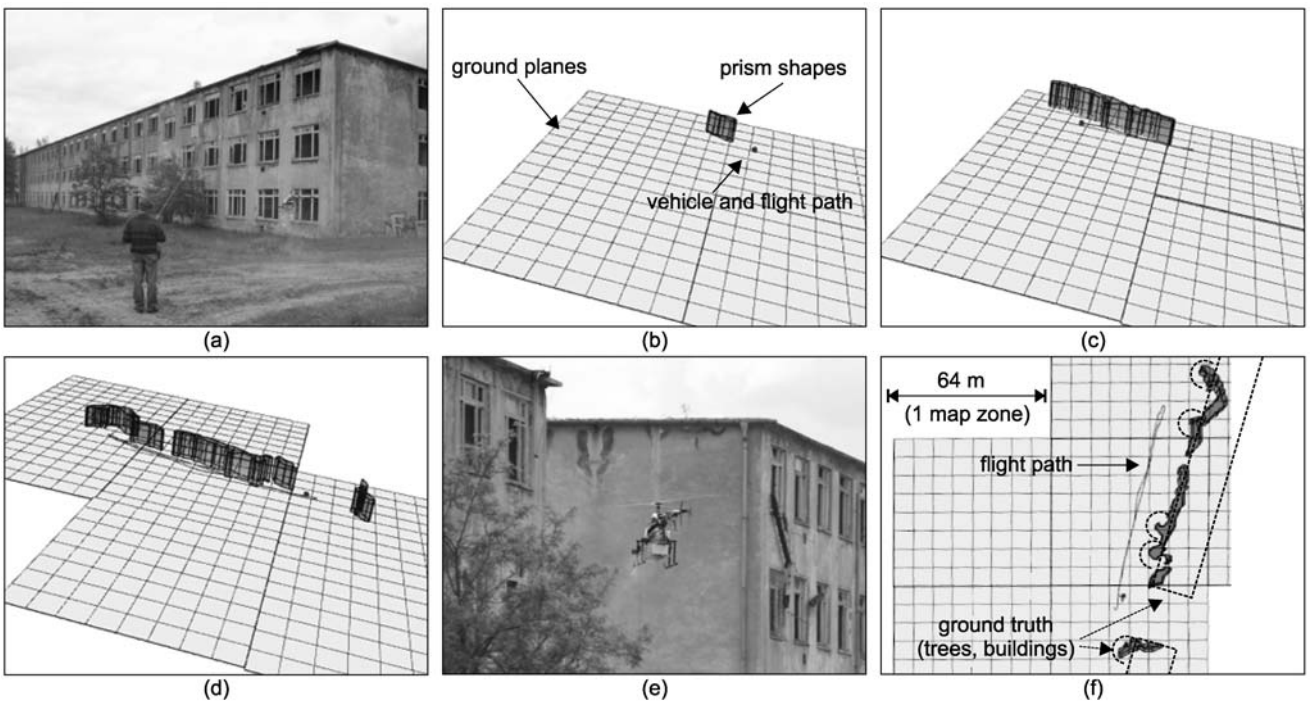

Fig. 16 Incremental map building during a helicopter flight in an urban area. Helicopter and the safety pilot for supervision in the beginning of the mapping flight (**a**), snapshots of the mapping process (**b**–**d**), photo at of the helicopter at the ending position (**e**), top view in comparison with ground truth (**f**). Buildings are shown with original size, trees are marked as *circles* with an increased radius to enhance visibility

Tests were performed in a simulation environment on 3.0 GHz Pentium 4 PC yielded an average overall plan update time of 60 milliseconds. The runtime share for the graph update was 35%, for the incremental graph search 10% and for path smoothing 60%. The remaining time was used for collision checks and vertex grid updates. During the initial road map construction phase the world sampling results of all three test cases are shown in Fig. 10, where 278 samples were distributed into the a-priori world of 780 m × 400 m area with a minimum height above each ground polygon of 5 m and 25 m maximum height of the Cartesian reference frame. The maximum height of 25 m above ground yields to sparse distribution of connected samples between the building streets. However, with a LRM or a QRM sampling the number of samples was sufficient enough to ensure at least one collision-free path per corridor. Moreover, during replanning a spline trajectory could be found every time such that no replacement with less feasible linear path segments would have been necessary.

The world sampling quality applied to the test environment yielded very few samples (278) compared to other application domains for roadmap-based path searches (e.g. robot manipulation with thousands of samples within the dexterous workspace). Hence, online sampling, edge updates and graph search take comparably short runtime. To fit the cubic spline between waypoints, splines are approximated by linear path segments during every collision check iteration. This yields a comparably high density of path segments and a higher number of iterations. A speed improvement could be achieved by reducing the number of iterations necessary to determine a collision-free spline.

The path planner finds collision-free paths between a start and an end position. However, the closer a new obstacle obstructs a path to a goal position, the more impact it has on the final configuration of the vehicle on that goal position. This is especially the case for spline-based trajectories (e.g. Fig. 13c), where the successive spline segment from A to B is not considered during path smoothing as long as A has not been reached. It is beyond the scope of this work to address this scheduling problem since more complex consideration must be made, e.g. to change the overall order in which user defined waypoints have to be passed depending on path updates.

6 Summary

This work presents an integrated architecture to navigate a flying vehicle through partially or totally unknown 3D environments. It consists of a world modeling step to gather environmental information and a global path planning step, based on the continuously updated world model. All algorithms are optimized to run on standard computer hardware onboard a small aerial vehicle and provide key enabling techniques for its autonomous navigation in complex environments.

World modeling refers to robotic mapping procedures. Sensor data fusion is performed with an occupancy grid map, and this map is the basis for polygonal shape extraction to generate the proper world model. A shape defined through one or multiple right prisms is calculated for each cluster of occupied grid cells. The shapes are calculated with each new sensor data in real-time. Ground detection is provided separately. As an example, the mapping algorithm is tested in flight by measuring obstacles with a stereo camera from an unmanned rotorcraft system and confirm the resulting compact map representations of obstacles.

The presented global path planner is suited to find paths at any time when unforeseen obstacle updates yield world model changes. The planner uses a deterministic sampling source and is extended by a vertex grid to constrain the search for affected graph edges to a minimal volume. This way, world model updates affect the global roadmap to a locally bounded minimum. An optimized AD* derivate is used for incremental path replanning. Furthermore, a spline-based path smoothing method compensates shortcomings of sampling based path planning. The path planner implementation runs on a consumer class computing hardware. Research is underway, to explore runtime behavior on larger terrain data of about five times the area, and double the height of the tested environments.

The current state of the system leaves room for optimizations and relaxing constraints. To extend the flight performance, research is underway to relax the holonomic path planning constraint. Moreover, an online task scheduling is desirable to avoid situations where the overall mission cost (e.g. total duration time) is affected due to path replanning. For example, if the initial plan was to fly from task A to C over B, it is possible that due to new obstacles it could be more suitable to fly to task C first and then to task B.

References

Adolf, F., Langer, A., de Melo Pontes e Silva, L., & Thielecke, F. (2007). Probabilistic roadmaps and ant colony optimization for UAV mission planning. In *6th IFAC symposium on intelligent autonomous vehicles*.

Andert, F., & Goormann, L. (2008). A fast and small 3-d obstacle model for autonomous applications. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 2883–2889).

Branicky, M. S., LaValle, S. M., Olson, K., & Yang, L. (2001). Quasi-randomized path planning. In *IEEE international conference on robotics and automation* (pp. 1481–1487).

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

Bruce, J., & Veloso, M. (2007). Real-time randomized motion planning for multiple domains. In *RoboCup 2006: Robot Soccer World Cup X* (pp. 532–539). Berlin: Springer.

David Ferguson, M. L., & Stentz, A. T. (2005). A guide to heuristic-based path planning. In *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*.

Dittrich, J. S., Bernatz, A., & Thielecke, F. (2003). Intelligent systems research using a small autonomous rotorcraft testbed. In *2nd AIAA unmanned unlimited conference, workshop and exhibit*.

Eberly, D. (1999). *Distance between two line segments in 3d*. Tech. rep., Geometric Tools, LLC, http://www.geometrictools.com/.

Ersson, T., & Hu, X. (2001). Path planning and navigation of mobile robots in unknown environments. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 858–864).

Garratt, M. A., & Chahl, J. S. (2008). Vision-based terrain following for an unmanned rotorcraft. *Journal of Field Robotics*, *25*(4–5), 284–301.

Green, W. E., Oh, P. Y., & Barrows, G. (2004). Flying insect inspired vision for autonomous aerial robot maneuvers in near-earth environments. In *IEEE international conference on robotics and automation* (pp. 2347–2352).

Griffiths, S., Saunders, J., & Curtis, A. (2006). Obstacle and terrain avoidance for miniature aerial vehicles. *Robotics and Automation Magazine*, *13*(3), 34–43.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetic*, *4*(2), 100–107.

Hrabar, S. E. (2006). *Vision-based three-dimensional navigation for an autonomous helicopter*. Ph.D. thesis, University of Southern California, Los Angeles, CA, USA, adviser-Gaurav Sukhatme.

Hrabar, S. E. (2008). 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 807–814).

Hsu, D., Latombe, J. C., & Kurniawati, H. (2006). On the probabilistic foundations of probabilistic roadmap planning. *International Journal on Robotics Research*, *25*(7), 627–643.

Iocchi, L., Konolige, K., & Bajracharya, M. (2000). Visually realistic mapping of a planar environment with stereo. In *Seventh international symposium on experimental robotics*.

Johnson, E. N., & Schrage, D. P. (2003). The Georgia tech unmanned aerial research vehicle: Gtmax. In *AIAA guidance, navigation and control conference*.

Konolige, K. (1997). Improved occupancy grids for map building. *Autonomous Robots*, *4*, 351–367.

Langer, A. (2006). *Three-dimensional path planning for an unmanned rotorcraft using probabilistic roadmaps*. Instituto Tecnológico de Aeronáutica, São José dos Campos.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge: Cambridge University Press. Also available at http://planning.cs.uiuc.edu/.

Likhachev, M., Gordon, G., & Thrun, S. (2003). Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in neural information processing systems conference*. Cambridge: MIT Press.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2005). Anytime dynamic a*: An anytime, replanning algorithm. In *AAAI international conference on automated planning and scheduling*.

Morales, Y., & Tsubouchi, T. (2007). Gps moving performance on open sky and forested paths. In *IEEE/RSJ international conference on intelligent robots and systems* (pp. 3180–3185).

Moravec, H. P., & Elfes, A. (1985). High resolution maps from wide angle sonar. In *IEEE international conference on robotics and automation* (Vol. 2, pp. 116–121).

Okada, K., Kagami, S., Inaba, M., & Inoue, H. (2001). Plane segment finder: Algorithm, implementation and applications. In *IEEE international conference on robotics and automation* (pp. 2120–2125).

Pettersson, P. O., & Doherty, P. (2006). Probabilistic roadmap based path planning for an autonomous unmanned helicopter. *Journal of Intelligent and Fuzzy Systems*, *17*(4), 395–405.

Plaku, E., Bekris, K. E., Chen, B. Y., Ladd, A. M., & Kavraki, L. E. (2005). Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, *21*, 597–608.

Prassler, E., Scholz, J., & Elfes, A. (2000). Tracking multiple moving objects for real-time robot navigation. *Autonomous Robots*, *8*, 105–116.

Praxedes, L. G. (2007). *Advanced three-dimensional route planning under complex constraints*. Instituto Tecnológico de Aeronáutica, São José dos Campos.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, *1*(3), 244–256.

Ren, M., Yang, J., & Sun, H. (2002). Tracing boundary contours in a binary image. *Image and Vision Computing*, *20*, 125–131.

Ruffier, F., & Franceschini, N. (2005). Optic flow regulation: the key to aircraft automatic guidance. *Robotics and Autonomous Systems*, *50*, 177–194.

Saunders, J. B., Call, B., Curtis, A., Beard, R. W., & McLain, T. W. (2005). Static and dynamic obstacle avoidance in miniature air vehicles. In *AIAA infotech aerospace conference*.

Scherer, S., Singh, S., Chamberlain, L., & Saripalli, S. (2007). Flying fast and low among obstacles. In *IEEE international conference on robotics and automation* (pp. 2023–2029).

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

Thrun, S. (2002). Robotic mapping: A survey. In G. Lakemeyer & B. Nebel (Eds.), *Exploring artificial intelligence in the new millennium*. San Mateo: Morgan Kaufmann.

Vestka, L. E. K. P., Claude Latombe, J., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*, 566–580.

Watanabe, Y., Calisey, A. J., & Johnson, E. N. (2007). Vision-based obstacle avoidance for uavs. In *AIAA guidance, navigation, and control conference*.

Wu, A. D., Johnson, E. N., & Proctor, A. A. (2005). Vision-aided inertial navigation for flight control. In *AIAA guidance, navigation and control conference*.

Zufferey, J. C., & Floreano, D. (2005). Toward 30-gram autonomous indoor aircraft: vision-based obstacle avoidance and altitude control. In *IEEE international conference on robotics and automation* (pp. 2594–2599).

Franz Andert received the Diploma degree in computer science in 2006 from the Humboldt University, Berlin, Germany. He is currently a doctoral canditate working at the German Aerospace Center (DLR) in Braunschweig, Germany. His research interests include autonomous vehicles, computer vision, world modeling and obstacle avoidance.

Florian Adolf undertakes research in the design and implementation of Decision Making Systems. Since 2005 he works in as a postgraduate fellow for the Institute of Flight Systems at the German Aerospace Center (DLR). Before joining DLR, he studied computer vision applications for behavior-based obstacle avoidance in the Robocup Middle-Size team AIS/BIT, a project on "autonomous mobile robotics in highly dynamic environments", which is also known as robot soccer. He graduated from the Department of Computer Science at Trier University of Applied Sciences in 2003. In early 2006 he received a Master of Science in Autonomous Systems from the Department of Computer Science at Bonn-Rhein-Sieg University of Applied Sciences.