UAV manuscript No.
(will be inserted by the editor)

Intuitive 3D Maps for MAV Terrain Exploration and Obstacle Avoidance

Stephan Weiss · Markus Achtelik · Laurent Kneip
Davide Scaramuzza and Roland Siegwart

Received: date / Accepted: date

Abstract Recent development showed that Micro Aerial Vehicles (MAVs) are nowadays capable of autonomously take off at one point and land at another using only one single camera as exteroceptive sensor. During the flight and landing phase the MAV and user have, however, little knowledge about the whole terrain and potential obstacles. In this paper we show a new solution for a real-time dense 3D terrain reconstruction. This can be used for efficient unmanned MAV terrain exploration and yields a solid base for standard autonomous obstacle avoidance algorithms and path planners. Our approach is based on a textured 3D mesh on sparse 3D point features of the scene. We use the same feature points to localize and control the vehicle in the 3D space as we do for building the 3D terrain reconstruction mesh. This enables us to reconstruct the terrain without significant additional cost and thus in real-time. Experiments show that the MAV is easily guided through an unknown, GPS denied environment. Obstacles are recognized in the iteratively built 3D terrain reconstruction and are thus well avoided.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 231855 (sFly). Stephan Weiss is currently PhD student at the ETH Zurich. Markus Achtelik and Laurent Kneip are currently PhD students at the ETH Zurich as well. Davide Scaramuzza is currently senior researcher and team leader at the ETH Zurich. Roland Siegwart is full professor at the ETH Zurich and head of the Autonomous Systems Lab.

S. Weiss · M. Achtelik · L. Kneip · D. Scaramuzza · R. Siegwart
ETH Autonomous Systems Laboratory, 8092, Zurich, Switzerland, www.asl.ethz.ch
E-mail: stephan.weiss@mavt.ethz.ch, markus.achtelik@mavt.ethz.ch, kneipl@ethz.ch
davide.scaramuzza@ieee.org, rsiegwart@ethz.ch

1 Introduction

In the past years, micro aerial vehicles (MAVs) strongly gained in autonomy. This was motivated through the very wide field of applications for these little platforms. Commonly associated keywords are: search and rescue, exploration, surveillance, agriculture and inspection. Research in the field of aerial vehicles evolved incredibly fast. Today MAVs are able to autonomously take off, land, hover and follow a path using one single camera as the only exteroceptive sensor [1].

Medium scale aerial vehicles usually have a lot of payload. They can carry highly accurate and sophisticated sensors such as long range laser scanners, high performance inertial sensors and a lot of computation power. In this paper, we focus on low weight MAVs (around 500g). This implies many restrictions on the sensor suite one can mount on the vehicle. Also, computation power is so critical that until today almost all solutions need a ground station for off-platform computing. These two main restrictions - sensors and computation power - will stand in the center of the future research of the community.

Our concern in this contribution lies in the exploitation of the on board sensors. Our aim is to use only one exteroceptive low cost and low weight sensor for all tasks: one omnidirectional camera. In our case, the camera is already used for the localization and stabilization of the vehicle. We use a visual SLAM framework [2] as a base for the visual navigation. Having the SLAM position estimate as controller input we are able to autonomously take off, land and follow paths using only one single camera [1].

Usually (visual) SLAM algorithms manage a map built out of point features. The point cloud of such a map is neither for a human user nor for standard path planning and obstacle avoidance algorithms of great use. It is thus highly desirable to have an intuitive 3D map instead of an illegible point cloud. Also, this map shall be available and updated while the SLAM is running (i.e. the MAV is flying) in real-time.

We use now the same camera we use for the visual SLAM framework to also reconstruct in real-time a textured 3D mesh of the environment for efficient exploration. In this textured mesh we can also recognize obstacles and avoid them using standard obstacle avoidance routines and path planners in 3D. Moreover, the texture on the mesh map may be used to facilitate interaction with human operators. Thanks to the texture the operator has a good understanding of the environment.

Note that for localization and stabilization of the vehicles often a GPS sensor is used for outdoor scenarios. Even though GPS sensors are lightweight and their data easily is processable, we do not rely on this sensor type. We aim at a system able to operate indoor and outdoor. Thus GPS is not an option. Also, GPS signals are highly distorted in urban environments and yield thus false measurements.

The paper is organized as follows. In section 2 we compare our work to the current state of the art. In section 3 we review shortly the framework we are using for the feature extraction and matching. Section 4 is dedicated to the mesh reconstruction and to the texturing of the mesh. Then, in section 5, we show the results of the framework we built for efficient unmanned terrain exploration. We also shed light to how to use the mesh for (autonomous) obstacle avoidance. Finally we conclude the paper in section 6.

2 Related Work

Map generation in 3D is a well known problem and solutions vary over a large field. The reason of this huge variation in representing and calculating a map lies in the various different needs of every application. To our knowledge the real-time dense 3D map reconstruction presented here on low weight MAVs for unknown mid scale GPS denied environments is the first of its kind.

Already in a very early work A. Elfes described the use of occupancy grids for mobile robots [3]. The author characterized sonar sensors to build an occupancy grid using a ground robot. The idea of occupancy grids is until today wide spread. Recent results are published by Zask [4]. The authors use a 3D occupancy grid to accurately model the environment including obstacles. The approach uses SIFT features and ISO surfaces to obtain an outlier free and smooth surface. The authors claim that the approach runs in real time and is applicable to any other features. However, they do not discuss the influence of outliers. Moreover the grid has to be manually initialized in order to fit the scene which shall be mapped during runtime. A general issue with occupancy grids is the memory cost. The higher the resolution the more storage is required. This becomes critical in outdoor scenarios and in 3D maps. To avoid this problem, our approach uses only the point features generated by the visual SLAM algorithm. This automatically ensures detailed structure on texture rich surfaces (i.e. where the environment changes) and assumes larger planes on texture-less areas. Thus, the reconstruction naturally adapts the degree of details to its environment.

A dense mapping approach is to use stereo vision. Nowadays hardware implementations are capable of building a disparity map in real time. Chen and Xu
[5] used stereo vision to build a 3D map in real time on an autonomous land vehicle. They use again a 3D occupancy grid and GPS information to build a globally consistent map. Early research of Lacroix [6] presents a method to build fine resolution digital terrain maps on the basis of a set of low altitude aerial stereo vision images. They match interest points between successive images to map a large outdoor scenario in fine resolution. The map generation is done offline. Dense stereo vision has also been used for pure obstacle detection in [7]. The authors did not imply the 3D mapping. A general issue with stereo vision is that the stereo effect vanishes the shorter the base line gets with respect to the scene depth. Thus we rely on a monocular solution in which the appropriate base line is provided by the keyframe based SLAM framework.

Laser scanners are often used to acquire a very reliable measurement of the environment to match afterwards the corresponding texture from images to the laser scan point cloud. Biber [8] demonstrated this technique for a 3DTV application. Also Triebel et al. showed in [9] how to solve the problem of underpassings and vertical structures in large outdoor environments using multi level surface maps and a SICK laser scanner on a ground robot. Laser scanners are, however, not an option for MAVs considering the weight and power budget. The same drawbacks also limit developments using range image cameras [10]. The authors show detailed 3D maps which make these sensors a valid option on high payload vehicles. In our case weight and power consumption discard this branch of solutions.

The computer vision community proved the accuracy of 3D reconstruction by only using one camera. They apply Structure from Motion (SfM) to merge different camera views to one 3D scenario. However, computational cost is the limiting factor. Paalanen et al. [11] select two image frames out of a monocular SLAM sequence to build a dense stereo map. First tests showed encouraging results for this kind of dense 3D reconstruction. The algorithm needs, however, improvement in its accuracy and is not yet running in real time (but close to it). To our knowledge, SfM needs still a lot of research to use it in real time for MAV navigation and mapping. Simplified approaches make use of a known model of the environment. Kemp uses in [12] lines in the

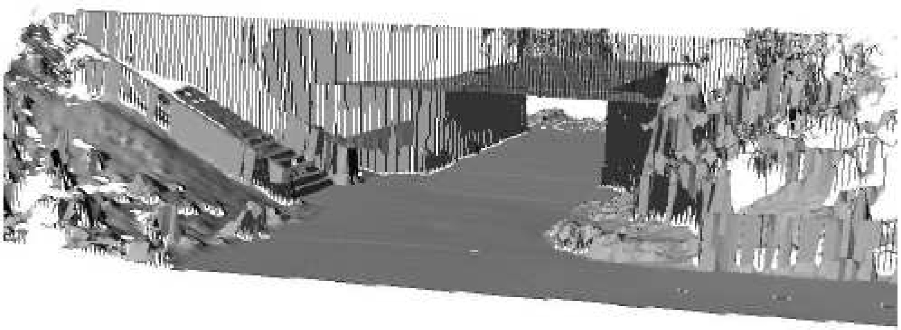


Fig. 1 Multi Level Surface Map: This technique handles well large data sets and vertical structures. Here an underpassing is correctly modeled whereas a simple elevation map would have closed the pass-through. (Courtesy of Triebel et al. [9])

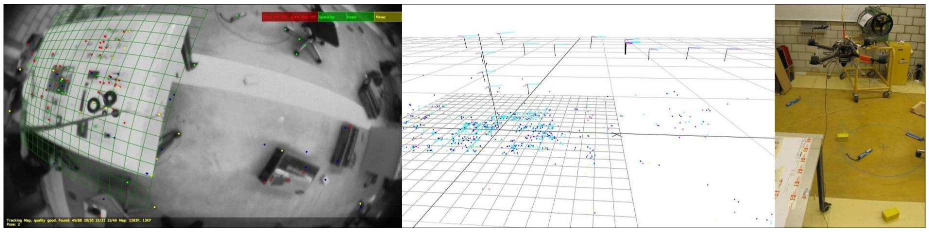


Fig. 2 Screenshot of Klein's visual SLAM algorithm. On the left, the tracking of the FAST corners can be observed. This is used for the localization of the camera as well as for the reconstruction of the 3D mesh map. In the middle, the 3D point map that was built by the mapping thread is shown. The 3-axis coordinate frames represent the location where new keyframes where added. The right picture shows our airborne vehicle (a *Hummingbird* from Ascending Technologies) in hovering state. The down-looking camera is mounted on the vehicle's bottom. All three images were taken at same time.

environment to control an MAV. For unknown terrain exploration the knowledge of a model is, however, rarely given.

Our here presented approach is new in several aspects. We are able to reconstruct a dense map in real-time purely based on a visual input from one single camera. With a weight of only 12g the camera is probably the lightest map building device available. We ensure low computational cost by using the same features for building the dense 3D map as for the visual navigation and control. The computational expensive features are only computed once. Our mesh based approach can handle very sparse features for a dense textured 3D mesh map.

3 Framework

3.1 Description of the Visual SLAM Algorithm

The approach presented here uses the visual SLAM algorithm of Klein and Murray [2] in order to localize the MAV and build a dense 3D map with a single camera (see Fig. 2). Note that the authors of [2] state that the framework is only suitable for small-scale environments. Our approach relies thus only on the local environment rather than on the consistency of the global map. For visual based navigation, exploration and obstacle avoidance it is indeed sufficient to have only locally a consistent map. We will discuss this fact later in section 5.

In summary, Klein and Murray split the simultaneous localization and mapping task into two separately scheduled threads: the tracking thread and the mapping thread. The tracking thread is first of all responsible for the tracking of salient features in the camera image, i.e., it compares the extracted point features with the stored map and thereby attempts to determine the position of the camera. This is done with the following steps: first, a simple motion model is applied to predict the new pose of the camera. Then the stored map points are projected into the camera frame and corresponding features (FAST corners in this case) are searched. This is also referred to as the data association procedure. When this is done, the algorithm refines the orientation and position of the camera such that the total error between the observed point features and the projection of the map points into the actual frame is minimized. The Mapping thread uses a subset of all camera images - also called keyframes - to build a 3D point map of the surroundings. The keyframes are selected using some heuristic criteria. After adding a new keyframe, a batch optimization is applied to the joint state of map points and keyframe poses. This attempts to minimize the total error between projected map points and the corresponding observations in the keyframes. In the computer vision community, this procedure is also referred to as bundle adjustment. It is alternately applied to the global or to a local set of map points and keyframes. There are several important differences between the SLAM algorithm considered here and the standard approach (for example by Davison et al. [13] . First of all Kleins algorithm does not use any Extended Kalman Filer based state estimation and does not consider any uncertainties, be it for the pose of the camera or for the location of the features. This saves a lot of computational effort that would occur with the processing of the corresponding data. Considering the uncertainty of the state could ease the data association process and enable larger loop closing. The lack of modeling uncertainties, however, is compensated by using a vast amount of features and the local and global batch optimization. Therefore, despite using a fixed area for feature matches, the algorithm is still able to track efficiently the point features and to close smaller loops. This makes the algorithm fast and the map very accurate.

3.2 Analysis of the SLAM Algorithm

The main advantage of the thread splitting lies therein that both the mapping and the tracking thread can run at different frequencies. Thus, the mapping thread is able to apply a much more powerful and time-consuming algorithm to build its map. Simultaneously, the tracking thread can estimate the camera pose at a higher frequency. This does strongly improve the performance. Compared to frame-by-frame SLAM, the algorithm of Klein et al. saves a lot of computational effort in that it does not process every single image. Particularly when using a camera with a wide field of view, consecutive images often contain a lot of redundant information. In addition, for example, when the camera is moving very slowly or if it stays at the same position, the mapping thread does rarely evaluate the images and requires thus only very little power. This is the main reason why we chose this SLAM algorithm. When moving the helicopter through a region, our camera is facing downwards. This increases the overlapping image portion of neighboring keyframes, so that we can even further loosen the heuristics for adding keyframes to the map. In addition, once the MAV has explored a certain region, no more new keyframes will be added within the region boundaries. The computation time remains thus constant. On the other hand, when exploring new areas the global bundle adjustment can be very expensive, limiting the number of keyframes to a few hundred on our platform (around 50-100 m2, depending on the keyframe heuristics). Another strength of the SLAM algorithm is its robustness against partial camera occlusion. If a sufficient part (around 50%) of the point features can still be tracked the pose estimate is accurate enough to sustain stable MAV control. In addition, the algorithm will avoid adding any keyframes in such a situation so as not to corrupt the map. An intricate hurdle when using a monocular camera is the lack of any depth information. Because of that, the algorithm must initialize new points based on the observations from more than one keyframe. This could motivate the use of a stereo camera. However, for a stereo camera to bring any further advantage, the observed scene must be within some range of the stereo camera, otherwise the setup will degenerate to the same performance as a single camera. Closely linked to this problem is the unobservability of the map scale, to tackle this we are forced to estimate the map scale by hand and pass it manually to the controller. On-board real-time scale estimation is subject to our current research.

3.3 Adaptations to the SLAM Algorithm

We adapt some parameters of Klein's visual SLAM algorithm to increase its performance within our framework. First, we use a more conservative keyframe selecting heuristic in order to decrease the number of keyframes added during map expansion. Thereby, we are able to map much larger areas before reaching the computational limit. Additionally, we reduce the number of points being tracked by the tracking thread from 1000 to 200. This again increases the maximal map size and the frame rate, while keeping the accurate tracking quality and the robustness against occlusion. This leads to a very sparse information for the 3D mesh map, however, our tests show still very satisfying results underlining the strength of our approach for dense textured 3D mesh maps. To increase the autonomy of the system we write a routine that can store and load maps. With this, we are able to overcome the initialization of the map during which no position estimate is available. It provides the possibility to start the helicopter from a small known patch and to skip the initialization process. Later the loaded map of the patch can be expanded by exploring the environment.

3.4 Implementation of a Visual SLAM Based Controller

In previous work [1] we implemented the above described SLAM framework in a LQG/LTR controller for autonomous visual based navigation. We used the pose estimation of the SLAM algorithm as input to the controller. This way, we are able to take off, land and hover. Moving the hovering set point step by step also allows set point following (i.e. trajectory following). Figure 3 shows a path flown by the MAV completely autonomously in an indoor environment. We will implement our real-time map generation in the same framework. Originally the framework only has the sparse 3D point features as a map. We will extent this in the present work to a textured 3D mesh. This step makes efficient terrain exploration possible - such as it is often desirable in disaster areas. By avoiding the MAV flying through the mesh we have as well an efficient possibility of autonomous obstacle avoidance.

4 3D Map Generation

In section 3 we described our framework from previous work which enables us to fly an MAV autonomously in unknown terrain. In this section we focus on the generation of the real-time 3D map. We will first describe the algorithms to build the mesh for the 3D map. Then we highlight

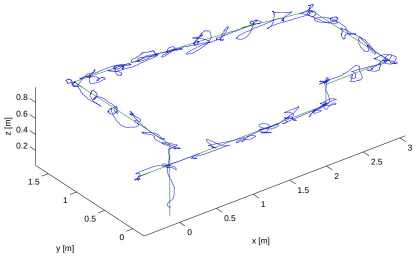


Fig. 3 Path that the helicopter has flown. This does not represent ground truth, it is the pose estimation of the SLAM algorithm. However the attitude of the helicopter can be observed while successfully flying a rectangular loop and landing on the ground. The RMS value of the position error is 9.95 cm in x, 7.48 cm in y and 4.23 cm in z. The path has a total length of a little bit more than 10 m in an area of 3.5x2x1m^3

the texturing procedure and our technique to always have the highest resolution texture available on any given mesh triangle.

4.1 3D Mesh Generation from a Point Cloud

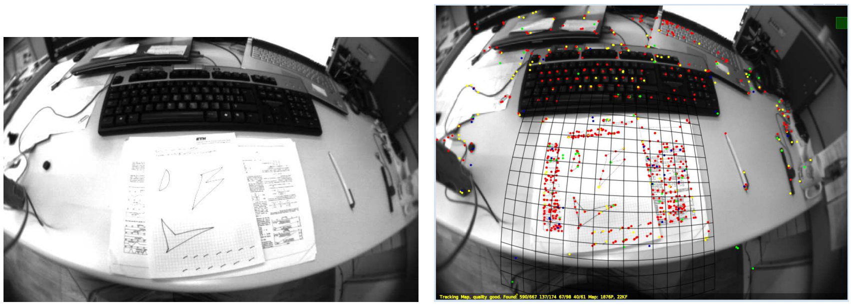
Feature extraction and matching have a very high computational cost. Hence visual SLAM algorithms typically use as few features as possible. On the contrary, this is unsuitable for 3D reconstruction. In our approach we show that one can use the same information for the SLAM task as for the 3D reconstruction of the map and thus one can save significant processing time. This way the features have only to be extracted and matched once for both tasks.

In Figure 4 a sample scene is depicted. For simpler understanding we will refer to this scene throughout this section. Note that it is a small scale scene, however, due to our monocular approach, all techniques and algorithms applied to this scene are perfectly scalable. That is, huge terrain captured from far away looks identical to a small terrain captured from very close - i.e. the images and thus the map are scale invariant. We will show our algorithms in section 5 in a mid scale indoor environment. Figure 4(b) shows the information available in a keyframe of the SLAM algorithm. The 3D point features can be projected to the 2D image plane in order to extract the desired texture.

The first step to generate the textured 3D map is to transform the point cloud into a 3D mesh. Assume the point cloud $\{\mathbf{p_i}\}$ with M 3D points $\mathbf{p_i}$ representing the initial map constructed by the visual SLAM algorithm in the start phase. Without any restrictions to the terrain to explore later on we assume the start area to be relatively flat and the aerial vehicle in hover mode. The main map plane H is found using a least square method on $\{\mathbf{p_i}\}$ or a RANSAC algorithm. In our case the latter one is used to be more robust against outliers. This is done in the given SLAM framework. All current and future map points are projected to this main plane to reduce the dimensionality:

$$\mathbf{r_i} = P * \mathbf{p_i} \tag{1}$$

where $\mathbf{p_i}$ is a three dimensional point of the current map and $\mathbf{r_i}$ is its two dimensional counterpart projected to the main map plane H using the 2x3 projection matrix P. Note that H usually corresponds to a physical plane in the scene (i.e. table or floor). Furthermore, as the camera is down looking on a helicopter this plane usually is only slightly inclinated to the xy-plane in the camera frame. Thus the two dimensional positions of the features $\mathbf{r_i}$ are accurate while the third


Tracking Map, quality good. Found: 590/667 137/174 67/98 40/61 Map: 1876P, 22KF

(a) (b)

Fig. 4 a) Sample image of the scene mapped for the following illustration of the algorithm in this section. The sheets in front of the keyboard are flat and represent the main plane H whereas the keyboard has a soft inclination in depth towards the upper part of the image. b) Scene with 3D point features. This represents the data available in a keyframe of the visual SLAM algorithm. Back projecting a 3D triangle of the meshed map allows getting the texture for the triangle in question. Note that this is the distorted image while for texturing the mesh we use the undistorted one.

(eliminated by the projection) is very noisy due to the depth triangulation of the visual SLAM algorithm.

After the projection a Delaunay Triangulation is run in 2D space to generate a 2D mesh. We use a Sweep algorithm for the triangulation to keep calculation power low. For the Sweep triangulation, calculation is in the order of $O(n \log n)$ compared to the standard algorithm with $O(n^2)$.

The 3D point cloud of the scene is depicted in Figure 5. One can note the difficulty even a trained eye has to interpret the scene. Standard path planning and obstacle avoidance algorithms cannot be used. In Figure 6 the generated mesh is shown. After the Delaunay Triangulation in 2D space we add again the third dimension. As equation (1) is not invertible (P is not a square matrix and we therefore have ambiguities in the back projection) we only use the edge information of the Delaunay Triangulation. That is if an edge in the 2D Delaunay Triangulation is defined by

$$d_{2d} = \overline{\mathbf{r_i r_j}} \tag{2}$$

we map it to an edge in 3D space according to

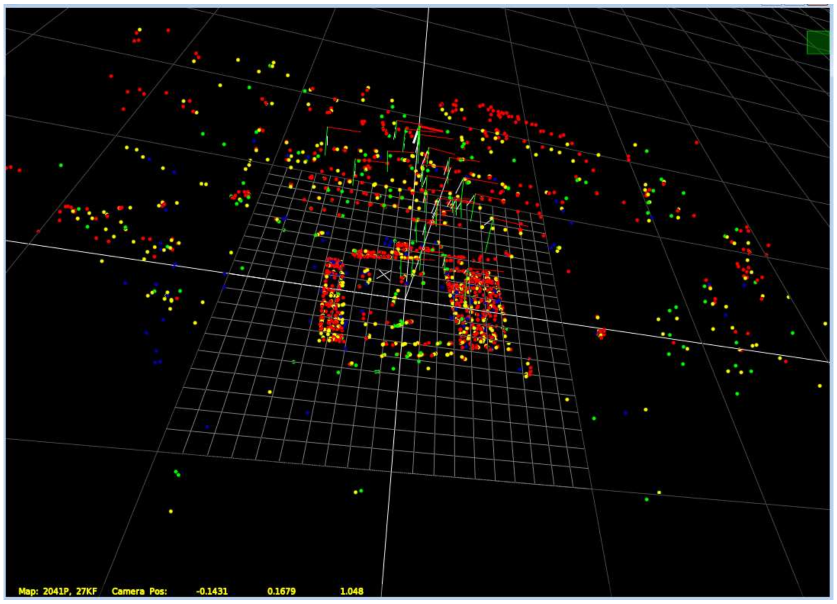
$$d_{3d} = \overline{\mathbf{p_i p_j}} \tag{3}$$

with $\mathbf{r_k} = P * \mathbf{p_k}$ and $k \in$ map. This initial 3D mesh is then median filtered in the third coordinate to remove outliers and noise. The median value is calculated using all adjacent vertices to the center vertex. That is

$$p_{zk} = \text{median}(p_{zi} \forall p_{zi} \in d_{3d} = \overline{\mathbf{p_k p_i}}) \tag{4}$$

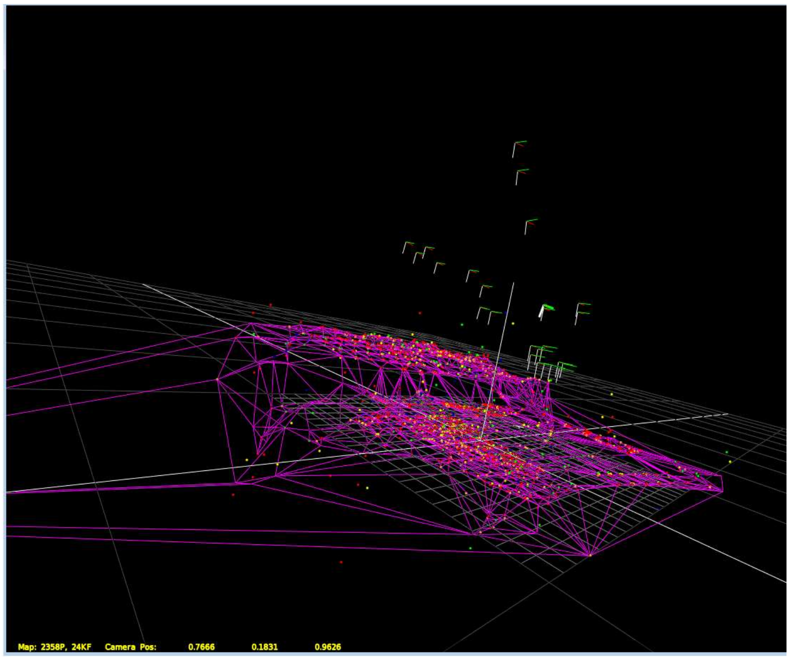
where p_{zi} denotes the third coordinate of the 3D point $\mathbf{p_i}$ previously eliminated for the Delaunay Triangulation.

At this point standard path planning and obstacle avoidance algorithms could be applied for enhanced autonomous navigation. The most simple rule for obstacle avoidance is to not traverse the mesh. That is, if the airborne vehicle always stays on the same side of the mesh it will not crash against an obstacle. Note that thanks to the sparseness of the point features this rule is highly robust, however, may be too restrictive in some particular cases. We will discuss this issue in section 5.



Map: 2041P, 27KF Camera Pos: -0.1431 0.1679 1.048

Fig. 5 The 3D point cloud of the sample scene. A trained eye can spot the papers and the keyboard. However, usually neither human users nor standard path planning and obstacle avoidance algorithms understand the point cloud



Map: 2358P, 24KF Camera Pos: 0.7666 0.1831 0.9626

Fig. 6 Applying Delaunay Triangulation to the point cloud reveals the real topology of the scene. The ’hill’ represents the keyboard in the sample scene. Note that we applied a median filter to the mesh vertices in order to eliminate outliers. Thus the 3D points may not always lie on the grid. This grid is already sufficient for path planning and obstacle avoidance.

4.2 Texturing the 3D Mesh Map

For the user it is important to further augment the map. Only with further map information (i.e. texture) he can spot areas of interest. This is of great use in disaster areas where the MAV autonomously explores the environment - using for example our approach in [1] . A user can then send the MAV to an area of interest to gather even more detailed information.

The projection of the texture to the mesh can be divided in to three steps. First we augment each keyframe data with the undistorted image. Second we take the three points of a 3D triangle of the mesh and project them onto the normalized image plane of a specific keyframe. The texture of this projected triangle is taken and back-projected to the 3D mesh triangle.

We constrain the size of the undistorted image to be defined in a square. Note that undistorting a wide angle of view image will result in a star like shape with undefined values along the bounding box edges. These undefined parts cause issues if crossed by a connecting edge of 2 projected 3D points. Figure 7 illustrates the issue. Constraining the image size has also the effect, that we limit calculation power to only the necessary pixels. Adapting the resolution of the image in the normalized plane allows further control also on the memory budget.

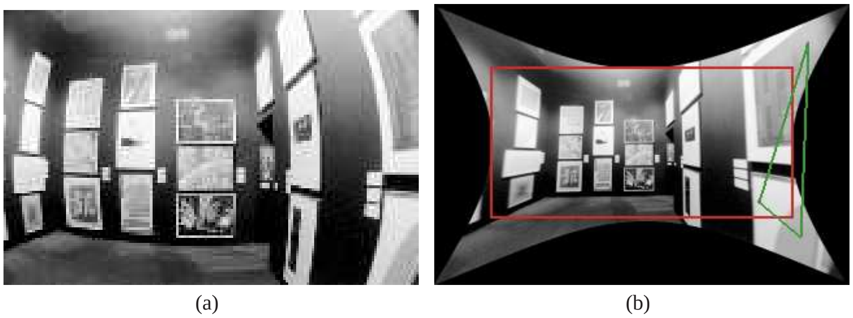

(a) (b)

Fig. 7 a) Distorted image taken by a wide angle of view lens. b) Undistorted counterpart. The values in the black areas are not defined. We only store the part in the red square in the keyframe as the texture triangle in green would cause issues in the back projection to the mesh. (Courtesy of Becker)

More formally the texture patch of each keyframe is calculated using

$$\mathbf{q} = \begin{bmatrix} x \\ y \end{bmatrix} = K^{-1} * \begin{bmatrix} u \\ v \end{bmatrix} \tag{5}$$

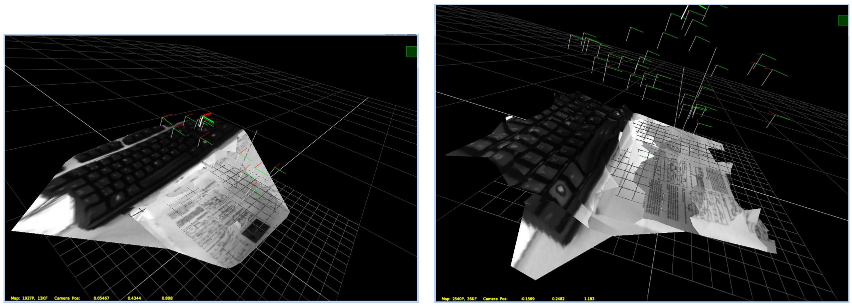
where (x,y) are the normalized image coordinates in the $z = 1$ plane of the camera frame satisfying the condition to be in the red square depicted in Figure 7(b). (u,v) are the pixel coordinates in the distorted camera image and K is the intrinsic camera matrix of the calibrated camera. We use the simple nearest neighbor for pixel interpolation in the normalized image plane. For the projection of the 3D triangle vertices to the normalized image plane and the back-projection of the texture triangle in the undistorted image back to the 3D mesh, we use

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = Q * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{6}$$

where (x,y) are the normalized coordinates, (X,Y,Z) the 3D point coordinates and Q the projection matrix obtained from the keyframe pose.

Since each keyframe has only stored one viewpoint we check for each triangle in which keyframe all three vertices of the triangle in question are visible. We select the keyframe in which the vertices are furthest apart of each other while still being in the keyframe image (i.e. we select the keyframe closest to the selected triangle). This allows maximal resolution for the texture projection and thus most information for the user.

Figure 8 shows two versions of the final textured 3D mesh. In Figure 8(a) we used a low resolution undistorted image in order to save memory whereas in Figure 8(b) we doubled the resolution. Note that higher resolution only affects the memory as for each key frame one has to store this undistorted image. It does barely affect speed since undistorting and saving the image only takes place once a key frame is created.



(a) (b)

Fig. 8 a) Textured 3D mesh of the sample scene. The keyboard and papers in front of it are clearly visible, even though not legible as the resolution of the undistorted image has been chosen low. b) Higher resolution was chosen here for the undistorted image in each key frame. Note that the keys of the keyboard are clearly legible.

To visualize better that the textured map is constructed out of several triangles we varied the camera gain and forced the algorithm to take key frames which are closer and further away to reconstruct adjacent triangles. Figure 9 shows a close-up of the sample scene. Observe that adjacent triangles were taken from different keyframe views with different gain and from different distances (thus the resolution is lower or/and the texture is darker in some triangles). The texture, however, is still very good aligned and shows thus the robustness of the algorithm.

Finally, Figure 10 shows the depth profile achieved by our method. The keyboard is accurately modeled with the 3D mesh and the texture gives the user immediately a detailed understanding of the whole 3D scene. Note also that some outliers (here in the flat part in front of the keyboard) persist despite the median filtering.

Outliers in the map give the user a wrong impression of the scene topology and - more important - let path planners and obstacle avoidance fail. Keyframes are added according to the distance to the objects and the map is updated and refined in real-time. The closer the camera is to the object the smaller is the baseline between (newly added) keyframes. This leads to a more precise map estimation for objects close to the vehicle. That is, if the aerial vehicle approaches the mesh outliers will be less probable and eventually vanish completely thanks to our median filtering. Path planners may thus not always find the shortest way to goals further away, however, obstacle avoidance is always granted to work fine.

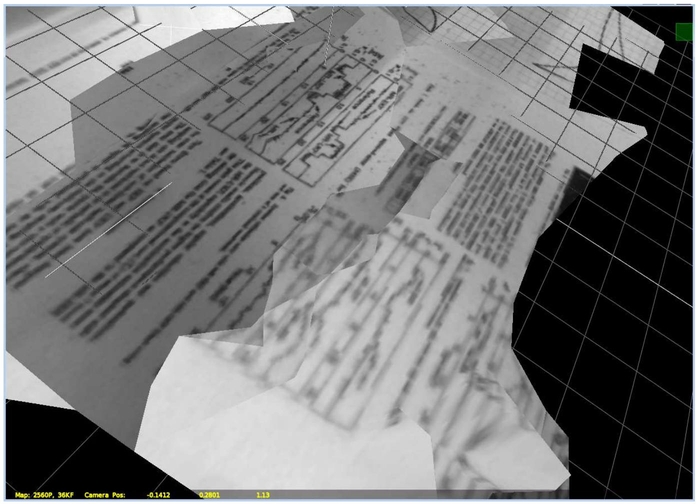


Fig. 9 Close-up of the sample scene. Note the different gains and distances enforced on the key frames taken for the reconstruction. It shows that even taken the adjacent source triangles from different key frame views the texture is well aligned.

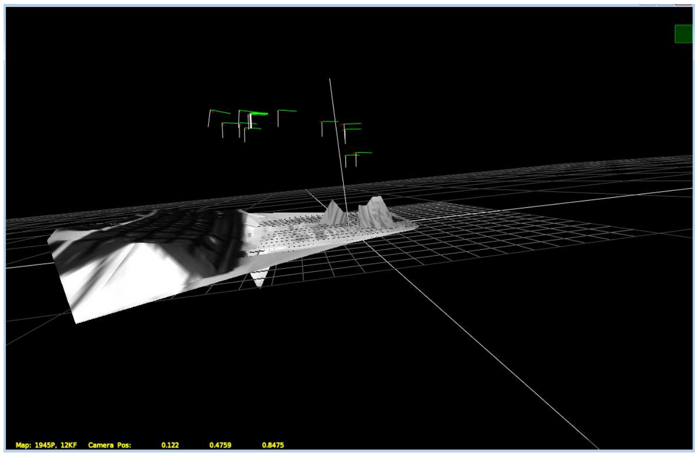


Fig. 10 Side view of the sample scene. Observe the accurate depth of the 3D textured mesh showing in detail the profile of the keyboard

5 Results on Unmanned Exploration and Obstacle Avoidance

In the previous sections we described our approach to build an intuitive 3D textured mesh map based on a point cloud from a monoSLAM framework. For illustration purpose we used a hand held camera and a small sample scene. We can extend our approach from the sample scene directly to a larger environment as a single camera is not affected by scale. Note however that the algorithm is limited in size of the scene wrt. the number of features as the visual SLAM framework is limited to a finite number of point features and key frames. We use our visual controller framework to guide an MAV in order to explore the scene while avoiding obstacles.

5.1 Indoor Scenario

Our indoor test arena is depicted in Figure 5.1. Note that the newspapers provide both features and a feedback of the texturing performance of the algorithm. Of course a visual approach without features will fail. We expanded our visual control framework described in section 3 so that we can set new waypoints at runtime (i.e. while the MAV is flying). As an aerial vehicle we use the quadcopter Hummingbird from Ascending Technologies mounted with a 150 degree fisheye lens on a uEye camera with WVGA resolution and global shutter. As a test and demonstration sequence we started the MAV on the table to explore the area on the floor to the right of the table. The intuitive 3D mapping ensures that we never hit the border of the table while descending to the floor and ascending to return to the starting position on the table.

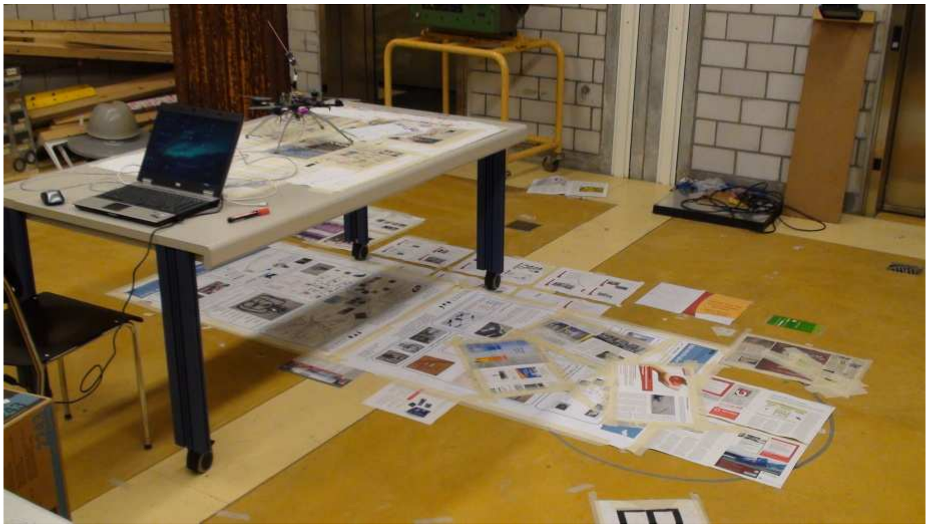


Fig. 11 Test arena for our MAV. As test sequence we stared on the table and flew to the right. As soon as the textured 3D mesh map indicated us that it is safe to descend (i.e. that we do not hit the table) we approached the floor to get a higher resolution of the floor for the map. After, we ascended and returned to the starting position on the table always using the information of the intuitive 3D map.

We performed the described sequence several times and succeeded to nearly 100%. Failures occurred very rarely and only on low batteries and communication link losses or security cord entangling. In Figure 12 we show a sample sequence of a test sequence. The left part of the left images (i.e. camera view) is shown in the bottom left part in the right images (i.e. textured 3D mesh map). In Figure 12(a) the initial map is shown. Note that the terrain is not yet accurate and the slope between table and floor is not steep enough. This is due to lack of 3D points. As we explore the area in Figure 12(b) the map improves and the table-floor step gets more and more accurate. The user can navigate the white ball in the map to set a new waypoint for the helicopter. In Figure 12(c) we safely navigated the MAV below the table level near to the floor. Note that the map got more extended and more detailed thanks to our keyframe texture selection procedure described in section 4. Observe also the differences in figures 12(c) and 12(d). While returning, more keyframes were added and allowed thus a better representation of the map. The small corrections can be seen in particular around the edges of the newspapers on the floor. On the last Figure in the sequence one can also see that the user placed the next waypoint just on the table which allows a safe landing.

In these test sequences the user always sets one waypoint after the other on runtime. It is easy to implement standard obstacle avoidance and path planning algorithm since the only information needed is the mesh surface location with respect to the vehicles location. Both information is available, the latter by the visual SLAM framework and the former by our real time map generation algorithm. A valid path for the MAV is thus one that does not traverse the mesh. This is, the MAV must stay on the same side of the mesh as it started. Autonomous exploration will then consist of flying towards the map borders close to the mesh surface for higher resolution of the map.

5.2 Outdoor Scenario

For the outdoor scenario we flew the helicopter through a small village in wintertime. This shows that even if for the human eye a (snowy) scene may look self-similar, visual approaches rarely lack on features outdoors. Also note again the scalability of our approach. We started at the desk scene, performed indoor tests of a few meters and apply it now outdoors over more than 100 meters. We used the same camera as for the indoor sequences but changed the platform to the Ascending Technology's quadcopter Pelican. Figure 13 shows the flight path through the village. We flew and mapped a path of over 100 meters at a variable flight height between 5 and 10 meters. We will pay particular attention to the one house and the creek labeled in the Figure. They show on one hand the high accuracy of the 3D map and on the other hand also limitations of our texturing heuristics.

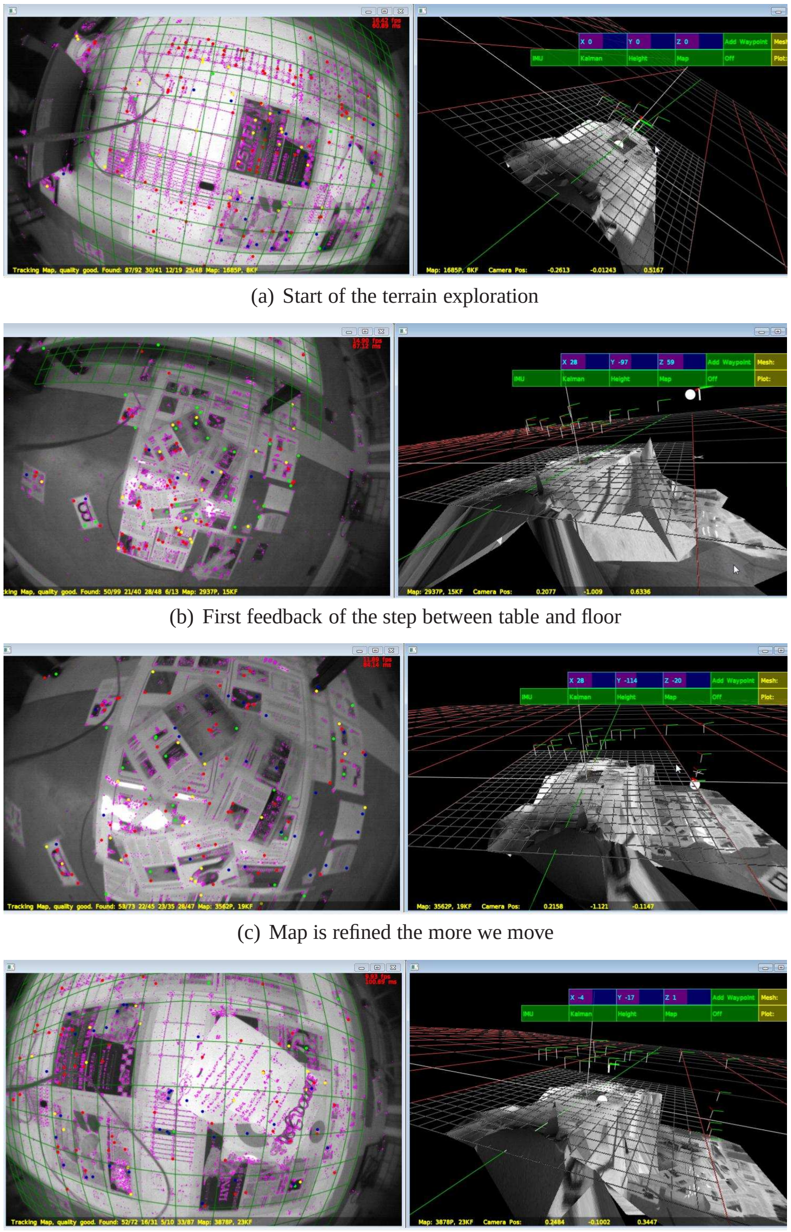
In Figure 14 the start of the mapping sequence is shown. For the human eye the scene may look self similar, however, the vision algorithm easily finds a sufficient amount of stable salient features. The camera path is plotted as faint triangles on the right side of the Figure in the 3D map view. Note also the house in the camera view, it is the house labeled in Figure 13.

In Figure 15 an overview of the first half of the total path is shown. As seen in the camera view in the Figure, the helicopter's current position is just after the junction near the bridge. Observe also the grid lines appearing along the creek, this indicates that the creek's height wrt. the rest of the surface is modeled correctly.

In Figure 16 and 17 we examine the 3D modeling and texturing accuracy. On the snow covered house in Figure 16 there were not many features extracted for the map. In particular the helicopter did not fly around the house but passed it at a high altitude. We have thus most features on the white roof. This explains why the house is modeled as a white hill rather than as a real house. This situation is explained in more detail in section 5.3 and Figure 18. Nevertheless, width, length and height of the house are meshed correctly in the conservative manner as described in earlier. The creek shown in detail in Figure 17 is less covered with snow and has a concave structure in the non-neglected directions. That is, many features have been found and the structure is not prone to texturing errors. Hence the 3D model of the Creek as well as the texture reflect the reality with a very high accuracy.

5.3 Limitation

The above test sequence shows both the strengths and weaknesses of our approach. The strength of our approach is that it works in real time and the map can thus be adjusted while exploring the environment. Rough terrain estimations are adjusted and improved gradually always keeping a



```
Tracking Map, quality good. Found: 87/92 30/41 12/19 25/48 Map: 1685P, 8KF
Map: 1685P, 8KF    Camera Pos:    -0.2613    -0.01243    0.5167
```

(a) Start of the terrain exploration

```
king Map, quality good. Found: 50/99 21/40 28/48 6/13 Map: 2937P, 15KF
Map: 2937P, 15KF    Camera Pos:    0.2077    -1.009    0.6336
```

(b) First feedback of the step between table and floor

```
Tracking Map, quality good. Found: 53/73 22/45 23/35 28/47 Map: 3562P, 19KF
Map: 3562P, 19KF    Camera Pos:    0.2158    -1.121    -0.1147
```

(c) Map is refined the more we move

```
Tracking Map, quality good. Found: 52/72 16/31 5/10 33/87 Map: 3878P, 23KF
Map: 3878P, 23KF    Camera Pos:    0.2484    -0.1002    0.3447
```

(d) Landing on table (white ball) after successful exploration

Fig. 12 The images show a typical test run to explore an unknown environment. Left: camera view. Right: online reconstructed textured 3D mesh map. One observes the refinement of the step between table and floor the more we move and the more information we have about the surroundings. Note that the transition from the table to the ground is imprecise in the first steps of the estimation. However, it is always conservative such that the vehicle does not crash as long as not traversing the mesh. Here, the waypoints are set manually at runtime. However, it is perfectly possible to use any path planning and obstacle avoidance algorithm since all information about the mesh is available.

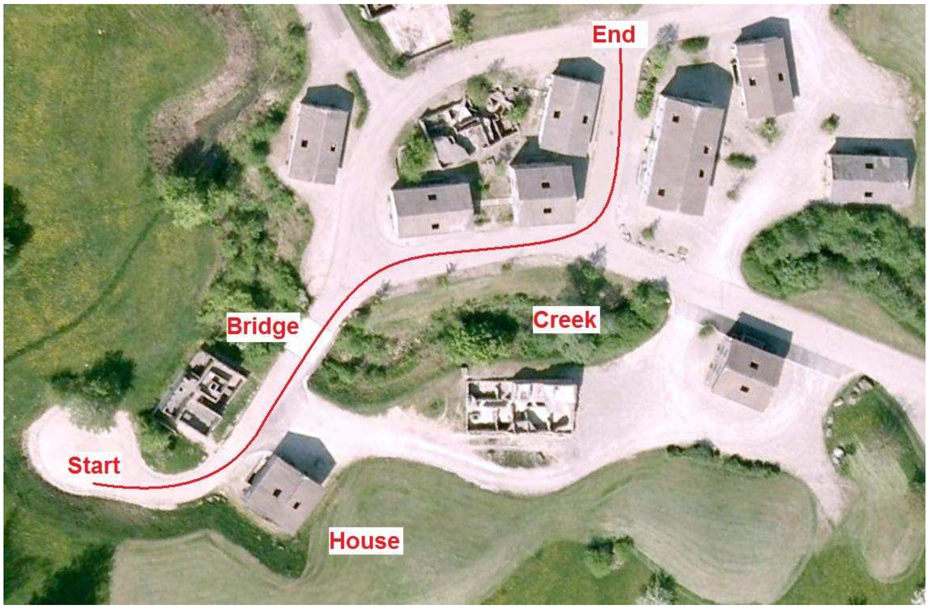
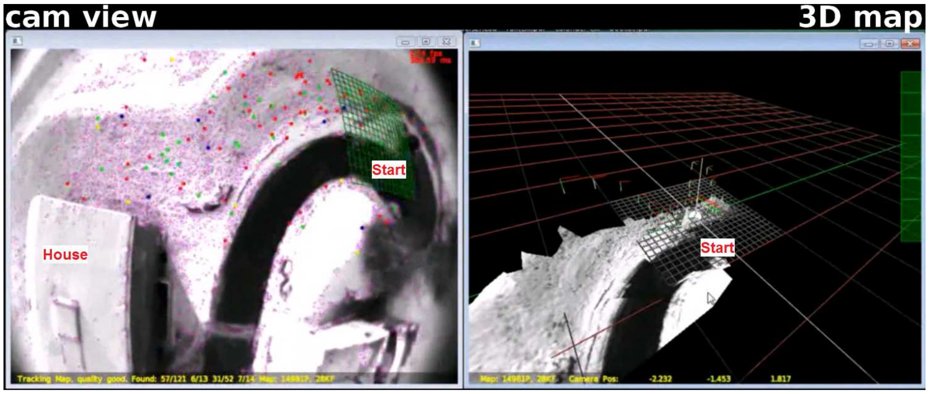


Fig. 13 Outdoor flight path through the small village. Note the labeled positions for easier understanding of the following figures. The path length is over 100m and the flight height varies between 5 and 10m [Google Maps]



Tracking Map, quality good. Found: 57/121 6/13 31/52 7/14 Map: 14981P, 28KF

Map: 14981P, 28KF Camera Pos: -2.232 -1.453 1.817

Fig. 14 Start of the mapping sequence. Observe that for the human eye the scene appears fairly self similar. However, due to the strength of (also faint) sunlight vision algorithms rarely have difficulties finding salient features in outdoor scenarios. The camera trajectory is plotted in faint triangles in the 3D map view.

conservative security boundary for the flying vehicle. However, not all concave terrain situations can be recovered. Concave terrain in the dimension which was neglected during the Delaunay Triangulation cannot be estimated correctly. In the test sequence for example the empty space between the table platform and the floor is filled with triangles. The texture is corrupted as only a few pixels are mapped to that area. This issue can be tackled by applying methods such as in [9], however, this has still to be proven to work in real time for an MAV.

Our algorithm comes also to its limits on sudden terrain changes and occlusions as no visibility constraints to the 3D point cloud are applied. Figure 18 demonstrates the problem. A Feature

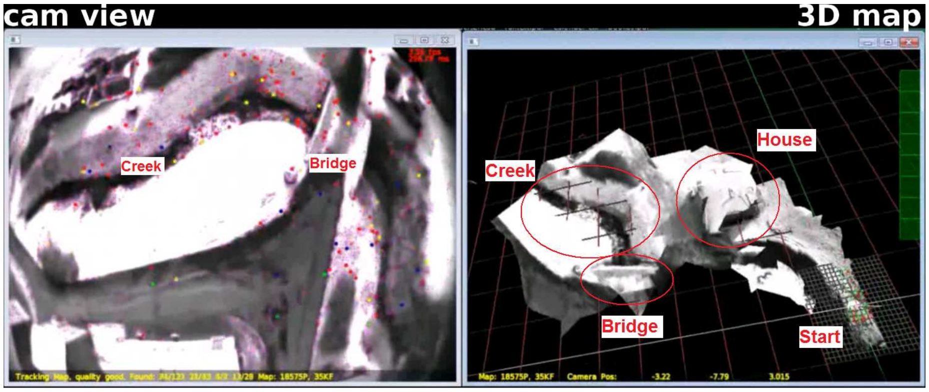


Fig. 15 Overview of the first half of the total path. Compare the labeled locations with the ones in Figure 13

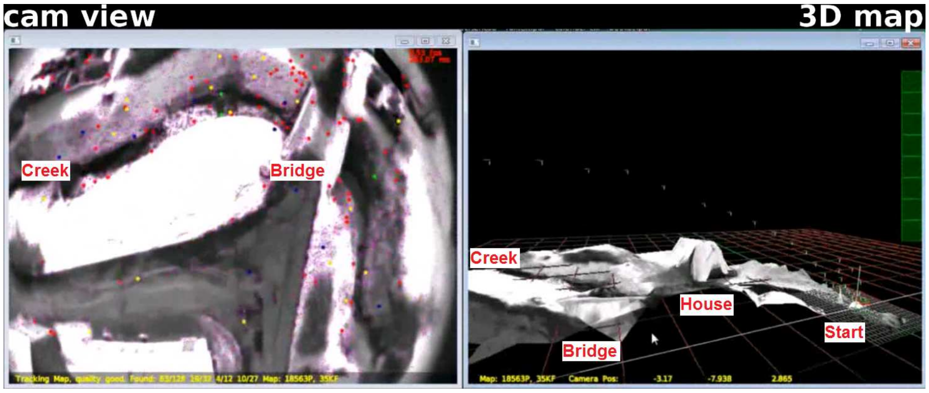


Fig. 16 Side view of the terrain to show the correctly modeled house elevation. Note that even though the 3D model of the house is very accurate, the texturing indicates some failures. We discuss this issue in section 5.3. Nevertheless, for autonomous obstacle avoidance and path planning the mesh model is sufficiently accurate.

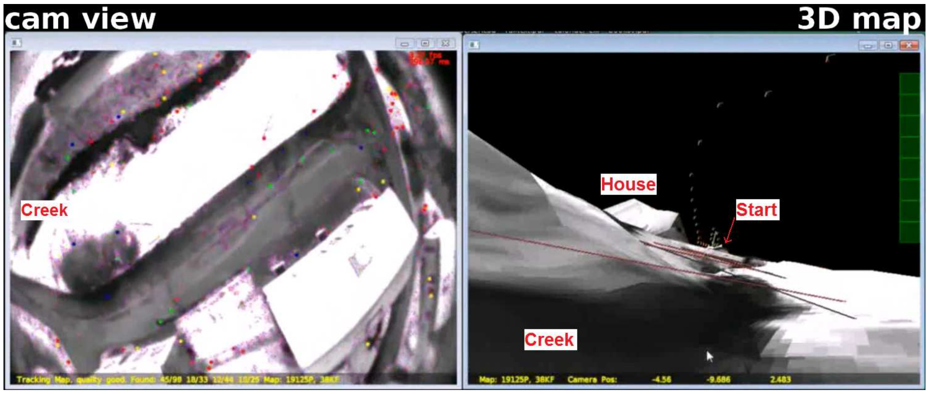


Fig. 17 View along the creek. Due to the richness of features in this area the 3D mesh model is very accurate. Moreover, thanks to the concave structure in xy-direction our texturing heuristics are prone to errors. Hence, the 3D model as well as the texture represents the reality with high accuracy. The lines crossing the creek are the reference lines of the $z = 0$ plane spanned by the vision algorithm.

F3 is seen by the keyframe KF2 far away. As the keyframe KF1 is closer to the features F2 and F3 the red texture seen by KF1 is taken for texturing the according triangle. Taking into account that KF1 does not see the feature F3 an optimized algorithm would take KF2's texture. This failure, however, only affects slightly the visual appearance of the map, not its topology (i.e. path planners or obstacle avoidance are not influenced).

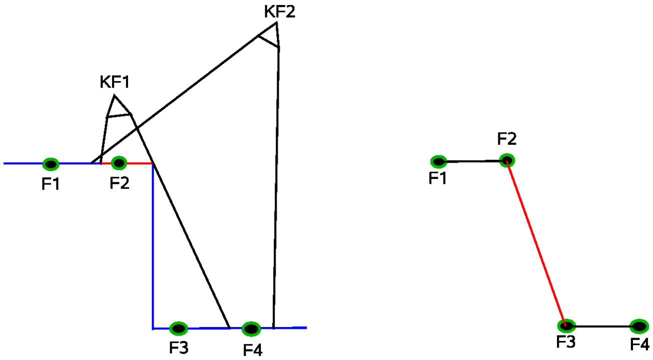


Fig. 18 Selecting the keyframe closest to the triangle in order to get the highest resolution for the mesh map texture is not always correct. Here, because of not taking visibility into account, a wrong texture is mapped to the slope. It is taken from the closer keyframe KF1 instead of the better choice KF2. This failure, however, does not affect the map's topology.

As we do not use GPS to eliminate drifts, the map will never be globally consistent. Making loops while exploring new terrain minimizes drifts, but does not eliminate them. Our approach, however, does not rely on a globally consistent map. The 3D mesh will locally be accurate enough to prevent the helicopter to crash against obstacles and to provide the user with detailed information. This is because it uses locally extracted point features. In a larger map, the total shape of the map may differ from reality. But the user and the MAV still have all information needed to perform path planning, obstacle avoidance and terrain exploration with respect to the map's coordinate frame. This leads to the conclusion that for our tasks a locally consistant map is sufficient.

Last, it is clear that our vision based framework (the MAV controller and map reconstruction) does reach its limitation on texture less scenes. Small texture free areas can be well handled assuming they are flat. Note that in our test arena the floor was texture free. However, flying at around 2m height would have provided enough features from the surroundings to make our algorithms already work. Nevertheless, we used texture rich newspapers in order to perform our maneuver - which was approaching the floor. Also in outdoor scenarios usually there are plenty of features even on cloudy days.

6 Conclusion

In this work we presented a method to reconstruct in real time an intuitive dense textured 3D mesh map out of a sparse point cloud from any given visualSLAM algorithm. Moreover, we showed successful removal of point feature outliers aid of median filtering the mesh. The real time map construction allowed us to iteratively refine, improve and extend the map. As the mesh represents a conservative boundary to objects and to non traversable areas it can directly be used in standard obstacle avoidance and path planning algorithms. This allows fully autonomous efficient terrain exploration. It can be applied for autonomous rescue and surveillance tasks as well as in desaster areas. We proved the feasibility of such an application by manually set new waypoints to fly around obstacles and explore new terrain only having the reconstructed mesh map as feedback to the user.

References

1. M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart, “Vision based mav navigation in unknown and unstructured environments,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, May 2010.
2. G. Klein and D. Murray, “Parallel tracking and mapping for small ar workspaces,” in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR’07)*, Nara, Japan, November 2007.
3. A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer*, vol. 22, no. 6, pp. 46–57, 1989.
4. R. Zask and M. Dailey, “Rapid 3d visualization of indoor scenes using 3d occupancy grid isosurfaces,” in *ECTI-CON*, 2009.
5. H. Chen and Z. Xu, “3d map building based on stereo vision,” in *Networking, Sensing and Control, 2006. ICNSC ’06. Proceedings of the 2006 IEEE International Conference on*, 0-0 2006, pp. 969–973.
6. A. M. S. Lacroix, I. Jung, “Digital elevation map building from low altitude stereo imagery,” in *Proc. Of the 9 th Int. Symposium on Intelligent Robotic Systems*, 2001.
7. S. Hrabar and G. S. Sukhatme, “Combined optic-flow and stereo-based navigation of urban canyons for a uav,” in *In Proceedings of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, 2005, pp. 2–6.
8. P. Biber, S. Fleck, F. Busch, M. Wand, T. Duckett, and W. Strasser, “3d modeling of indoor environments by a mobile platform with a laserscanner and panoramic camera,” in *European Signal Processing Conference (EUSIPCO)*, 2005.
9. R. Triebel, P. Pfaff, and W. Burgard, “Multi-level surface maps for outdoor terrain mapping and loop closing,” in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006.
10. P. Jenke, B. Huhle, and W. Strasser, “Selflocalization in scanned 3dtv sets,” in *3DTV CON - The True Vision*, 2007.
11. Paalanen, V. P. Kyrki, and J.-K. Kamarainen, “Towards monocular on-line 3d reconstructiontowards monocular on-line 3d reconstruction,” vol. version 1, 30 Sep 2008.
12. C. Kemp, “Visual control of a miniature quad-rotor helicopter,” Ph.D. dissertation, 2006.
13. A. Davison, I. Reid, N. Molton, and O. Strasse, “Monoslam: Real-time single camera slam,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.