A Fully Autonomous Indoor Quadrotor

Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard

***Abstract*—Recently, there has been increased interest in the development of autonomous flying vehicles. However, as most of the proposed approaches are suitable for outdoor operation, only a few techniques have been designed for indoor environments, where the systems cannot rely on the Global Positioning System (GPS) and, therefore, have to use their exteroceptive sensors for navigation. In this paper, we present a general navigation system that enables a small-sized quadrotor system to autonomously operate in indoor environments. To achieve this, we systematically extend and adapt techniques that have been successfully applied on ground robots. We describe all algorithms and present a broad set of experiments, which illustrate that they enable a quadrotor robot to reliably and autonomously navigate in indoor environments.**

***Index Terms*—Navigation, quadrotor, simultaneous localization and mapping (SLAM), unmanned aerial vehicle (UAV).**

IN RECENT years, the robotics community has shown an increasing interest in autonomous aerial vehicles, especially quadrotors. Low-cost and small-size flying platforms are becoming broadly available, and some of these platforms are able to lift relatively high payloads and provide an increasingly broad set of basic functionalities. This directly raises the question of how to equip them with autonomous navigation abilities. However, as most of the proposed approaches for autonomous flying [14], [32] focus on systems that are for outdoor operation, vehicles that can autonomously operate in indoor environments are envisioned to be useful for a variety of applications that include surveillance and search and rescue [10]. In such settings and compared with ground vehicles, the main advantage of flying devices is their increased mobility.

As for ground vehicles, the main task for an autonomous flying robot consists in reaching a desired location in an unsupervised manner, that is, without human interaction. In the literature, this task is known as *navigation* or *guidance*. To address the general task of navigation one is required to tackle a set of problems that range from state estimation to trajectory planning. Several effective systems for indoor and outdoor navigation of ground vehicles are currently available [1], [2].

Manuscript received November 5, 2010; revised May 23, 2011; accepted July 19, 2011. Date of publication August 30, 2011; date of current version February 9, 2012. This paper was recommended for publication by Associate Editor M. Sitti and Editor W. K. Chung upon evaluation of the reviewers' comments. This work was supported by the European Community under grant FP6-IST-034120 Micro/Nano-based systems.

S. Grzonka is with the Albert-Ludwigs-University of Freiburg, 79110 Freiburg, Germany (e-mail: grzonka@informatik.uni-freiburg.de).

G. Grisetti is with the University of Rome "La Sapienza", 00185 Rome, Italy (e-mail: grisetti@informatik.uni-freiburg.de).

W. Burgard is with the University of Freiburg, 79110 Freiburg, Germany (e-mail: burgard@informatik.uni-freiburg.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2011.2162999

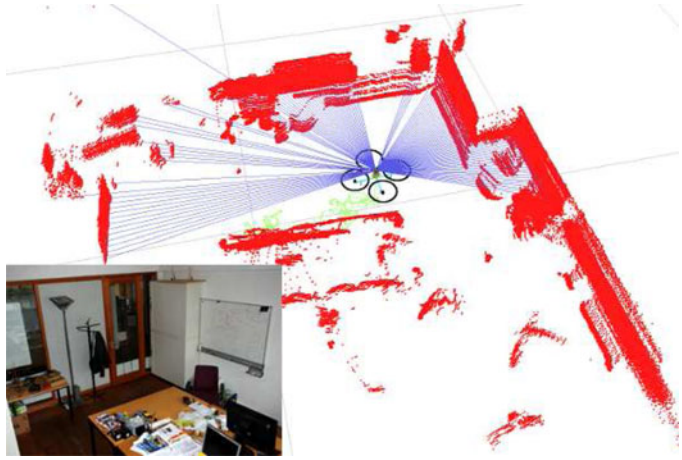


Fig. 1. Autonomous flight of our quadrotor in a cluttered office room. The free space around the robot is seriously confined, imposing high demands on pose stability, localization, and control. The image on the bottom left shows the office room from a similar view point as the snapshot.

However, the general principles of the navigation algorithms, which have been successfully applied on ground robots, could in principle be transferred to flying vehicles, this transfer is not straightforward for several reasons. Ground robots are inherently stable, in the sense that by issuing a zero-velocity command results in the robot to smoothly decelerate until it stops. The same does not apply for flying robots that need to be actively stabilized, even when they are already in the desired location. Furthermore, because of the fast dynamics of a flying vehicle compared with a ground one all the quantities necessary to stabilize the vehicle should be computed within a short time and with an adequate level of accuracy. Thus, porting navigation systems for ground robots to aerial vehicles requires to fulfill more stringent constraints on both accuracy and efficiency.

In this paper, we present the technology that enables autonomous quadrotor navigation in indoor environments and describe a navigation system that includes key functionalities namely localization, planning, surface estimation, map-learning, and control. However, as a flying vehicle moves in 3-D, indoors, there is usually enough structure to describe the environment with 2-D representations. Instead of using a full 3-D representation we rely on a 2-D one for the walls augmented with the elevation of the floor. The advantage of this choice compared with the full 3-D representation is that we can operate in a large class of indoor environments by using efficient variants of 2-D algorithms that work on dense grid maps instead of space and time consuming 3-D methods. Having these functionalities adapted for the 3-D case would be either too slow or not accurate enough given the limited time constraints to make the system stable. This paper extends our previous work [17] by introducing improved algorithms for simultaneously estimating

the altitude of the vehicle and the elevation of the underlying surface. We furthermore provide quantitative results of our simultaneous localization and mapping (SLAM) approach and discuss the effect of different modes of the incremental scan-matching on the pose stability of the robot. We also describe our algorithms for path planning and obstacle avoidance and provide additional details and experiments.

Our system is a result of an integrated hardware/software design that meets several of the challenging constraints imposed by small-sized flying vehicles, while preserving a large degree of flexibility. It further can be operated at different levels of autonomy. It can be used to assist a pilot by preventing collisions with obstacles and keeping the position of the vehicle when no commands are given. It can construct a map on-line, while flying in an unknown environment, or it can be instructed to autonomously reach given locations in a known map. We evaluated our system on an open source quadrotor, the so-called Mikrokopter [3]. Fig. 1 visualizes our quadrotor system and its internal state, while autonomously flying within a highly cluttered office room.

I. Related Work

In the last decade, flying platforms received an increasing attention from the research community. Many authors focused on the modeling and on the control of these vehicles [8], [11], [25], [29], with a particular emphasis on small- or microhelicopters [10]. Hoffmann *et al*. [19] presented a model-based algorithm for autonomous flying with their STARMAC-quadrotor. Their system flies outdoors and utilizes GPS and inertial measurement unit measurements. Ng and colleagues [14] have developed algorithms for learning controllers for autonomous helicopter navigation. Their approach allows helicopters to perform impressive maneuvers in outdoor environments. Scherer *et al*. [28] describe algorithms for flying fast among obstacles at low altitude by the usage of a laser scanner. Tempelton *et al*. [30] demonstrate how to use vision for outdoor terrain mapping and autonomous landing. Tournier *et al*. [33] and Bourquardez *et al*. [12] used vision to estimate and stabilize the current pose of a quadrotor. Thrun *et al*. [32] used a remotely controlled helicopter to learn large-scale outdoor 3-D models. There has also been some work that addressed the navigation of flying vehicles in indoor environments and in absence of the GPS signal. Several authors used vision to control or assist the control of an indoor quadrotor [7], [20], [21]. Roberts *et al*. [26] used ultrasound sensors for controlling a flying vehicle in a structured testing environment, while He *et al*. [18] presented a system for navigating a small-sized quadrotor without GPS. Here, the pose of the vehicle is estimated by an unscented Kalman filter. Whenever the robot has to reach a given location, a path that ensures a good observation density is computed from a predefined map. These highly dense observations minimize the risk of localization failures. In parallel to our work, Achtelika *et al*. [6] developed an indoor autonomous quadrotor equipped with a laser range scanner and cameras enabling autonomous hovering in a constraint indoor environment. Recently, Celik *et al*. [13] presented their system for indoor SLAM by using a monocular camera and ultrasound. Our paper is orthogonal to a recent work of Bachrach *et al*. [9], where the authors present a system for performing autonomous exploration and map acquisition in indoor environments. They extend the 2-D robot navigation toolkit CARMEN [27] by adding a Rao–Blackwellized particle filter for SLAM and an algorithm for frontier-based autonomous exploration. However, they do not provide localization, map optimization, obstacle avoidance, or mutlilevel SLAM. Furthermore, we utilize a more robust graph-based SLAM algorithm in our system that allows for map optimization and present our algorithm for estimating the altitude of the surface underlying the robot. This enables a quadrotor equipped with our system to fly over surfaces whose height is piecewise constant.

II. Indoor Navigation of an Autonomous Flying Quadrotor

To autonomously reach a desired location, a mobile robot has to be able to determine a collision-free path connecting the starting and the goal locations. This task is known as *path planning* and requires a map of the environment to be known. Usually, this map has to be acquired by the robot itself by processing the sensor measurements that are obtained during an exploration mission. This task of generation of the map is known as *simultaneous localization and mapping* (SLAM). For most of the applications it is sufficient to perform SLAM off-line on a recorded sequence of measurements. To follow the path with a sufficient accuracy, the robot needs to be aware of its position in the environment at any point in time. This task is known as *localization*. A further fundamental component of a navigation system is the *control module* that aims to move the vehicle along the trajectory, given the pose estimated by the localization.

Because of the increased risk of damaging the flying platform during testing, the user should have the possibility of taking over the control of the platform at any point in time. Finally, the more complex dynamics of a flying platform poses substantially higher requirements on the accuracy of the state estimation process than for typical ground-based vehicles. Although in outdoors scenarios, positioning errors up to 1 m might be acceptable, they are not indoors, as the free-space around the robot is substantially more confined.

III. Hardware Architecture

Fig. 2 shows a Mikrokopter [3] open source quadrotor equipped with sensors and computational devices. The Mikrokopter comes with a low-level controller for roll, pitch, and yaw. Our quadrotor is similar to the one proposed by He *et al*. [18] and consists of the following components: (1) a Hokuyo-URG miniature laser sensor for SLAM and obstacle avoidance, (2) an XSens MTi-G microelectromechanical systems IMU for estimation of the attitude of the vehicle, (3) a Linux-based Gumstix embedded PC with USB interfaces and a WiFi network card that communicates with the microcontroller on the quadrotor via an RS-232 interface, (4) and a mirror that is used to deflect some of the laser beams along the z-direction to measure the distance to the ground.

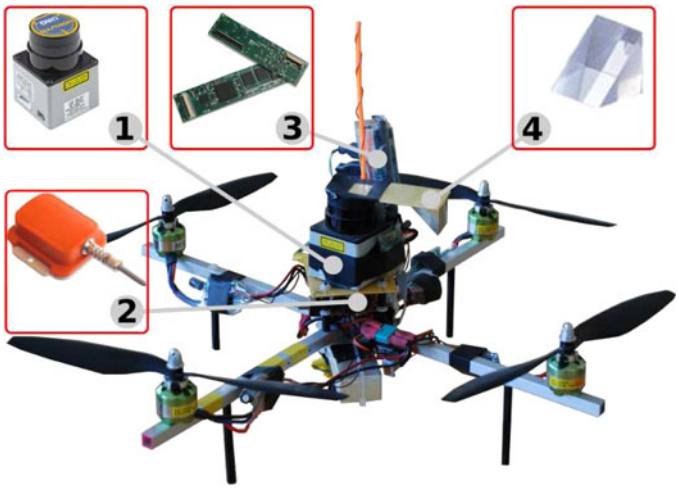


Fig. 2. Quadrotor platform that is used to evaluate the navigation system is based on a Mikrokopter and includes (1) a Hokuyo laser range finder, (2) XSens IMU, (3) Gumstix computer, and (4) laser mirror.

IV. Navigation System

Our navigation system is based on a modular architecture in which different modules communicate via the network by using a publish–subscribe mechanism. In our current system, all device drivers are executed on-board, while the more computationally intensive algorithms run on a remote PC communicating over wireless with the platform.

Since roll (ϕ) and pitch (θ) measured by the IMU are in general accurate up to 1°, we can directly use this information within our navigation system. This allows us to reduce the localization problem from 6 degrees of freedom (DOF) namely $(x, y, z, \phi, \theta, \psi)$ to 4DOF, consisting of the 3-D position (x, y, z) and the yaw angle ψ. The only sensor that is used to estimate these 4DOF and detecting obstacles is the laser range scanner.

Based on known initial calibration parameters and on the current attitude (ϕ, θ) that is estimated by the IMU, we project the endpoints of the laser into the global coordinate frame. Given the projected laser beams, we estimate the (x, y, z, ψ) of the vehicle in a 2-D map containing multiple levels per cell. To compensate for the lack of odometry measurements we estimate the incremental movements in (x, y, ψ) by 2-D laser scan-matching. Finally, we control the altitude of the vehicle and simultaneously estimate the elevation of the underlying surface by fusing the IMU accelerometers and the distance from the ground measured by the laser. Accordingly, we track and map multiple levels within an environment, which enables our robot to correctly maintain its height, even when it flies over obstacles such as tables or chairs.

A. Incremental Motion Estimation

The laser range scanner measures at time t a set of distances $\mathbf{r}_t$ along the x–y plane in its own reference frame. We therefore first calculate a projection of the measured distances $\mathbf{b}_t$ for the beams not deflected by the mirror that uses the roll and pitch estimate from the IMU. Consequently, we calculate the points $\mathbf{h}_t$ for all beams deflected by the mirror that uses a chain of transformations from the IMU to the *virtual* laser position that accounts for the effect of the mirror. Some tasks, like pose stabilization, rely on an accurate local pose estimate of the vehicle in its surroundings. To this end, we can estimate the relative movement of the robot between two subsequent scans by the usage of a scan-matching algorithm. Since the attitude is known from the IMU, this procedure can be carried out in 2-D, assuming structured indoor environments. A scan-matching algorithm estimates the most likely pose of the vehicle $\hat{\mathbf{x}}_t$ at time t given the previous k poses $\mathbf{x}_{t-k:t-1}$ and the corresponding laser measurements $\mathbf{b}_{t-k:t}$, as follows:

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}:=(x,y,\psi)}{\operatorname{argmax}} \; p(\mathbf{x}_t \mid \mathbf{x}_{t-k:t-1}, \mathbf{b}_{t-k:t}). \tag{1}$$

To solve (1), we use a variant of the multiresolution correlative scan matcher proposed by Olson [24]. The idea behind a correlative scan-matcher is to discretize the search space $\mathbf{x}_t = (x_t, y_t, \psi_t)$ and to perform an exhaustive search in these discretized parameters around a given initial guess. To efficiently evaluate the likelihood $p(\mathbf{x}_t \mid \mathbf{x}_{t-k:t-1}, \mathbf{b}_{t-k:t})$ of a given solution $\mathbf{x}_t$, we use likelihood fields [31] obtained by the most likely map that is generated from the last observations $\mathbf{b}_{t-k:t-1}$.

The complexity of a correlative scan-matcher depends linearly on the resolution at which the parameters are discretized and on the search range. A naive implementation of this algorithm is not adequate for our application that demands both high accuracy and efficient computation. To overcome this problem, we employ a multiresolution approach. The idea is to perform the search at different resolutions: from coarse to fine. The solutions that are found at a coarse level are then used to restrict the search at a higher resolution.

In our implementation, we use a constant velocity model to compute the initial guess for the search, and we perform the correlative scan matching at three different resolutions (i.e., $4\,\text{cm} \times 4\,\text{cm} \times 0.4^\circ$, $2\,\text{cm} \times 2\,\text{cm} \times 0.2^\circ$, and $1\,\text{cm} \times 1\,\text{cm} \times 0.1^\circ$). We set the search area r depending on the maximum speed $v_{\max}$ of the vehicle and on the frequency f of the scanner as $r = v_{\max}/f$.

We control the position of the vehicle that is based on the velocities that are estimated by the scan-matcher. Accordingly, the performances of the scan-matcher play a major role in the stability of the robot. In particular, we want to have a fast, accurate but still smooth (i.e., less oscillations) estimate. To get an intuition about the desired accuracy, consider an error in the position estimate of ± 2 cm. Assuming a sensor frequency of 10 Hz this error leads to a variation of 20 cm/s in the velocity estimate between two laser scans. This in turn can generate wrong commands by the controller reducing stability.

In our hierarchical scan-matcher, the high-resolution estimate is affected by frequent oscillations because of the limited resolution of the likelihood field. Although these oscillations could in general be filtered out by a low-pass filter, this type of filtering would introduce a phase shift in the pose and velocity estimate (the estimated pose is past in time). To obtain both, an accurate position estimate and a smooth signal, we compute the final solution as the weighted mean of the estimates of all scan-matchers in the hierarchy. The weights of the sum lie on a Gaussian that is centered at the finest resolution estimate. In

TABLE I
EFFECT OF MATCHING ALGORITHM ON POSE STABILITY OF THE ROBOT

approach →	4 cm	2 cm	1 cm	weighted mean	unit
mean(x)	0.107	0.105	0.149	0.066	$[m]$
mean(y)	-0.045	0.060	-0.04	-0.05	$[m]$
std(x)	0.145	0.148	0.165	0.123	$[m]$
std(y)	0.081	0.088	0.087	0.076	$[m]$
mean($\|v_x\|$)	0.146	0.095	0.084	0.075	$[m/s]$
mean($\|v_y\|$)	0.159	0.106	0.09	0.072	$[m/s]$
std($\|v_x\|$)	0.118	0.071	0.065	0.058	$[m/s]$
std($\|v_y\|$)	0.117	0.083	0.072	0.057	$[m/s]$

several experiments, we found that the weighted average of the estimates, is better for control as each single estimate, as shown in Table I. The table contains experimental results that compare the effect on the pose stability by using the estimate of the individual scan-matchers versus our weighted mean approach. All runs reflect experiments where the goal of the quadrotor was to hover at the same spot at 0.5 m height for as long as the battery holds. To quantitatively evaluate our approach, we compare the mean and standard deviation in both, position and absolute velocity. As can be seen, usage of a weighted average of the different resolutions has a positive affect on the control loop. This originates from the fact that the weighted averaging has a smoothing effect on the pose estimate but does not include any phase shift into the system. Since we use a simplistic model of our quadrotor (constant velocity model), by using the output of the weighted mean (with the prediction used as the initial guess for the search) is equal to running a Kalman filter that has a large uncertainty on the prediction. However, as including a more sophisticated model for the prediction would lead to better estimates, the usage of this simplistic strategy was sufficient for our purposes.

B. Localization and SLAM

If a map of the environment is known *a priori*, pure localization (in contrast with SLAM) is sufficient for the estimation of the remaining 4DOF of the quadrotor. We estimate the 2-D position (x, y, ψ) of the robot in a given grid-map by Monte Carlo localization [15]. The idea is to use a particle filter to track the position of the robot. Here, we sample the next generation of particles according to the given relative movement estimated by the scan matcher and evaluate the current particle by using likelihood fields [31].

Our system can acquire models of unknown environments during autonomous or manual flights by simultaneously localizing and mapping the environment. The goal of a SLAM algorithm is to estimate both the vehicle position and the map of the environment by processing a sequence of measurements acquired, while moving in the environment. Even when a map is known *a priori*, a local map is needed until the robot is localized if the robot is running autonomously. In our system, we use a popular graph-based SLAM algorithm. The idea of these types of algorithms is to construct a graph from the measurements of the vehicle. Each node in the graph represents a position of the vehicle in the environment and a measurement taken at that position. Measurements are connected by pairwise constraints encoding the spatial relations between nearby robot poses. These relations are determined by matching pairs of measurements acquired at nearby locations. Whenever the robot re-enters a known region after traveling for long time in an unknown area, the errors accumulated along the trajectory become evident. These errors are modeled by constraints connecting parts of the environment that have been observed during distant time intervals and are known in the SLAM community as *loop closures*. To recover a consistent map we use a stochastic gradient descent optimization algorithm that finds the position of the nodes that maximizes the likelihood of the edges. The optimization approach is discussed in detail in [16], and an open source version is available on OpenSLAM [4].

Again, we restrict our estimation problem to 4DOF, since the attitude provided by the IMU is sufficiently accurate for our mapping purposes. Furthermore, we assume that the vehicle flies over a piecewise constant surface and that the indoor environment is characterized by vertical structures, such as walls, doors, and so on. Although trash bins, office tools on a table or the table itself are violating this assumption by the usage of a 2-D map is still sufficient for accurate mapping and localization. This arises from the fact that clutter in general is only visible in a small portion of the current measurement, whereas mapping, for example, the desk improves localization since there is a clear difference in x-y between a desk and a nearby wall. Thus, we restrict our approach to estimate a 2-D map and a 2-D robot trajectory spanning over 3DOF, (x, y, ψ), i.e., we map all objects if they had an infinite extend. The estimate of the trajectory is the projection of the 6DOF robot motion on the ground plane, along the z-axis. We estimate the altitude of the platform once the 2-D position and the attitude are known, which is based on the procedure described in the next section.

C. Altitude Estimation

Estimating the altitude of the vehicle in an indoor environment means determining the global height with respect to a fixed reference frame. Since the vehicle can move over a nonflat ground, we cannot directly use the beams $\mathbf{h}$ deflected by the mirror. Our approach therefore concurrently estimates the altitude of the vehicle and the elevation of the ground under the robot. In our estimation process, we assume that the (x, y, ψ) position of the robot in the environment is known from the SLAM module described earlier. We furthermore assume that the elevation of the surface under the robot is a piecewise constant. We call each of these connected surface regions having constant altitude a "level." The extent of each level is represented as a set of cells in a 2-D grid sharing the same altitude.

Since our system lacks global altitude sensors such as barometers or GPS to determine the altitude of the vehicle, we track the altitude of the vehicle over the ground and map different elevations by a two-staged system of Kalman filters. Algorithm 1 describes our approach in an abstract manner.

In the first stage, a Kalman filter is used to track the altitude z and the vertical velocity v_z of the vehicle by the combination of inertial measurements, altitude measurements, and already mapped levels under the robot. In the second stage, a set of Kalman filters is used to estimate the elevation of the levels

currently measured by the robot. To prevent drifts in the elevation estimate, we update the altitude of a level only when the robot measures the level for the first time or whenever the robot re-enters it (i.e., enters or leaves that particular level).

In detail, the first Kalman filter estimates the height state $\mathbf{z} = (z, v_z)$ and the corresponding uncertainty Σ_z. First, we predict the current altitude and velocity ($\hat{\mathbf{z}}_t$) given the previous estimate, $\mathbf{z}_{t-1}, \Sigma_{z_{t-1}}$, and the acceleration measured by the IMU (see line 4 of Algorithm 1).

The beams deflected by the mirror can measure more than one level simultaneously. For instance, when flying over a table it can happen that one fraction of the beams is fully reflected by the table, some beams are partially reflected by the table and partially by the floor, whereas the remaining beams are fully reflected by the floor. We therefore search in the local vicinity of the current multilevel-map for all levels that could have generated one of the measured altitudes $h \in \mathbf{h}_t$ (assuming the robot's altitude is $\hat{\mathbf{z}}_t$). This step is indicated in line 5.

If we found at least one correspondence, then we use them to calculate a virtual measurement to the ground (see line 7). We use the matched levels from the current map and the corresponding beams to calculate a single measurement. In other words, we calculate the measurement we would obtain if no obstacles were present underneath the robot and use this information for the measurement update of the Kalman filter as shown in line 8.

However, when the robot explores the environment, it can happen that none of the current beams, i.e., $h \in \mathbf{h}_t$ falls into a confidence region of a level in the current map, i.e., $\mathbf{E} = \emptyset$. In this case, we cannot create a virtual measurement and thus are unable to perform a measurement update of the filter. The prediction, therefore, is then the best estimate of the robot's altitude as described in line 10.

Given the estimated altitude of the robot, we can now update the current multilevel map. Recall that the beams deflected by the mirror can measure more than one level simultaneously. We therefore cluster them into neighboring sets. Each of these sets is assumed to originate from a single level, and it is parameterized by the mean and the covariance matrix calculated by the beams in the set. The outcome of this process is the set $\mathbf{L}$ consisting of the estimated levels as indicted in line 13.

We assume measurements not falling into a confidence region of existing levels in the local neighborhood to be generated by a new floor level. These new floor levels can be directly included into the map, as shown in line 14 in Algorithm 1. For all measurements, falling into the confidence region of a level in the map, there exist two possibilities. Either this level has been already seen in the previous time-step, i.e., the robot is flying over the table, and thus, it has seen the corresponding level before, or it is currently entering or leaving this particular level. In the latter case, we can use the current altitude estimate in order to update the altitude of the level in the map (line 15). The elevation of each level is tracked by an individual Kalman filter.

Since we explicitly store objects in 2-D with an extend in x–y rather than individual levels per cell, we seek for those levels present in the neighborhood of the map that are explained by one of the measurements that are currently obtained. If such a level is found and not present at the current location, we extend this level to the current cell, as shown in line 16.

Algorithm 1 Multilevel-SLAM

Input: beams deflected by mirror at time t: $\mathbf{h}_t$
Input: previous multilevel map: $\hat{\mathbf{M}}$
Input: elapsed time: Δt
Input: current pose: $\mathbf{x}_t = (x_t, y_t)$ // output of SLAM module
Input: previous height state $\mathbf{z}_{t-1} = (z_{t-1}, v_{z_{t-1}})$
Input: previous height state uncertainty $\Sigma_{z_{t-1}}$
Input: z-acceleration and uncertainty: a_z, σ_z // from IMU
Output: current height state: $\mathbf{z}_t, \Sigma_{z_t}$
Output: current multilevel map: $\mathbf{M}$

1: **function** Multilevel-SLAM
2: // ———— 1st stage: update height estimate ————
3: // KF is short for Kalman Filter
4: $(\hat{\mathbf{z}}_t, \hat{\Sigma}_{z_t}) = \text{KF}(\mathbf{z}_{t-1}, \Sigma_{z_{t-1}}).\text{predictionStep}(\Delta t, a_z, \sigma_z)$
5: $\mathbf{E} = \hat{\mathbf{M}}.\text{at}(\mathbf{x}_t \pm \Delta\mathbf{x}).\text{getExistingLevelsMatching}(\mathbf{h}_t, \hat{\mathbf{z}}_t)$
6: **if** $\mathbf{E} \neq \emptyset$ **then**
7: $(\tilde{m}, \tilde{\sigma}_m) = \text{createVirtualHeightMeasurement}(\mathbf{h}_t, \mathbf{E})$
8: $(\mathbf{z}_t, \Sigma_{z_t}) = \text{KF}(\hat{\mathbf{z}}_t, \hat{\Sigma}_{z_t}).\text{measurementUpdate}(\tilde{m}, \tilde{\sigma}_m)$
9: **else**
10: $(\mathbf{z}_t, \Sigma_{z_t}) = (\hat{\mathbf{z}}_t, \hat{\Sigma}_{z_t})$
11: **end if**
12: // ———— 2nd stage: update map ————
13: $\mathbf{L} = \text{estimateLevels}(\mathbf{h}_t, \mathbf{z}_t)$
14: $\mathbf{M} = \hat{\mathbf{M}}.\text{addNewLevels}(\mathbf{L}, \mathbf{x}_t)$
15: $\mathbf{M} = \mathbf{M}.\text{updateExistingLevels}(\mathbf{L}, \mathbf{x}_t)$
16: $\mathbf{M} = \mathbf{M}.\text{extendExistingLevels}(\mathbf{L}, \mathbf{x}_t)$
17: $\mathbf{M} = \mathbf{M}.\text{searchForLoopClosures}(\mathbf{x}_t)$
18: return $\mathbf{z}_t, \Sigma_{z_t}, \mathbf{M}$
19: **end function**

Note that the robot observes only a limited portion of the underlying surface. Thus, it may also happen that the robot "joins" the surfaces of different levels to form a new one. Fig. 3 illustrates this situation. Initially two levels corresponding to a chair (Level 1) and a table (Level 2) are identified (a). The robot then left the table behind, makes a turn, and flies over a different area of the same table. Since Level 2 is not mapped in the neighborhood of the current pose, our system creates a new level (for the same table), which is noted as Level 3 in (b). Finally, the quadrotor continues to the originally covered area of the table that introduces an intersection of the current Level 3 and the previously generated Level 2. As a consequence, it joins Levels 2 and 3 [see Fig. 3(c) and (d)].

When two levels, L'_j and L'_k, having altitudes h_j and h_k and covariances σ_j^2 and σ_k^2 are merged, the Gaussian estimate $< h, \sigma^2 >$ of the joint level has the following values:

$$\left\langle h = \frac{\sigma_k^2 h_j + \sigma_j^2 h_k}{\sigma_j^2 + \sigma_k^2}, \qquad \sigma^2 = \frac{\sigma_j^2 \sigma_k^2}{\sigma_j^2 + \sigma_k^2} \right\rangle. \tag{2}$$

This step is indicated in line 17 of Algorithm 1.

To summarize, we store a level as a set of 2-D grid cells representing the area covered by the corresponding object. First,

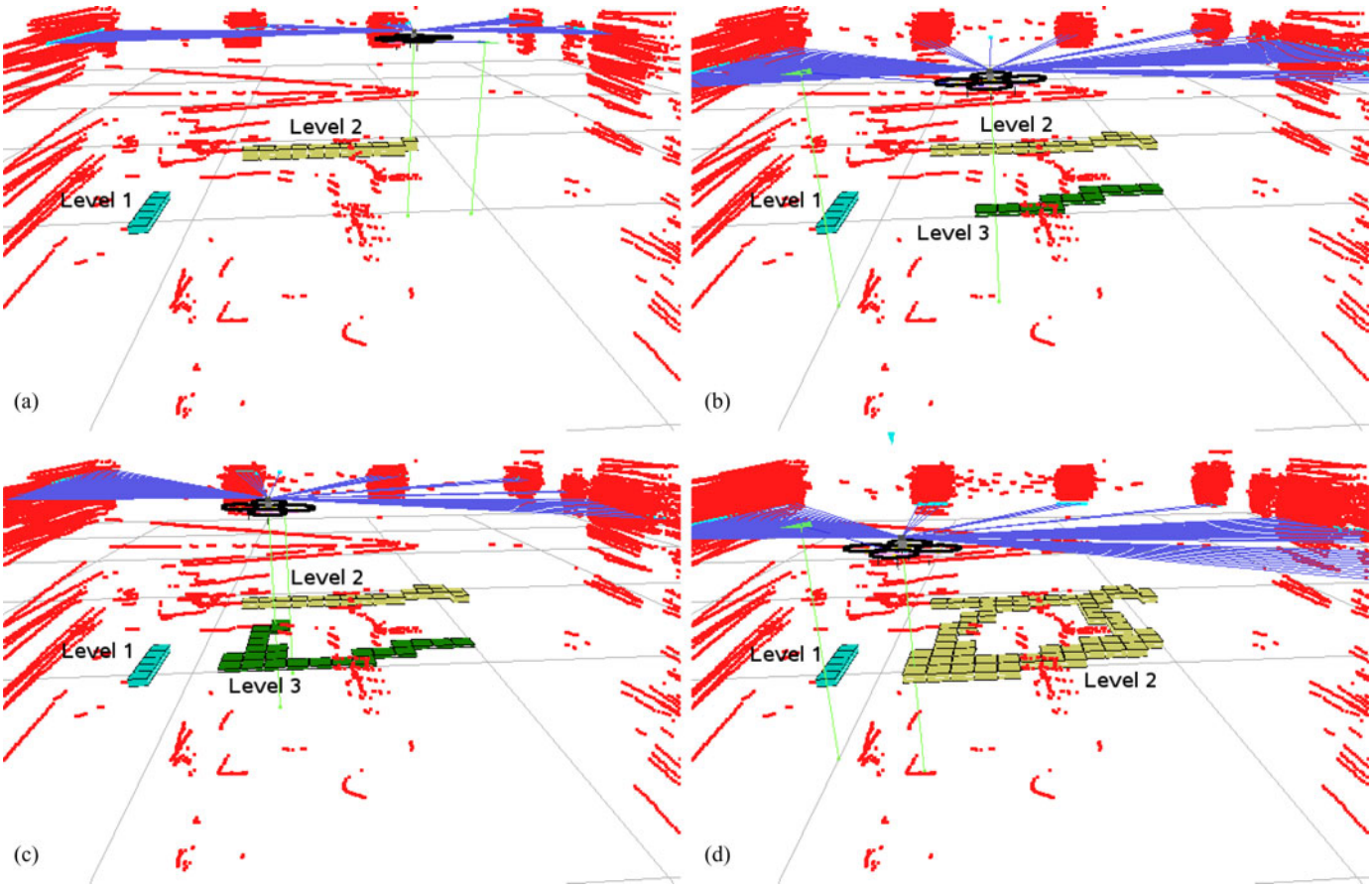


Fig. 3. Example of level joining during the estimation of the altitude of the vehicle and of the elevation of the underlying surfaces. Each level is represented as a set of contiguous cells in the 2-D grid that share the same elevation. (a) The robot starts exploring an office environment. Initially, it recognizes two levels (Level 1 and Level 2), corresponding to a chair and a table. (b) Subsequently, it flies away from the table, turns back, and flies over a different region of the same table. (c) This results in the creation of the new Level 3. Then, the robot keeps on hovering over the table until it approaches the extent of Level 2 that has the same elevation of Level 3, which is originated by the same table. This situation is shown in (c). Finally, the robot enters Level 2 from Level 3. (d) Our system recognizes these two levels to have the same elevation. Accordingly, it merges them and updates the common elevation estimate.

we estimate the current height of the robot given the known levels in the multi-level map. In a second step, we update the map, given the estimated altitude of the robot. Here, a level is constantly re-estimated whenever the vehicle enters or leaves this specific level, and the data association is resolved by the known (x, y, ψ) position of the vehicle. Finally, measurements not explained by any level that is present in the map are assumed to be generated by new levels that are then included in the map.

D. High-Level Control for Pose and Altitude

The high-level control algorithm is used to keep the vehicle in the current position. The output of the control algorithm are variations in the roll, pitch, yaw, and thrust, which are denoted, respectively, as u_ϕ, u_θ, u_ψ, and u_z. The input are the position and the velocity estimates coming from incremental scan-matching. A variation of the roll translates in a motion along the y axis, a variation in the pitch results in a motion along the x-axis and a variation of the thrust results in a change in the vertical velocity. We separately control the individual variables via proportional integral differential (PID) or proportional differential (PD) controllers. Since in our case all control commands are dependent on the current pose estimate, our high-level control module runs at a 10 Hz, since the laser scanner provides measurements at exact this rate.

Note, that the Mikrokopter (and most of commercial available platforms) comes with low-level controllers for roll, pitch, and yaw; thus, we do not have to take care about the control of the individual motors but of the control of commands resulting in a desired angle. In our particular case, the low-level controller of the Mikrokopter quadrotor runs at 500 Hz. Since commands for the yaw on common platforms result in how fast the quadrotor should turn and not how far, these parameters reflect the users wish of the robots aggressiveness with respect to the yaw rotation. In contrary to this, commands for roll and pitch result in a desired angle for which independent mapping functions must be learned. In order to learn the mapping for our quadrotor, we fixed one axis of the vehicle to an external frame allowing the vehicle to rotate only along the other axis. We learned the mapping function by monitoring the current angle measured by the IMU compared with the sent command. Our test bench for learning this mapping is shown in Fig. 4.

The calculated commands are sent directly to the microcontroller via RS232 that is in charge of the low-level control (roll,

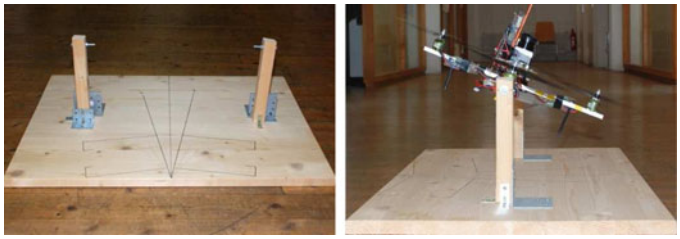


Fig. 4. Our test bench for learning a mapping between the command and the corresponding angle. This simple device allows the fixation of one axis of the quadrotor and monitoring the other one using the IMU.

pitch, and yaw) of the platform. For safety reasons, the user can always control the vehicle via a remote control and our system mixes the user and the program commands. During our experiments, we allow the programs to perturb the user commands by $\pm 20\%$. This way, if one of the control modules fails the user still has the possibility to safely land the vehicle with no loss of time since he does not need to press any button first.

In particular, we control the pitch and the roll by two independent PIDs that are fed with the x and the y coordinates of the robot pose. The control function in x is the following:

$$u_\phi = K_p \cdot (x - x^*) + K_i \cdot e_x + K_d \cdot v_x. \tag{3}$$

Here x and x^* are the measured and the desired x-positions, v_x is the corresponding velocity, and e_x denotes the error integrated over time. The control in the y is analogous to the control in x. Note, that the integral part could be omitted (i.e., $K_i = 0$), but we have encountered an improved hovering behavior if a small K_i is used. This originates from the fact that in our case only integer values can be transmitted to the microcontroller, although the desired command is a float value.

We control the yaw by the following proportional controller:

$$u_\psi = K_p \cdot (\psi - \psi^*). \tag{4}$$

Here ψ and ψ^* are the measured and desired yaw, and u_ψ is the control input.

The altitude is controlled by a PID controller that utilizes the current height estimate z, the velocity v_z, and the current battery voltage U_t, respectively. The control u_z is defined as

$$u_z = C(U_t) + K_p \cdot (z - z^*) + K_i \cdot e_z + K_d \cdot v_z \tag{5}$$

with K_p, K_i, and K_d being the constants for the P, I, and D part and $C(U_t)$ being the thrust command offset given the current battery voltage U_t respectively. Here z^* denotes the desired height and e_z denotes the integrated error. Including a thrust command offset $C(U_t)$ allows us to treat the system as stationary, and therefore to use constant coefficients for the PID. We learned $C(U_t)$ by monitoring the thrust and the battery level of the vehicle in an expectation–maximization fashion. We started with a PID control without $C(U_t)$ and computed the average thrust command required to keep the current altitude by the usage of several test flights. For each battery level U_t we computed the average thrust command required to keep the current altitude. In subsequent flights, we used this offset as an initial guess for $C(U_t)$ and repeated the experiments resulting in a refinement for $C(U_t)$ until no major change in the estimated offset appeared.

E. Path-Planning and Obstacle Avoidance

The goal of the path-planning module is to compute a path from the current location to a user-specified goal location that satisfies one or more optimality criteria and is safe enough to prevent collisions, even in the case of small disturbances. Safety is usually enforced by choosing a path that is sufficiently distant from the obstacles in the map. Finally, because of the increased DOF of a flying vehicle compared with a ground robot, the path should be planned in 4DOF space instead of 3DOF. In our system, we use D* lite [22], which is a variant of the A^* algorithm that can reuse previous solutions to correct an invalid plan rather than recomputing it from scratch. Since directly planning in 4DOF is too expensive for our system, we compute the path in two consecutive steps. First, we use D* lite to compute a path in the $x - y - z$ space, but we only consider actions that move the robot in the 2-D space $x - y$. For each (x, y) location we know from the multilevel map the elevation of the surface that is underneath the robot. This known elevation is used to determine a possible change in altitude the robot would have to take when moving to a nearby cell. A change in the altitude is reflected by increased traversability costs proportional to the distance in the z-direction. Furthermore, the cost function of a state (x, y, z) of the robot depends on the distance of that location to the closest vertical obstacle in the map.

Once we have the 2.5-D trajectory calculated with D* lite, we augment it with the ψ component. Since the laser scanner is heading forward, it is desirable that the robot turns toward the direction of flight first to avoid collisions. On the other hand, we want the quadrotor to perform small maneuvers, like flying 10 cm backwards, without turning first. To achieve this behavior we calculate the desired angle that would result in flying forward wrt. the local frame of the quadrotor. Trading off the costs of rotation versus costs of moving to the desired cell without rotating first allows the robot to perform pure sidewards or even backwards movements and thus prevents the vehicle from performing unnatural maneuvers.

Instead of switching to a new plan at every point in time, we try to re-use the existing solution whenever possible. A new plan is generated only when the actual plan is not valid anymore because of dynamic obstacles or when a certain period of time has been reached ($\Delta t = 500$ ms). The latter constraint enables us to correct for detours in the trajectory that has been introduced to avoid obstacles that are no longer present. In our implementation, we use a grid resolution of 4 cm. With these settings, the planner requires about 50–80 ms to compute a typical 10 m path from scratch. Re-planning can be done in less than 10 ms. Dynamic obstacles are detected by the planner by considering the endpoints of the laser beams that are not explained by the known map (background subtraction). Additionally, we run a reactive obstacle avoidance module on-board in parallel that is based on potential fields [23].

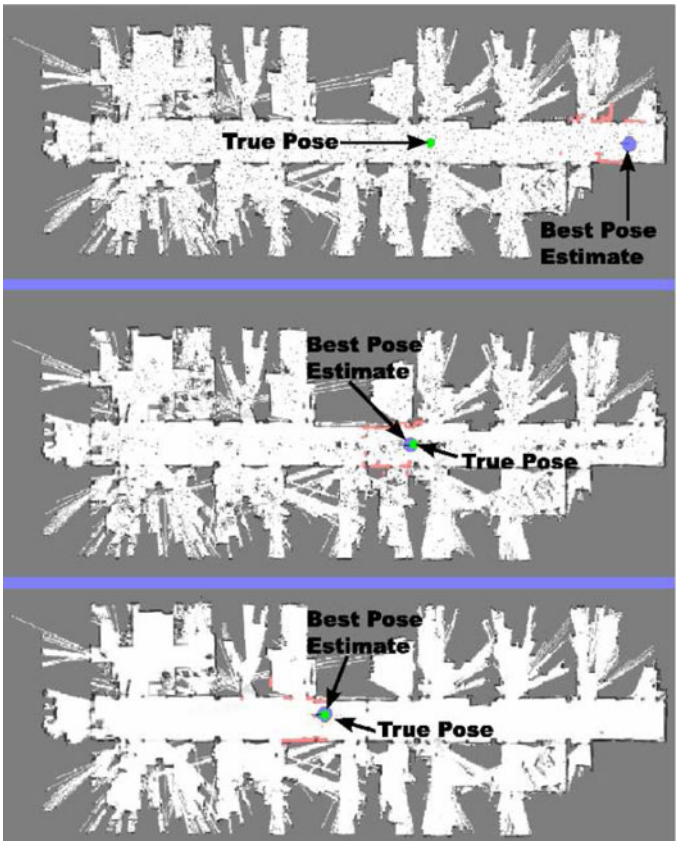


Fig. 5. Global localization of our quadrotor in a map previously acquired by a ground-based platform. The blue and the green circle highlight the current estimate of the particle filter and the true pose, respectively. Particles are shown as black dots within the free space. (Top) Initial situation. (Middle and bottom). After about 1 and 5 m of flight. In the latter case, the quadrotor is localized.

V. Experiments

In this section, we present experiments that show the performances of each of the modules presented in the previous section, namely, localization, SLAM, multilevel mapping, autonomous pose stabilization, path planning, and obstacle avoidance. Videos of a series of different flights can be found on the Web [5].

A. Localization

Using 2-D grid maps for localization enables our system to operate with maps acquired by different kinds of robots and not necessarily built by the flying vehicle itself. In this section, we present an experiment in which we perform global localization of the flying quadrotor in a map acquired with a ground-based robot. This robot was equipped with a Sick LMS laser scanner. The height of the scanner was 80 cm. Throughout this experiment, the unmanned aerial vehicle (UAV) kept a height of 50 ± 10 cm and the particle filter algorithm employed 5000 particles. Given this number of particles, our current implementation requires 5 ms per iteration on a Dual-Core 2 GHz laptop, while scan matching requires 5 ms on average. Fig. 5 shows three snapshots of the localization process at three different points in time. The top image depicts the initial situation, in which the particles were sampled uniformly over the free space. After approximately 1 m of flight (middle image), the particles start to focus around the true pose of the vehicle. After approximately 5 m of flight, the quadrotor was globally localized (bottom image). The blue circle indicates the current estimate of the filter.

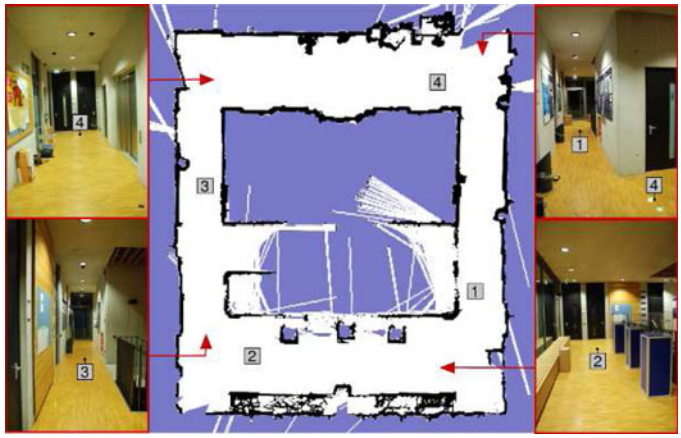


Fig. 6. Map of on office building built with our approach using the quadrotor. The labels 1–4 reflect the locations of individual markers that are used to evaluate the accuracy of our mapping approach. Red triangles indicate the pose of the corresponding camera images. The clutter in the bottom of the map originates from the seating containing horizontal slots (see bottom-right image).

B. SLAM

We also evaluated the mapping system by letting the quadrotor fly four loops (approximately 41 m each) in a rectangle-shaped building of approximate corridor size 10×12 m. The result of our SLAM algorithm is shown in Fig. 6. To quantitatively evaluate the accuracy of our mapping system, we placed markers on the floor (labeled $1, \ldots, 4$) and manually landed the quadrotor close to the markers. Since we never perfectly landed on those, we manually moved the quadrotor the remaining centimeters to match the predefined spots. This enables us to measure three types of errors: the re-localization error, the absolute positioning error, and the error in open-loop. The re-localization error is the difference between the current estimate and the estimate of the same real world pose in the previous loop. The error in open-loop is the re-localization error without enabling graph optimization. The absolute error is the difference between the estimated pose and the ground truth. To measure the absolute error we manually measured the relative locations of the markers and compared it with the positions estimated by the robot when landing to the corresponding markers. Table II shows the manually measured and the estimated poses of the markers for all loops. As can be seen, both the relative error between the individual loops and the global pose estimation with respect to the manually measured ground-truth have a maximum error of 1 cm. In this experiment, the incremental mapping during the first loop was accurate enough (<1 cm error); thus, no optimization was needed since all subsequent loops were also re-localized in the existing map. We therefore also evaluated each loop independently of each other without

TABLE II
ESTIMATED AND MANUALLY MEASURED LOCATIONS OF THE MARKERS FOR THE FLIGHT CONTAINING FOUR LOOPS IN TOTAL

marker	loop 1	loop 2	loop 3	loop 4	ground-truth
x_1	1.11 m	1.11 m	1.11 m	1.10 m	1.11 m
y_1	-7.50 m	-7.51 m	-7.50 m	-7.50 m	-7.50 m
x_2	-6.21 m	-6.21 m	-6.21 m	-6.21 m	-6.21 m
y_2	-9.21 m	-9.21 m	-9.21 m	-9.21 m	-9.21 m
x_3	-7.85 m	-7.85 m	-7.85 m	-7.85 m	-7.85 m
y_3	-3.83 m	-3.83 m	-3.83 m	-3.82 m	-3.82 m
x_4	-0.01 m	-0.01 m	-0.01 m	-0.01 m	0.00 m
y_4	-0.00 m	-0.00 m	-0.00 m	-0.00 m	0.00 m

TABLE III
COMPARISON OF SINGLE LOOPS FOR DIFFERENT GRID RESOLUTIONS

marker	loop 1	loop 2	loop 3	loop 4	finest resolution
x_4	-0.01 m	-0.35 m	-0.08 m	-0.17 m	0.01 m
y_4	-0.00 m	0.12 m	-0.07 m	0.04 m	
x_4	-0.42 m	-0.59 m	-0.36 m	-0.64 m	0.02 m
y_4	0.20 m	0.23 m	0.11 m	0.33 m	
x_4	-0.91 m	-0.59 m	-0.54 m	-0.60 m	0.04 m
y_4	0.28 m	0.38 m	0.29 m	0.29 m	

enabling graph optimization. The results of the individual loop flights for marker 4 (origin) are shown in Table III (first row). The worst flight (second loop) resulted in an error of approximately 0.37 m total distance to the origin. The remaining rows in Table III show the effect of using different grid resolutions at the finest level of our hierarchical mapping approach on the accuracy of the individual loops.

C. Multi-Level SLAM and Altitude Estimation

In the following, we show the typical behavior of our altitude estimation module. In this experiment, we let the robot fly autonomously in a typical office containing chairs, tables, and lots of clutter. The chairs have a height of 48 cm and the tables are arranged next to each other having a height of 77 cm. During this mission, the system flew once over the chair and several times over the tables where it also flew in a loop. Fig. 7 shows a snapshot of our multilevel mapping system during this mission. As can be seen from this figure, our algorithm correctly detected the objects at corresponding levels. The estimated heights of the chair and the tables were 48.6 $\pm$2.7 cm and 74.9 $\pm$2.8 cm, respectively.

D. Pose control

Since the system is stabilized by independent controllers, we discuss the result of each individual controller.

1) Yaw control: To test the yaw controller, we set a desired yaw of 0°, and once in a while, we turned the helicopter via the remote control. When the user released the remote control, the vehicle always returned back to its desired yaw with an error of $\pm 2°$. Fig. 8(a) plots the outcome of a typical run for yaw stabilization.

2) Altitude control: Similar to the experiment regarding the yaw, we ran an experiment to assess the behavior of the altitude control. In this test we set the desired altitude to 150 cm. In the beginning the vehicle was hovering over the ground. After enabling the stabilization the vehicle started climbing to the desired altitude. The desired height was kept by the vehicle up to an error of $\pm$10 cm. The results are shown in Fig. 7. Note that this experiment was performed while flying over different elevations.

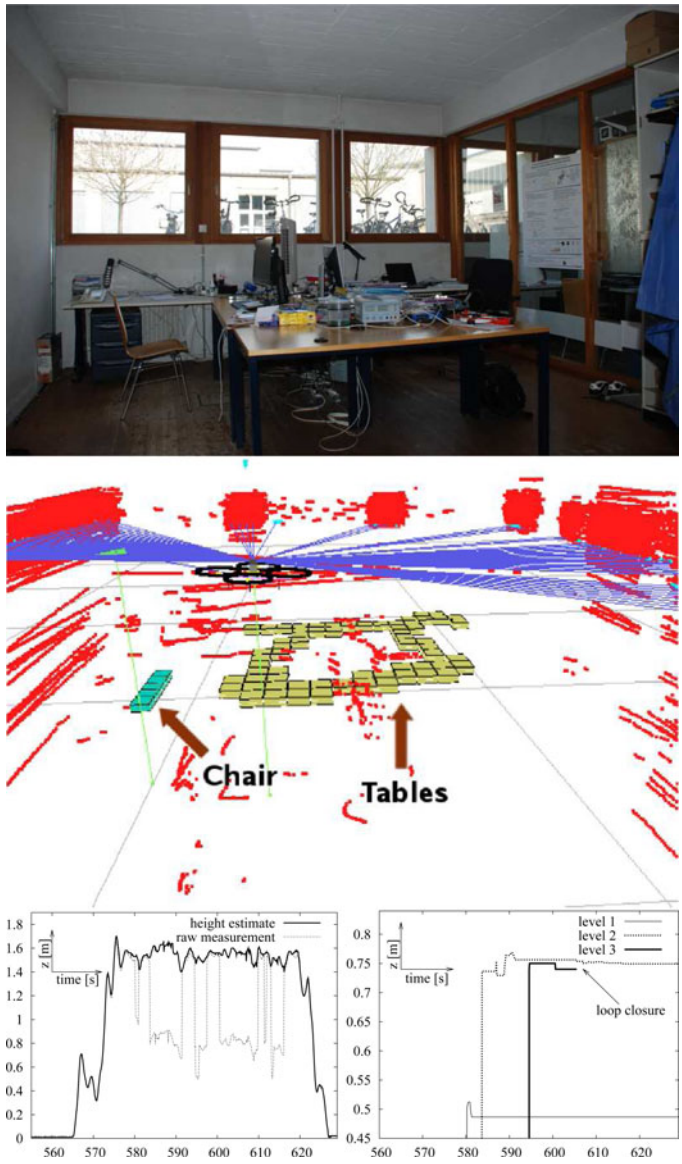


Fig. 7. Estimation of the global height of the vehicle and the underneath floor level. Whenever the quadrotor moves over a new level, the innovation is used to determine a level transition. The estimate of the height of each level is refined whenever the robot reenters that particular level. (Top) Office environment. (Middle) Corresponding map after autonomously flying over the vertical objects with a desired altitude of 150 cm. (Bottom left) Plot showing the estimated altitude of the vehicle over time versus the raw measurement. The corresponding estimated levels are depicted in the bottom-right plot. Note that Level 3 is merged with Level 2 after the loop closure.

3) x, y control: Finally we show an experiment for the pose stabilization only. Note, that the pose stability is strongly affected by the latency of the system (i.e., the time needed to calculate the command given the laser data). Although incremental motion estimates take only 5 ms in average (with a maximum of 15 ms), we have to deal with a latency of 120 ms in average because of the wireless transmission and because of the sensor

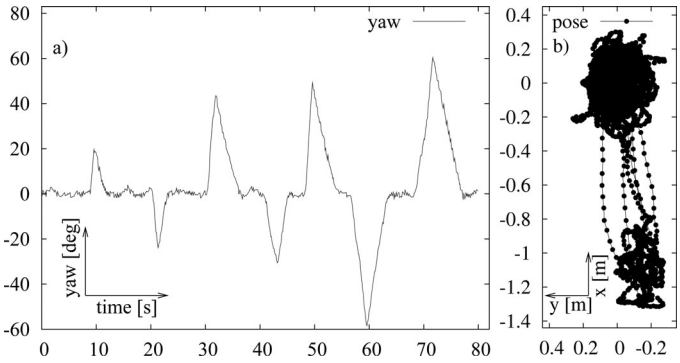


Fig. 8. Experiments for the autonomous stabilization of yaw (a) and pose (b). During the yaw stabilization experiment, the quadrotor was required to rotate to 0°, while the user manually turned the robot once in a while to a random orientation. Within the pose stability experiment, the quadrotor was set to hover at $(0, 0)$ but was manually moved backwards once in a while and required to fly autonomously back to the initial pose.

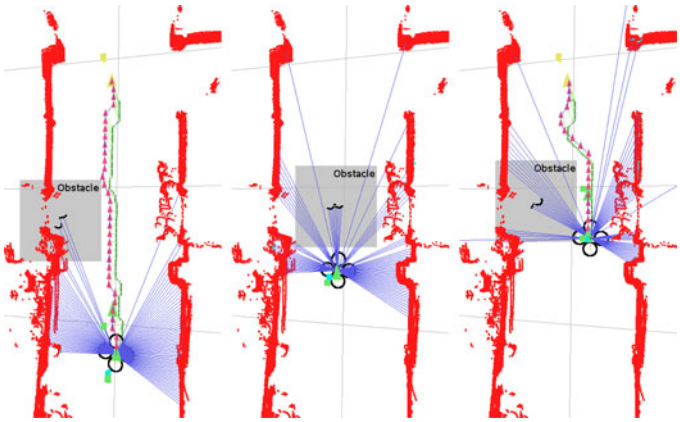


Fig. 9. Experiment for path planning and dynamic obstacle avoidance. The quadrotor is given a goal point 5 m in front of it. The planned path is shown in the left image. A person enters the corridor (shaded area) and blocks the robot's path, which results in an invalid plan. The quadrotor therefore hovers around the last valid way point (second image). In the third image, the person moved back, leaving the quadrotor enough space for a de-tour.

buffer. A typical run including autonomous pose stabilization is shown in Fig. 8(b). Here, the quadrotor was set to keep the initial pose of $(0, 0)$, and once in a while, the user used the remote control to move the quadrotor around 1 m backwards. The quadrotor then autonomously moved back to the desired position. Depending on the latency in the system, the pose oscillations are typically around ± 20 cm.

E. Path Planning and Obstacle Avoidance

In this section, we present an experiment that demonstrates our algorithms for path planning and dynamic obstacle avoidance. The quadrotor was given a goal point approximately 5 m in front of it. A person was standing on the left (see the shaded area in Fig. 9) enter the corridor when the quadrotor moved to its desired goal. The second image shows the situation when the person is completely blocking the robot's path. In this case, the quadrotor hovered around the last valid way point since there was no longer a valid plan to the goal. When the person moved to the left again, the quadrotor was able to follow a de-tour as shown in the right image of Fig. 9. Note, that the snapshots show the endpoints of the laser only. Although it looks like the quadrotor might have the space to fly around the person in the second image, there is no valid plan because of the safety margins around the walls.

VI. Conclusion and Future Work

In this paper, a navigation system for autonomous indoor flying utilizing an open-hardware quadrotor platform has been presented. A complete navigation solution that approaches different aspects of localization, mapping, path-planning, height estimation, and control has been described. Since we do not rely on special characteristics of the flying platform like the dynamics model, we believe that our system can easily be adapted to different flying vehicles. All modules in our system run online. However, because of the relatively high computational cost of some algorithms only a part of the software runs on-board on the ARM processor, whereas the other part runs off-board on a laptop computer. Some preliminary tests make us confident that the whole system can run on-board by the usage of the next generation of embedded computers that are based on the Intel Atom processor. We provided a wide range of experiments and some videos that highlight the effectiveness of our system. In future work, we plan to add a time of flight camera into our system. It has been believed that this technology can be effectively integrated and will allow us to relax the assumption that the vehicle moves over a piecewise planar surface.

References

[1] Carmen. (Oct. 8, 2011). [Online]. Available: http://carmen.sourceforge.net/
[2] ROS—Robot Open Source. (Oct. 8, 2011). [Online]. Available: http://www.ros.org
[3] Mikrokopter. (Oct. 8, 2011). [Online]. Available: http://www.mikrokopter.de/
[4] OpenSLAM—Open Source Navigation Software Repository, (Oct. 8, 2011). [Online]. Available: http://www.openslam.org
[5] (Oct. 8, 2011). [Online]. Available: http://ais.informatik.uni-freiburg.de/projects/quadrotor/
[6] M. Achtelika, A. Bachrach, R. He, S. Prentice, and N. Roy, "Autonomous navigation and exploration of a quadrotor helicopter in GPS-denied indoor environments," in *Proc. Robot.: Sci. Syst.*, 2008.
[7] S. Ahrens, D. Levine, G. Andrews, and J. P. How, "Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 2643–2648.
[8] E. Altug, J. P. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2002.
[9] A. Bachrach, R. He, and N. Roy, "Autonomous flight in unknown indoor environments," *Int. J. Micro Air Veh.*, vol. 1, no. 4, 2009.
[10] S. Bouabdallah, C. Bermes, S. Grzonka, C. Gimkiewicz, A. Brenzikofer, R. Hahn, D. Schafroth, G. Grisetti, W. Burgard, and R. Siegwart, "Towards palm-size autonomous helicopters," presented at the Int. Conf. Exhib. Unmanned Areal Veh., Dubai, UAE, 2010.
[11] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007.
[12] O. Bourquardez, R. Mahony, N. Guenard, F. Chaumette, T. Hamel, and L. Eck, "Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle," *IEEE Trans. Robot.*, vol. 25, no. 3, pp. 743–749, Jun. 2009.
[13] K. Celik, S. J. Chung, M. Clausman, and A. K. Somani, "Monocular vision SLAM for indoor aerial vehicles," in *Proc. IEEE Intell. Robots Syst.*, 2009, pp. 1566–1573.

[14] A. Coates, P. Abbeel, and A. Y. Ng, "Learning for control from multiple demonstrations," in *Proc. Int. Conf. Mach. Learning*, 2008.
[15] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," presented at the *IEEE Int. Conf. Robot. Autom.*, Leuven, Belgium, 1998.
[16] G. Grisetti, C. Stachniss, and W. Burgard, "Non-linear constraint network optimization for efficient map learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, pp. 428–439, Sep. 2009.
[17] S. Grzonka, G. Grisetti, and W. Burgard, "Towards a navigation system for autonomous indoor flying," in *Proc. IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, 2009.
[18] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a GPS-denied environment," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008.
[19] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin, "The Stanford testbed of autonomous rotorcraft for multiagent control," in *Proc. 23rd Digital Avion. Syst. Conf.*, 2004, p. 2.
[20] N. G. Johnson, "Vision-assisted control of a hovering air vehicle in an indoor setting," Master's thesis, Dept. Mech. Eng., Brigham Young Univ., Provo, UT, 2008.
[21] C. Kemp, "Visual control of a miniature quad-rotor helicopter," Ph.D. dissertation, Churchill College Univ., Cambridge, U.K., 2005.
[22] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, Jun. 2005.
[23] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991, pp. 295–356.
[24] E. Olson, "Real-time correlative scan matching," in *Proc. IEEE Int. Conf. Robot. Autom.*, Kobe, Japan, Jun. 2009, pp. 4387–4393.
[25] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a quad-rotor robot," in *Proc. Australian Conf. Robot. Autom.*, 2006.
[26] J. F. Roberts, T. Stirling, J. C. Zufferey, and D. Floreano, "Quadrotor using minimal sensing for autonomous indoor flight," in *Proc. Eur. Micro Air Veh. Conf. Flight Competition*, 2007.
[27] N. Roy, M. Montemerlo, and S. Thrun, "Perspectives on standardization in mobile robot programming," presented at the IEEE/RSJ Int. Conf. Intell. Robots Syst., Las Vegas, NV, 2003..
[28] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *Int. J. Robot. Res.*, vol. 27, no. 5, p. 549, 2008.
[29] A. Tayebi and S. McGilvray, "Attitude stabilization of a VTOL quadrotor aircraft," *IEEE TX Control Syst. Technol.*, vol. 14, no. 3, pp. 562–571, May 2006.
[30] T. Templeton, D. H. Shim, C. Geyer, and S. S. Sastry, "Autonomous vision-based landing and terrain mapping using an MPC-controlled unmanned rotorcraft," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2007, pp. 1349–1356.
[31] S. Thrun, W. Burgard, and D. Fox, "Robot perception," in *Probabilistic Robotics*, Cambridge, MA: MIT Press, 2005, pp. 171–172.
[32] S. Thrun, M. Diel, and D. Hahnel, "Scan alignment and 3-D surface modeling with a helicopter platform," *Field Serv. Robot. (STAR Springer Tracts Adv. Robot.)*, vol. 24, pp. 287–297, 2006.
[33] G. P. Tournier, M. Valenti, J. P. How, and E. Feron, "Estimation and control of a quadrotor vehicle using monocular vision and moire patterns," in *Proc. AIAA Guid., Navigat. Control Conf. Exhibit.*, 2006, pp. 21–24.

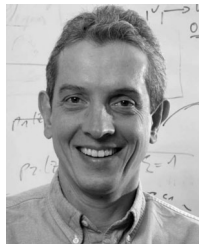
Slawomir Grzonka received the Master's degree in computer science from the University of Freiburg, Freiburg, Germany, in June 2006. He is currently working toward the Ph.D. degree from the Autonomous Intelligent Systems Lab, University of Freiburg.

His research interests include unmanned aerial vehicles, simultaneous localization and mapping, and human activity recognition.

Mr. Grzonka received best paper awards for his work on autonomous quadrotor systems from the International Conference on Robotics and Automation in 2009 and from the International Conference and Exhibition on Unmanned Areal Vehicles in 2010.

Giorgio Grisetti received the Ph.D. degree from the University of Rome La Sapienza, Rome, Italy, in April 2006.

He is an Assistant Professor with the Department of Systems and Computer Science, University of Rome, and a member of the Autonomous Intelligent Systems Lab, University of Freiburg, Freiburg, Germany. From 2006 to 2010, he was a Postdoctoral Researcher with the Autonomous Intelligent Systems Lab, University of Freiburg. His research interests include mobile robotic navigation, simultaneous localization and mapping, 3-D mapping, and path planning.



Wolfram Burgard received the Ph.D. degree in computer science from the University of Bonn, Bonn, Germany, in 1991.

He is a Full Professor of computer science with the University of Freiburg, Freiburg, Germany, where he heads the Laboratory for Autonomous Intelligent Systems. His research interests include artificial intelligence and mobile robots. In the past, with his group, he developed several innovative probabilistic techniques for robot navigation and control. They cover different aspects such as localization, map-building, path-planning, and exploration.

Dr. Burgard received several best paper awards from outstanding national and international conferences. In 2009, he received the Gottfried Wilhelm Leibniz Prize, which is the most prestigious German research award. Recently, he received an Advanced Grant from the European Research Council.