Towards a Navigation System for Autonomous Indoor Flying

Slawomir Grzonka Giorgio Grisetti Wolfram Burgard
{grzonka, grisetti, burgard}@informatik.uni-freiburg.de
Autonomous Systems Lab, Department of Computer Science
University of Freiburg, D-79110 Freiburg, Germany

***Abstract*— Recently there has been increasing research on the development of autonomous flying vehicles. Whereas most of the proposed approaches are suitable for outdoor operation, only a few techniques have been designed for indoor environments. In this paper we present a general system consisting of sensors and algorithms which enables a small sized flying vehicle to operate indoors. This is done by adapting techniques which have been successfully applied on ground robots. We released our system as open-source with the intention to provide the community with a new framework for building applications for indoor flying robots. We present a set of experiments to validate our system on an open source quadrotor.**

I. Introduction

In recent years, the research community has shown an increasing interest in autonomous aerial vehicles. 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 enables even unexperienced pilots to control these vehicles and allows them to be equipped with autonomous navigation abilities. Whereas most of the proposed approaches for autonomous flying [18], [8] focus on systems for outdoor operation, vehicles that can autonomously operate in indoor environments are envisioned to be useful for a variety of applications including surveillance and search and rescue. In such settings and compared to 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, i.e. without human interference. In the literature, this task is known as *navigation*. To address the general task of navigation one requires to tackle a set of problems ranging from state estimation to trajectory planning. Several effective systems for indoor and outdoor navigation of ground vehicles are nowadays available [1], [2]. However, we are not aware of a similar system for flying robots.

Whereas 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. First, due to their limited payload and size an indoor flying robot cannot carry the variety of sensors which can be easily mounted on a mobile robot. Second, the additional degrees of freedom of the vehicle prevents the direct use of well known and efficient 2D algorithms for navigation. Third the dynamics of a flying robot is substantially more complex than that of ground-based vehicles which makes them harder to control.

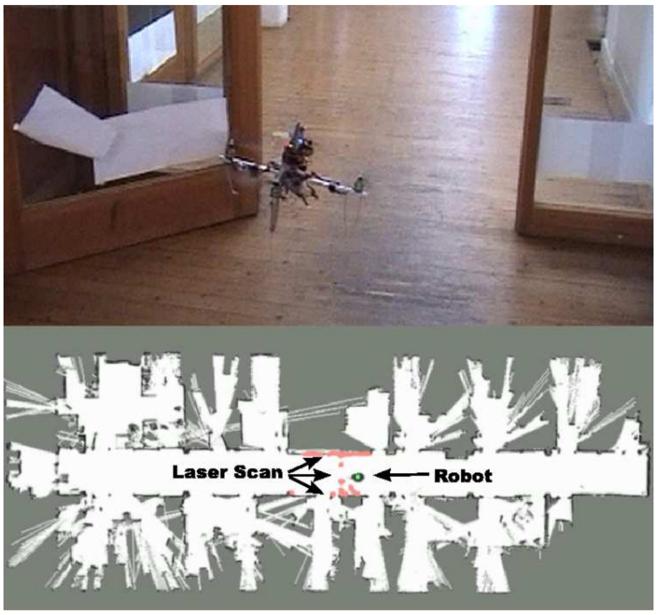

Fig. 1. A quadrotor robot equipped with our navigation system during a mission (top) and position of the vehicle estimated on-line during the flight with our navigation system (bottom).

In this paper, we describe a navigation system for indoor flying vehicles which are able to lift a payload of at least 300 grams and can supply an additional power of 7.5 watts. Our system is a result of an integrated hardware/software design which meets several of the challenging constraints imposed by small size flying vehicles while preserving a large degree of flexibility. We evaluated our system on an open source micro quadrotor, namely the Mikrokopter [3]. Figure 1 shows the Mikrokopter equipped with our navigation system during a mission. Special care has been taken to allow potential users to adapt the system to alternative platforms. An open source implementation is available at `www.openquadrotor.org`.

The remainder of this paper is organized as follows. In Section II we give an overview of the related literature. Subsequently, we discuss the requirements of a navigation system for an indoor flying vehicle (Section III) and we present our system in sections IV and V. We conclude with a set of experiments which illustrate the functionalities currently implemented in our navigation system in Section VI.

II. 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 [14], [16], [17], [4], with a particular emphasis on small helicopters.

Hoffmann *et al.* [13] present a model-based algorithm for autonomous flying with their STARMAC-quadrotor. Their system flies outdoors and utilizes GPS and IMU measurements. Bouabdallah *et al.* [6], [7] developed a complete model of their quadrotor platform and a set of different control strategies. Recently [5] they discussed the requirements of a flying platform for indoor navigation. Ng and colleagues [8] have developed algorithms for learning controllers for autonomous helicopter navigation. Their approach allows helicopters to perform impressive maneuvres in outdoor environments. Tournier *et al.* [19] used monocular vision to estimate and stabilize the current pose of a quadrotor. Thrun *et al.* [18] used a remotely controlled helicopter to learn large-scale outdoor 3D models.

There also has been some work that addressed the navigation of flying vehicles in indoor environments and in absence of GPS signal. Roberts *et al.* [15] used ultrasound sensors for controlling a flying vehicle in a structured testing environment, while He *et al.* [12] presented a system for navigating a small-size quadrotor without GPS. The pose of the vehicle is estimated by an unscented Kalman filter. Whenever the robot has to reach a given location, a path which ensures a good observation density is computed from a predefined map. These highly dense observations minimize the risk of localization failures.

In contrast to this approach, our system is suitable to be used on less structured environments which can be effectively represented by grid maps. We focus on adapting a set of algorithms which have been validated on ground robots to indoor flying platforms.

III. Indoor Navigation of an Autonomous Flying Vehicle

In this section, we first present the general problems in robot navigation and discuss the additional issues introduced by a flying platform. To autonomously reach a desired location, a mobile robot should be able to determine a collision free path connecting the starting and the goal locations. This task is known as *path planning*. To compute this path, a map of the environment should be known, which often also has to be acquired by the robot by processing the sensor measurements obtained during an exploration mission. This task 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. Finally, 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* which aims to move the vehicle along the trajectory, given the pose estimated by the localization given the measurements of the on-board sensors.

Several authors proposed effective control strategies to accurately steer ground vehicles with complex kinematics. Most of these approaches rely on high frequency estimates of the relative movements of the vehicle obtained by integrating the wheel encoders. The localization module does not need to run at high frequency due to the accuracy of the odometry within short time intervals. Unfortunately an odometry estimate is often not available on flying vehicles. In principle, one could obtain a dead reckoning estimate by integrating the inertial sensors. However, the limited payload typically requires designers to use only lightweight MEMS devices which are affected by a considerable drift. For these reasons, one needs frequent localization updates to implement effective control strategies.

In outdoor scenarios one can estimate the pose of the vehicle from the integration of GPS and inertial measurements. Unfortunately, indoors GPS is not available. Furthermore, the position accuracy obtained by a GPS would in general not be sufficient for navigating indoors. In these contexts, the robot is required to localize with the on-board sensors only. To detect and avoid obstacles, these sensors should reliably reveal the surrounding obstacles.

Due to the increased risk of damaging the flying platform during testing, the developer should have the possibility of intercepting at any point in time and take over the control of the platform.

Finally, the more complex dynamics of a flying platform poses substantially higher requirements on the accuracy of state estimates than for typical ground-based vehicles. As an example, on a helicopter an error in the pitch estimate of 2° would cause an error in the estimate of the translational thrust of approximately $\tan(2^\circ) \cdot 9.81 \approx 0.34 \frac{m}{s^2}$. Thus, such a relatively small error would force the helicopter to move by $68\,cm$ within two seconds. Whereas in outdoor scenarios such a positioning error can often be neglected, it is not acceptable indoors, as the free-space around the robot is much more confined.

In sum, a navigation system for an indoor flying vehicle should meet the following additional requirements: it should

- use sensors which can reliably detect obstacles in the neighborhood of the robot,
- estimate the pose over time with high accuracy and at high frequency,
- allow the user to take over control,
- provide a set of off-the-shelf basic behaviors, and
- use only lightweight on-board computers and sensors.

IV. Hardware Architecture

Figure 2 shows a Mikrokopter [3] open source quadrotor equipped with sensors and computational devices. Our system is similar to the one proposed by He *et al.* [12] and consists of the following components:

- a Linux-based Gumstix embedded PC with USB interfaces and a WiFi network card,

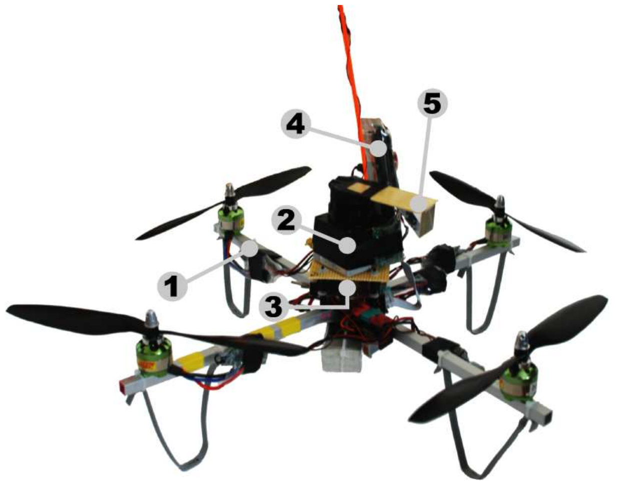

Fig. 2. The quadrotor platform used to evaluate the navigation system includes a Mikrokopter (1), Hokuyo laser range finder (2), an XSens IMU (3), a Gumstix computer (4), and a laser mirror (5).

- an Hokuyo-URG miniature laser sensor for localization and obstacle avoidance,
- an XSens MTi-G MEMS inertial magnetic unit (IMU) for estimating the attitude of the vehicle, and
- a mirror which is used to deflect some of the laser beams along the z direction to measure the distance from the ground.

The Gumstix communicates with the microcontroller on the quad-rotor via an RS-232 interface and reads all the sensors. Since the embedded PC runs Linux, we can develop our algorithms off-board on standard PCs and execute them on-board. We use the laser range finder for both measuring the distances to the obstacles in the surrounding of the robot and the distance from the ground. The IMU provides accurate estimates of the roll and the pitch of the vehicle, which are directly used for localization and mapping. All on-board sensing and computation devices together weight about 300 grams and drain approximately 7.5 watts of power.

V. Navigation system

In this section, we present the functionalities currently implemented in our navigation system. It is based on a modular architecture in which the different components communicate via the network using a publish-subscribe mechanism. At the current state, all the device drivers and some time-critical modules are executed on-board. The more computing-intensive algorithms for localization and mapping as well as the user interface are executed on a remote PC that communicates over wireless network with the platform.

The roll ϕ and pitch θ measured by the IMU are typically accurate up to 1°, which is sufficient for localization and mapping. In practice, we therefore calculate only four of the six components of the vehicle pose vector $\mathbf{x} = (x \; y \; z \; \phi \; \theta \; \psi)^T$, namely the 3D position $(x \; y \; z)^T$ and the yaw ψ.

The only sensor used for measuring the distances of nearby objects is the laser range finder. Based on known calibration parameters and on the attitude estimated by the IMU, we project the endpoints of the laser in the global frame. We address the problems of controlling and stabilizing the platform along different partitions of the state space separately. From the projected laser beams, we estimate the $x-y$ position and the yaw ψ of the vehicle in a 2D map. To compensate for the lack of odometry measurements we estimate the incremental movements by 2D scan matching. Finally, we control the altitude of the vehicle and estimate the height of the underlying surface by fusing the IMU accelerometers and the distance from the ground as measured by the laser.

In the remainder of this section, we first discuss the projection of the laser data and the estimation of the relative motion between subsequent laser scans. Subsequently, we present our localization, SLAM, and altitide estimation modules. We conclude by discussing the user interaction and the control algorithms.

A. *Projection of the Laser Data*

In this section, we explain how we project the laser data in the global frame of the helicopter, given a set of known calibration parameters. The laser range finder measures a set of distances b_i along the $x-y$ plane, in its own reference frame. Each of these distances can be represented by a homogeneous vector $\mathbf{b}_i$ in the 3D space $\mathbf{b}_i = (b_i \cos\alpha_i \;\; b_i \sin\alpha_i \;\; 0 \;\; 1)^T$, where α_i is the angle of the individual beams. Let $T_{\mathrm{IMU}}^{\mathrm{laser}}$ be the homogeneous transformation matrix from the IMU reference frame to the laser frame, known from a calibration procedure and let $T_{\mathrm{world}}^{\mathrm{IMU}}$ be the time dependent transformation from the world to the IMU. Note that $T_{\mathrm{world}}^{\mathrm{IMU}}$ is computed only from the estimated pitch and roll. We can compute the position of a laser endpoint $\mathbf{b}'_i$ which is *not* deflected by the mirror by the following equation:

$$\mathbf{b}'_i = T_{\mathrm{world}}^{\mathrm{IMU}} \cdot T_{\mathrm{IMU}}^{\mathrm{laser}} \cdot \mathbf{b}_i \tag{1}$$

Conversely, if a beam is deflected by the mirror, we obtain the point $\mathbf{h}'_i$ in the world frame by the following chain of transformations:

$$\mathbf{h}'_i = T_{\mathrm{world}}^{\mathrm{IMU}} \cdot T_{\mathrm{IMU}}^{\mathrm{mirror}} \cdot \mathbf{b}_i \tag{2}$$

Here, $T_{\mathrm{IMU}}^{\mathrm{mirror}}$ represents the transformation between the IMU and the *virtual* laser position which accounts for the effect of the mirror.

B. *Incremental Motion Estimation*

Some tasks, like pose stabilization, do not require to know the absolute location of the vehicle in the environment. Conversely, they rely on an accurate local pose estimate. We can estimate the relative movement of the robot between two subsequent scans by means of a scan matching procedure. Since the attitude is known from the IMU, this procedure can be carried on in 2D. In our implementation, we use an approach similar to [11]. This algorithm estimates the most likely pose of the vehicle $\hat{\mathbf{x}}_t$ given the previous pose $\mathbf{x}_{t-1}$, the current projected laser measurements $\mathbf{b}'_t$ and the previous one $\mathbf{b}'_{t-1}$, as follows

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}:=(x,y,\theta)}{\operatorname{argmax}} \; p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{b}'_{t-1}, \mathbf{b}'_t), \tag{3}$$

In our implementation we use a constant velocity model to compute the initial guess for the search.

C. Localization

We estimate the 2D position of the robot in a given grid-map by Monte-Carlo Localization [9]. The idea is to use a particle filter to track the positon of the robot. Whenever the robot travels over certain distance, we sample the next generation of particles based from a proposal distribution according to

$$\mathbf{x}_t^{[i]} \sim p(\mathbf{x}_t | \mathbf{x}_{t-1}^{[i]}, \mathbf{v}_t, \Delta\mathbf{x}) \tag{4}$$

where $\mathbf{x}_t^{[i]}$ is the generated sample, $\mathbf{x}_{t-1}^{[i]}$ is the previous sample, $\mathbf{v}_t$ are the velocities computed by integrating the IMU accelerations, and $\Delta\mathbf{x}$ is the relative movement estimated by the scan matcher. Subsequently, we sample a new set of particles proportional to likelihood

$$p(\mathbf{b}'_t | \mathbf{x}_t^{[i]}, \mathbf{m}) \tag{5}$$

of the measurement. Here $\mathbf{b}'_t$ is the current projected laser beam, $\mathbf{x}_t^{[i]}$ is the pose of the particle, and $\mathbf{m}$ is the known map. Note that whenever we use a scan for computing the odometry, the same scan is excluded from the evaluation of the likelihood. This prevents us from reusing the same information, which ultimately would result in overly confident estimates.

D. Simultaneous Localization and Mapping

Our mapping system addresses the SLAM problem by its graph based formulation. A node of the graph represents a 3DoF pose of the vehicle and an edge between two nodes models a spatial constraint between them. These spatial constraints arise either from overlapping observations or from odometry measurements. In our case the edges are labeled with the relative motion between two nodes which determine the best overlap between the scans acquired at the locations of the nodes.

To compute the spatial configuration of the nodes which best satisfy the constraints encoded in the edges of the graph, we use an online variant of a stochastic gradient optimization approach [10]. Performing this optimization on the fly allows us to reduce the uncertainty in the pose estimate of the robot whenever constraints between non-sequential nodes are added.

The graph is constructed as follows: Whenever a new $\mathbf{z_t}$ observation is incorporated into the system, we create a new node in the graph at the 2D position $\mathbf{x}_t = (x, y, \psi)$. We then create a new edge $\mathbf{e}_{t-1,t}$ between the current position $\mathbf{x}_t$ and the previous one $\mathbf{x}_{t-1}$. This edge is labeled with the transformation between the two poses $\mathbf{x}_t \ominus \mathbf{x}_{t-1}$. We determine the position of the current node with respect to the previous one by scan matching.

Whereas this procedure significantly improves the estimate of the trajectory, the error of the current robot pose tends to increase due to the accumulation of small residual errors. This effect becomes visible when the vehicle revisits already known regions. To solve this problem, we need to re-localize the robot in a region of the environment which has been visited long before. To resolve these errors, (i.e., to close the loop), we apply our scan matching technique on our current pose $\mathbf{x}_t$ and a former pose $\mathbf{x}_i$, where $i \ll t$. To detect a potential loop closure, we identify all former poses which are within the ellipsoid of the pose uncertainty obtained by a Dijkstra projection of the node covariances starting from the current robot position. If a match is found, we augment the graph by adding a new edge between $\mathbf{x}_i$ and $\mathbf{x}_t$ labeled with the relative transformation between the two poses computed by matching their corresponding observations.

E. Altitude Estimation

Estimating the altitude of the vehicle in an indoor environment means determining the global height w.r.t. a fixed ground h_g. Directly considering the z component of the beams $\mathbf{h}'_i$ deflected by the mirror would result in a correct estimate only when the vehicle moves on a single floor level. To relax this constraint, we simultaneously estimate the altitude of the vehicle and the altitude of the underlying surface with respect to an initial ground level. We assume the altitude of the floor to be piecewise constant and we utilize a Kalman filter for calculating the current altitude of the vehicle with respect to the current floor level. The state $\mathbf{s} = (z, v_z)$ used by the filter consists of the current height z and the corresponding velocity v_z along the z axis. The prediction of the filter is given by the following linear system

$$\hat{\mathbf{s}}_t = A\mathbf{s}_{t-1} + Ba_z, \tag{6}$$

with

$$A = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 0.5 \cdot \Delta t^2 \\ \Delta t \end{pmatrix}. \tag{7}$$

Here, a_z denotes the acceleration in z-direction measured by the IMU and Δt is the time elapsed between the current and the last iteration. If the current measurement falls into a confidence region of the prediction, we assume no change in the floor level. Otherwise, the gap between the current estimate and the measurement is assumed to be generated by a new floor level. This floor level is constantly re-estimated whenever the vehicle enters or leaves it.

The measurement update for the Kalman filter is given by:

$$\mathbf{s}_t = \hat{\mathbf{s}}_t + K \cdot (\hat{h} - C \cdot \hat{\mathbf{s}}_t), \tag{8}$$

with K being the Kalman gain, C describing the transformation from the state to the measurement, and $\hat{h} = \frac{1}{N_h}\sum_i z_i + h_g$. Here $N_h = |\{\mathbf{h}'_i\}|$ denotes the number of laser beams deflected by the mirror.

F. User Interaction

We control the flying vehicle by sending commands directly to the microcontroller which is in charge of the low level control of the platform.

For safety reasons, the user can always control the vehicle via a remote control (RC) and our system mixes the user and the program commands. During our experiments, we allow the programs to perturb the user commands by $\pm 20\%$. In this way, if one of the control modules fail the user still has the possibility of safely land the vehicle without loosing time of pressing a button first.

G. Control

The altitude is controlled by a PID controller which utilizes the current height estimate z and the velocity v_z respectively. The height control C_h can be summarized as

$$C_h = K_p \cdot (z - z^*) + K_i \cdot e_z + K_d \cdot v_z, \tag{9}$$

with K_p, K_i and K_d being the constants for the P, I, and D part respectively. Here z^* denotes the desired height and e_z denotes the integrated error.

The yaw is controlled by a proportional controller which computes the yaw command C_ψ as

$$C_\psi = K_p \cdot (\psi - \psi^*). \tag{10}$$

Here ψ and ψ^* are the measured and desired yaw.

VI. Experiments

In this section we present experiments for each of our modules described above, namely localization, SLAM, altitude and yaw control. During the experiments, altitude and yaw control were executed on-board, while scan matching, localization, SLAM, and altitude estimation were executed off-board on a standard laptop computer.

A. Localization

Using 2D grid maps for localization enables our system to operate with maps acquired by different kind 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 vehicle in a map acquired with a ground-based robot. This robot was equipped with a Sick Laser range scanner. The height of the scanner was $80\,cm$. Throughout this experiment, the UAV kept a height of $50\,cm \pm 10\,cm$ and the particle filter algorithm employed 5,000 particles. Given this number of particles, our current implementation requires $5\,ms$ on a Dual-Core 2 GHz laptop, while scan matching requires $30\,ms$ on average. Figure 3 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 separated 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 $5\,m$ the quadrotor was globally localized (bottom image). The blue circle highlights the current best estimate by the filter. A full video of a localization run is available on the Web under `www.slawomir.de/publications/grzonka09icra/localization_alufr.avi`.

B. Mapping

We also evaluated the mapping system by remotely steering the quadrotor along the corridor of our office environment. The result of our SLAM algorithm is depicted in Figure 4. The only mismatch between the map obtained by the quadrotor and the map generated from data gathered with ground-based robot consists in the very left part, where the pose estimation was inaccurate due to glass walls all around the robot. Despite this error, the map is sufficient for performing localization and we obtain similar results as with the map learned by the ground-based vehicle.

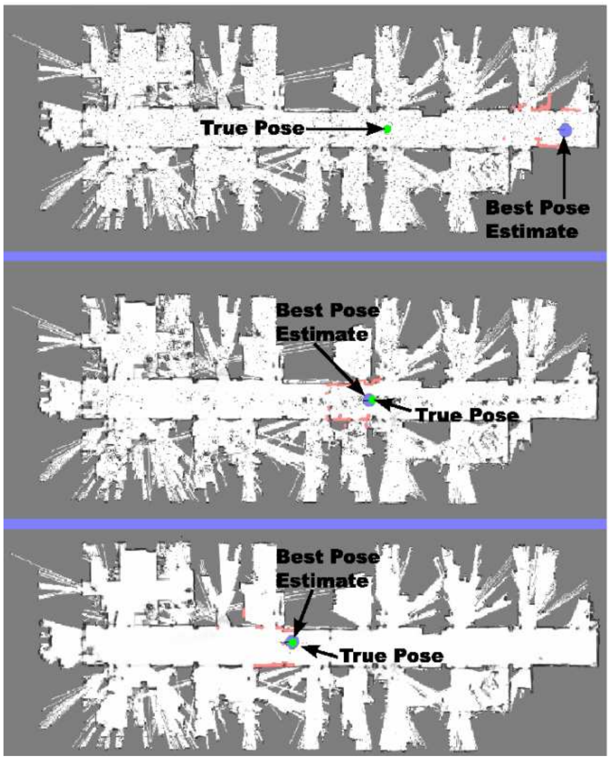

Fig. 3. Global localization of our quadrotor in a map, previously acquired by a ground-based platform. The blue circle highlights the current best estimate of the particle filter. The green circle marks the true pose of the vehicle. All potential robot poses are visualized as small black dots within the free (white) space of the environment. Top: initial situation. Middle: after about $1\,m$ of flight. Bottom: after approximately $5\,m$ of flight the quadrotor is localized.

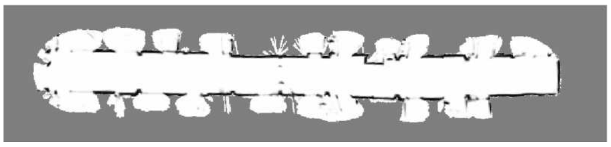

Fig. 4. Map of our office environment built with our approach and using the quadrotor. There is a small mismatch in the very left part if we compare this map with the one depicted in Figure 3. This mismatch originates from glass walls all around the robot which caused an error in the pose estimate. Still the map is sufficient to reliably localize the quadrotor.

C. Altitude estimation

In the following we present an experiment which validates our multi-level altitude estimation approach. We manually flew our vehicle in an environment with two different objects (a chair with a height of $46\,cm$ and a table with a height of $77cm$). During this flight the system flew four times over each of the objects. When flying backwards over the objects the vehicle passed them in the reverse order respectively. Figure 5 shows the estimate of the altitude and the floor level during one of the maneuvers. As can be seen from this figure, our algorithm correctly detected the objects at corresponding

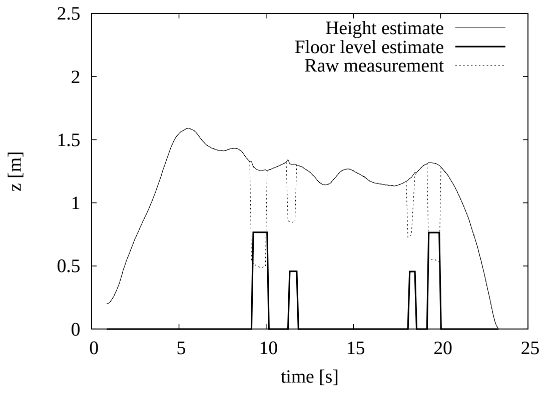

Fig. 5. Estimation of the global height of the vehicle and the underneath floor level. Whenever the quadrotor moves over a new level, the innovation is used to determine a level transition. The estimate of the height of each level is refined whenever the robot re-enters that particular level.

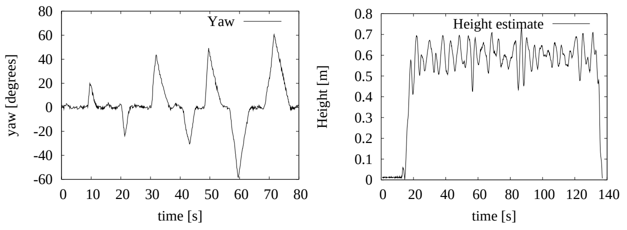

Fig. 6. Experiments for the autonomous stabilization of yaw (left) and height (right). During the yaw stabilization experiment, the quadrotor was required to rotate to $0°$. From time to time, the user manually changed the yaw . After the user released the remote control, the quadrotor autonomously rotated back to the desired yaw angle. During the height experiment (right) the quadrotor was required to maintain height of $60\,cm$. The resulting error in height was $\pm 10\,cm$.

levels. The estimated heights were $45.5\,cm \pm 2.1\,cm$ and $76.4\,cm \pm 2.4\,cm$. The vehicle first passes over the table and then over the chair.

D. Altitude and Yaw Control

In this final experiment, we show the capabilities of our yaw and altitude control modules. The yaw controller receives as input the yaw estimate coming from the scan matcher, while the altitude controller receives the feedback from the off-board altitude estimator. For testing 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 rc, the vehicle always returned back to its desired yaw with an error of $\pm 2°$. Figure 6 (left) plots the outcome of a typical run for yaw stabilization.

In a subsequent experiment, we tested the altitude stabilization. The designated altitude was $60\,cm$. 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 ± 10 cm. The results are shown in Figure 6 (right).

VII. Conclusions

In this paper, we proposed a navigation system for indoor flying vehicles. Our current system includes major relevant state estimation modules for localization, attitude and altitude estimation, and SLAM. We furthermore implemented a yaw and altitude control and an effective user interaction approach which allows to reduce the risk of collisions. Our system adapts a set of techniques which have been validated with ground robots, and it can also operate with data acquired by such platforms. We furthermore implemented some control strategies for yaw and altitude stabilization which can be further improved by incorporating a vehicle-specific model. Our aim is to close the gap between systems for wheeled robots and flying platforms. We want to provide a system which allows the type of robot to be transparent to the user. All modules described in this paper are made available as open source under `www.openquadrotor.org`.

VIII. Acknowledgments

This work has been supported by the EC under contract number FP6-IST-034120 Micro/Nano based Systems

References

[1] Carmen, http://carmen.sourceforge.net/.
[2] The Player Project, http://playerstage.sourceforge.net/.
[3] Mikrokopter, http://www.mikrokopter.de/.
[4] E. Altug, J.P. Ostrowski, and R. Mahony. Control of a quadrotor helicopter using visual feedback. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2002.
[5] S. Bouabdallah, M. Becker, and R. Siegwart. Autonomous Miniature Flying Robots: Coming Soon! *IEEE Robotics and Automation Magazine*, 13(3), September 2007.
[6] S. Bouabdallah, P. Murrieri, and R. Siegwart. Towards Autonomous Indoor Micro VTOL. *Autonomous Robots*, 18(2), March 2005.
[7] S. Bouabdallah and R. Siegwart. Full Control of a Quadrotor. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.
[8] A. Coates, P. Abbeel, and A.Y. Ng. Learning for Control from Multiple Demonstrations. *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.
[9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Leuven, Belgium, 1998.
[10] G. Grisetti, D. Lodi Rizzini, C. Stachniss, E. Olson, and W. Burgard. Online constraint network optimization for efficient maximum likelihood mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Pasadena, CA, USA, 2008.
[11] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, 2002.
[12] R. He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a GPS-denied environment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.
[13] G. Hoffmann, DG Rajnarayan, SL Waslander, D. Dostal, JS Jang, and CJ Tomlin. The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC). *The 23rd Digital Avionics Systems Conference (DASC).*, 2, 2004.
[14] P. Pounds, R. Mahony, and P. Corke. Modelling and Control of a Quad-Rotor Robot. *Proceedings of the Australasian Conference on Robotics and Automation (ACRA)*, 2006.
[15] J.F. Roberts, T. Stirling, J.C. Zufferey, and D. Floreano. Quadrotor Using Minimal Sensing For Autonomous Indoor Flight. *European Micro Air Vehicle Conference and Flight Competition (EMAV)*, 2007.
[16] A. Tayebi and S. McGilvray. Attitude stabilization of a four-rotor aerial robot. *43rd IEEE Conference on Decision and Control (CDC)*, 2, 2004.
[17] A. Tayebi and S. McGilvray. Attitude stabilization of a VTOL quadrotor aircraft. *Control Systems Technology, IEEE Transactions on*, 14(3):562–571, 2006.
[18] S. Thrun, M. Diel, and D. Hahnel. Scan Alignment and 3-D Surface Modeling with a Helicopter Platform. *Field and Service Robotics (STAR Springer Tracts in Advanced Robotics)*, 24:287–297, 2006.
[19] G.P. Tournier, M. Valenti, J.P. How, and E. Feron. Estimation and Control of a Quadrotor Vehicle Using Monocular Vision and Moire Patterns. *AIAA Guidance, Navigation and Control Conference and Exhibit*, pages 21–24, 2006.