Chapter 4
Computer Vision for Micro Air Vehicles

Roland Brockers, Martin Humenberger, Yoshi Kuwata, Larry Matthies and Stephan Weiss

Abstract Autonomous operation of small UAVs in cluttered environments requires three important foundations: fast and accurate knowledge about position in the world for control; obstacle detection and avoidance for safe flight; and all of this has to be executed in real-time onboard the vehicle. This is a challenge for micro air vehicles, since their limited payload demands small, lightweight, and low-power sensors and processing units, favoring vision-based solutions that run on small embedded computers equipped with smart phone-based processors. In the following chapter, we present the JPL autonomous navigation framework for micro air vehicles to address these challenges. Our approach enables power-up-and-go deployment in highly cluttered environments without GPS, using information from an IMU and a single downward-looking camera for pose estimation, and a forward-looking stereo camera system for disparity-based obstacle detection and avoidance. As an example of a high-level navigation task that builds on these autonomous capabilities, we introduce our approach for autonomous landing on elevated flat surfaces, such as rooftops, using only monocular vision inputs from the downward-looking camera.

4.1 Introduction

Miniature rotorcrafts are an ideal platform for exploration and reconnaissance missions, since they can operate in highly cluttered environmentsforest, close to the ground) or confined spaces (indoors, collapsed buildings, caves) and allow with

R. Brockers (✉) · Y. Kuwata · L. Matthies · S. Weiss
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, USA
e-mail: roland.brockers@jpl.nasa.gov

Y. Kuwata
e-mail: kuwata@alumni.mit.edu

L. Matthies
e-mail: lhm@jpl.nasa.gov

S. Weiss
e-mail: stephan.weiss@ieee.org

M. Humenberger
AIT Austrian Institute of Technology, Vienna, Austria
e-mail: martin.humenberger@ait.ac.at

B. Kisačanin and M. Gelautz (eds.), *Advances in Embedded Computer Vision*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-09387-1_4

their hovering ability to position a sensor payload in 3D space only constrained by the mission profile. However, in order to be deployable, a human–machine interface which allows an operator to easily control such a platform is key. The ingredient that most facilitates operation is autonomy, since autonomous vehicles can execute high-level commands without any further human interaction.

Thus, it requires the vehicle to know its position within the environment, and to have a capability to avoid collisions in flight and during takeoff and landing. All processes enabling such autonomy have to be implemented onboard, without requiring any external sensor input.

Miniature rotorcrafts (e.g., quadrotors) offer very high maneuverability and agility but require high rate of control because of their natural instability. Subsequently, sensor signals and images used for accurate pose estimation and for control input need to be processed fast. Since the platform has to be self-contained and payload capacities on micro air vehicles (MAVs) are in general very limited, only light-weight and low-power sensors and processing units can be used on-board the vehicle. This favors vision-based solutions that use small light-weight cameras and microelectromechanical systems (MEMS) inertial sensors. As recent developments in multicore smartphone processors are driven by the same size, weight, and power (SWaP) constraints, MAVs can directly benefit from new products that provide more computational resources at lower power budgets and low weight. This enables miniaturization of aerial platforms that are able to perform navigation tasks fully autonomously. In the subsequent sections, we introduce our autonomous navigation framework with focus on pose estimation, collision avoidance, and an example for a high-level navigation task that builds on these lower level functions: autonomous landing.

Viable solution for GPS-independent pose estimation from visual and inertial sensor inputs have been proposed in the literature [29, 41]. However, a major algorithmic challenge is to process sensor information at high rate to provide vehicle control and high-level tasks with real-time information about position and vehicle states.

In Sect. 4.2, we approach the issue of processing the vast camera information in real-time, rendering the camera a 6 degrees of freedom (DoF) pose sensor or a 3DOF velocity sensor. We discuss two methods representing two flavors of vision-based MAV state estimation. The first is a *map-based* approach using feature matches over long periods. The second is a *map free* and thus inherently fail-safe approach without using any kind of feature history. We will discuss that the first approach is more suitable for local drift-free navigation, while the latter is useful as a fall-back to keep the MAV airborne if a map corruption occurs. We will show that such an approach can quickly stabilize a thrown MAV and keep it at a constant heading and distance to the scene even though only two consecutive images and no feature history are used.

Once pose estimation is available, higher level autonomous navigation tasks which leverage and require this information can be executed. Examples for such tasks are: obstacle avoidance, autonomous landing, ingress, surveillance, exploration, and other.

In order to maneuver safely in highly cluttered environments and at low altitude, a MAV needs the ability to detect and avoid obstacles in its flight path autonomously.

Sophisticated solutions using active sensors (lidar, radar, etc.) exist for large aircraft, but they are in general unsuitable for small platforms with limited power, payload, and computational resources. To cope with these limitations, we developed a novel stereo vision-based obstacle avoidance approach, that is especially suited for onboard implementation on small aerial vehicles. Our approach is inspired by bird vision [36], using a forward-looking stereo camera system to provide depth information in the direction of flight, that can be expanded by range estimates from peripheral monocular optical flow. In Sect. 4.3, we explain our stereo vision-based obstacle avoidance system, that is designed for fast execution with small memory footprint by using: (1) a polar-perspective world representation in disparity space; (2) configuration space (C-space) expansion in image space; and implements (3) collision checking as a z-buffer like operation in disparity space. For motion planning, we use a closed-loop RRT approach that incorporates a vehicle model to plan local avoidance maneuvers in full 3D, which we believe to be scalable for flights at higher speeds.

As an example for a high-level navigation task, we explain autonomous landing with our MAV platform in Sect. 4.4. Autonomous landing is especially important not only for safety reasons, but also for mission endurance. Small rotorcrafts inherently suffer from overall short mission endurance, since payload restrictions do not allow carrying large batteries. For surveillance or exploration tasks, endurance can be greatly improved by not requiring the platform to be airborne at all time. Instead, such tasks may even favor a steady quiet observer at a strategic location (e.g., high vantage points like rooftops or on top of telephone poles)—still with the ability to move if required—which also could include recharging while in sleep mode (e.g., from solar cells).

4.1.1 Embedded Hardware Platforms

To evaluate the performance of our algorithms on an embedded system, we tested our framework with two different MAV platforms: an Asctec Pelican quadrotor equipped with an Asctec Mastermind flight computer (Core2Duo, 2×1.86 GHz CPU [4]) (total weight: $\sim$1.3 kg), and an Asctec Hummingbird quadrotor equipped with either an Odroid-X2 or a modified Odroid-U2 flight computer (total weight: $\sim$500 g; Fig. 4.1).

Both Asctec MAV platforms share the same low-level autopilot boards that include a MEMS IMU, and were equipped with a downward-looking Matrix Vision camera (mvBlueFOX-MLC200wG, CMOS, 752×480, grayscale, global shutter, up to 90 fps, 18.3 g with 100 FOV lens) that is connected to the flight computer.

The Odroid board (manufactured by Hardkernel [21]) is based on the Samsung Exynos 4412 system-on-a-chip (SoC)— a quadcore microcontroller for mobile applications that provides four ARM-cortex A9 for parallel computation, while only consuming 2.2 W (CPU only). For our implementation, we removed all non-necessary hardware components from the U2 in order to save weight, which included various connectors and the original heat sink. The final weight of the U2 flight computer was 12 g including the SD card which hosts the operation system.

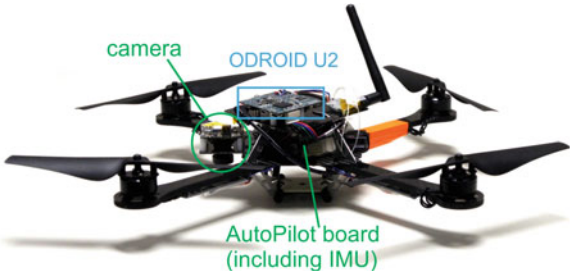

Fig. 4.1 Asctec Hummingbird with Odroid-U2 flight computer mounted on *top*

4.2 Pose Estimation

We start with explaining our pose estimation framework that is running onboard our quadrotors only using inputs from a camera and an IMU. We first present the related work followed by the detailed description of our two approaches and conclude with experimental results using the embedded hardware.

4.2.1 Related Work

Autonomous flights in unknown environments exclude the use of motion capture systems for MAV pose estimation as done for example in [39]. Furthermore, using GPS is not always reliable due to effects such as shadowing or multipath propagation in city-like environments. Therefore, commonly used sensors for GPS-independent MAV state estimation are stereo [38] and monocular cameras [63] as well as laser scanners [57]. Since heavy sensors cannot be used on low SWaP platforms and additional payload directly reduces endurance, monocular visual-inertial state estimators might be the most viable choice for MAVs.

Processing the vast information of the camera is a computationally complex task and cannot be processed at high rate. Multicopter MAVs require fast and precise control (and thus state estimates) at all times because the systems are inherently unstable. Hence, we propose to fuse the visual information with high-rate inertial cues from an IMU. We can categorize such a fusion into *loosely-coupled* and *tightly-coupled*. The loosely-coupled philosophy treats the inertial and visual units as two separate modules running at different rates and exchanging information, while the tightly-coupled paradigm combines both sources of information into a single, optimal filter. In general, loosely-coupled approaches are much less computationally expensive, since they use the low-dimensional processed visual information as measurement rather than every single feature. For this reason, we discuss a loosely-coupled Extended Kalman Filter (EKF) approach in this work. Among the loosely-coupled approaches are the works of [2, 3, 15, 19, 42, 51, 67], while among the tightly-coupled ones are those of [8, 13, 24, 27–29, 33, 34, 47, 58].

A filter-based approach not only allows to estimate the pose of the vehicle for control but also can estimate calibration parameters such as IMU biases, camera-IMU extrinsics, and visual drifts. Such *self-calibration* is crucial for long-term missions and renders the system *power-up-and-go* without the need of pre-mission calibration procedures. With a *map-free* inherently fail-safe vision module as we discuss below, it is further possible to eliminate the visual initialization procedure and failure modes. This literally renders the MAV *throw-and-go* as the MAV can be powered on and immediately be thrown in the air to deploy it.

4.2.2 Visual-Inertial State Estimation Approaches

A camera can be used in various ways to compute its pose in 3D space. We will discuss two approaches which can be classified into two main categories which we call *map-based* and *map free*. The first is a key frame-based visual odometry approach using a local map to estimate the arbitrarily scaled 6DoF of the camera. The second approach does not use temporal information or features (i.e., a local map) but only uses the current optical flow measurement to estimate the 3DoF arbitrarily scaled camera velocity vector, 3DoF attitude of the MAV, and distance to the current scene.

4.2.2.1 Map-Based Approach

The first approach is described in detail in [63, 67] which shows that, fusing with an IMU, we can navigate a MAV in large environments and high altitude with visual and inertial cues only. Because of robustness, real-time performance, and position accuracy, the keyframe-based solution proposed in [31] was selected and tailored to run on our embedded architecture. Our implementation uses a downward-looking camera and executes a sliding-window, vision-based self localization and mapping (VSLAM) feature tracking approach to extract pose estimates from visual inputs, maintaining constant computational complexity. This method is viable for large outdoor environments and long missions—only limited by the battery lifetime and not by processing power nor memory. We show how the proposed algorithm in [63] can be implemented on a 12 g, 5 W processing unit while still running at 50 Hz. This renders even a very light-weight MAV truly power-on-and-go.

The 6DOF pose of the VSLAM algorithm is fused with the inertial measurements of an IMU using an Extended Kalman Filter (EKF). More details are given in [63, 67]. An EKF framework consists of a prediction and an update step. The computational load required by these two steps is distributed among the different units of the MAV as described in [64]. The state of the filter is composed of the position $p_{\mathrm{w}}^{\mathrm{i}}$, the attitude quaternion $q_{\mathrm{w}}^{\mathrm{i}}$, and the velocity $v_{\mathrm{w}}^{\mathrm{i}}$of the IMU in the world frame. The gyroscope and accelerometer biases b_{ω} and b_{a} as well as the missing-scale factor λ are also included in the state vector. For completeness, the extrinsic calibration parameters describing the relative rotation $q_{\mathrm{i}}^{\mathrm{s}}$ and position $p_{\mathrm{i}}^{\mathrm{s}}$ between the IMU and the camera

frames were also added. This yields a 24-element state vector X:

$$X = \{p_{\mathrm{w}}^{\mathrm{i}\mathrm{T}}\, v_{\mathrm{w}}^{\mathrm{i}\mathrm{T}}\, q_{\mathrm{w}}^{\mathrm{i}\mathrm{T}}\, b_{\omega}^{\mathrm{T}}\, b_{\mathrm{a}}^{\mathrm{T}}\, \lambda\, p_{\mathrm{i}}^{\mathrm{s}}\, q_{\mathrm{i}}^{\mathrm{s}}\}. \tag{4.1}$$

Details about the EKF prediction and update equations can be found in [63]. A nonlinear observability analysis [62] reveals that all state variables are observable, including the intersensor calibration parameters $p_{\mathrm{i}}^{\mathrm{s}}$ and $q_{\mathrm{i}}^{\mathrm{s}}$. Note that the VSLAM pose estimates are prone to drift in position, attitude, and scale with respect to the world-fixed reference frame. Since these quantities become observable when fusing with an IMU (notably roll, pitch, and scale), gravity-aligned metric navigation becomes possible even in long-term missions. This is true as long as the robot excites the IMU accelerometer and gyroscopes sufficiently as discussed in [29]. Additionally, since the gravity vector measured by the IMU is always vertically aligned during hovering, the MAV will not crash due to gravity misalignment—even during long-term operations.

4.2.2.2 Map-Free Approach

The map-based approach described above is locally drift free. However, it requires to redetect the same features over several camera frames. This is prone to failure and mismatches, and can lead to corrupting the local map which in turn can lead to a crash of the MAV because of a wrong state estimate based on the corrupted map.

In [65, 66], we present an approach which only uses two consecutive camera images and inertial cues for MAV navigation. This inertial-optical flow (IOF)-based approach does not use any kind of history that can be corrupted and does not require to find the same features in later frames. In [66], we show that we still can estimate the metric velocity of the MAV, its metric distance to the scene, and its full attitude (roll, pitch, yaw) drift free while maintaining a self-calibrating system. That is, in addition to the states used for control, we can estimate the IMU biases and the camera-IMU extrinsics, and do not need specific calibration steps prior to launch. In fact, in this work, we show that this state estimation is robust and fast enough such that the MAV can be deployed by simply tossing it into the air rendering it a *throw-and-go* system. The state vector

$$\chi = \{p_{\mathrm{w}}^{\mathrm{i}} v_{\mathrm{w}}^{\mathrm{i}}\, q_{\mathrm{w}}^{\mathrm{i}}\, b_{\omega}\, b_{\mathrm{a}}\, \Lambda\, p_{\mathrm{i}}^{\mathrm{c}}\, q_{\mathrm{i}}^{\mathrm{c}}\, \alpha\} \tag{4.2}$$

contains the IMU-centered MAV position $p_{\mathrm{w}}^{\mathrm{i}}$, velocity $v_{\mathrm{w}}^{\mathrm{i}}$and attitude $q_{\mathrm{w}}^{\mathrm{i}}$with respect to the world frame. It also contains the IMU biases on gyroscopes b_{ω} and accelerometers b_{a}, the common visual scale factor Λ and the 6D transformation between the IMU and the camera in translation $p_{\mathrm{i}}^{\mathrm{c}}$ and rotation $q_{\mathrm{i}}^{\mathrm{c}}$. The system can additionally estimate the inclination α of the scene plane it currently observes (see Fig. 4.2).

A nonlinear observability analysis reveals that all states are observable except two dimensions in position. This is expected since optical flow and inertial measurements

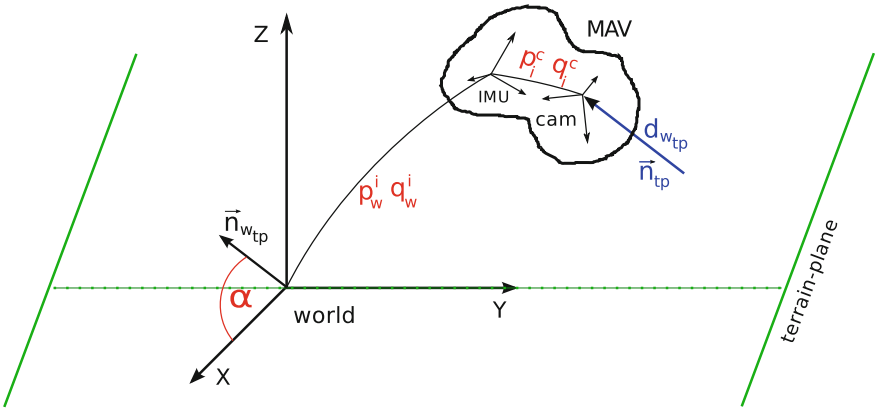

Fig. 4.2 Frame setup and state definition for the EKF framework. Without loss of generality, we can lock the gravity-aligned world y-axis along the terrain plane. The terrain plane normal vector can then be described as $n_{w_{tp}} = [\cos(\alpha)\ 0\ \sin(\alpha)]^T$ in the world frame. The *red* values are states estimated in the EKF framework, whereas the *blue* values are the scaled visual measurements normalized with our proposed approach aid of the terrain plane

only allow to estimate the distance to the scene. If this scene is inclined with respect to gravity, the system additionally can estimate the vehicle's heading with respect to the scene. Motion in parallel to the scene plane, however, is unobservable. We could overcome this issue and implement position hold by a place recognition or feature-tracking method. However, this would include a feature history and would defeat the purpose of a fail-safe algorithm.

Without having any type of history, an erroneous measurement will get rejected in the EKF update but would not corrupt a map, i.e., the next measurement will be independent of the previous one which is in line with the EKF assumptions. Such an inherently fail-safe approach allows the MAV to move very quickly and in an agile way since erroneous measurements get rejected and simply the next good measurement is taken into the filter process.

4.2.3 Embedded Implementation

While the previous sections described our algorithms, this section focuses on their implementation, parameter selection, and design choices on the embedded computing architecture on a MAV. To evaluate performance differences and the influence of weight reduction, we implemented our algorithms on the two different MAV platforms mentioned in Sect. 4.1.1, an Asctec Pelican quadrotor equipped with an Asctec Mastermind flight computer and an Asctec Hummingbird quadrotor equipped with an Odroid-U2 flight computer (Table 4.1). This renders even a very lightweight MAV truly throw-and-go.

Table 4.1 SWaP performance of tested computing platforms

Platform	Size (footprint) (mm)	Weight (g)	Power consumption (W)	Cores	Vision front-end frame rate (Hz)	CPU load (%)	Workload (%)
Asctec mastermind	144 × 135	300	30	2	30	59	30
Odroid-X2 dev. board (4412) including heat sink	90 × 94	122	8	4	30	125	31
Odroid-U2 (4412) stripped down version	48 × 52	12	5	4	30	125	31

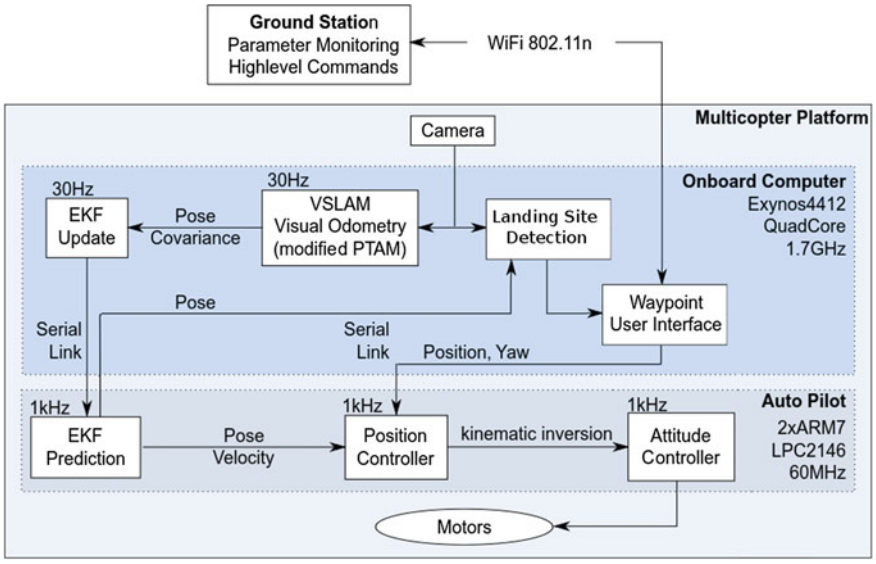

Fig. 4.3 System overview of the vision-aided pose estimation frame work

4.2.3.1 Map-Based Approach

Figure 4.3 gives an overview of the distributed implementation of our approach on the vehicle. All computational-expensive components are executed on the high-level flight computer, which includes VSLAM and pose filter update (EKF-update) as well as landing site detection. The EKF-update is passed down to the prediction loop that is executed on the autopilot board for efficiency reasons: the prediction loop, which includes IMU integration, and the position controller that uses the estimated pose to control the vehicle, both run at 1 kHz on a dedicated ARM7 microcontroller.

We ported the initial estimator implementation from the Asctec Mastermind to the U2 by using system specific changes in order to speed up the execution on the SoC. We used a highly ARM-customized Ubuntu version as operating system and Robot Operating System (ROS) [49] for interprocess communication. Our VSLAM implementation primarily consist of a *tracking* and a *mapping* part which we enforce to be executed on separate cores. *Tracking* is the most critical part, since it yields instantaneous pose measurements which are used to generate filter updates. Therefore, running this part on a dedicated core ensures uninterrupted pose handling at all time. *Mapping* is responsible for pose refinement and windowed bundle adjustment, and is thus less time critical. Note that the adjustments are refinements and we do not use global loop closure techniques. This avoids large and abrupt pose changes. Since the mapping task runs at a lower frequency and is less time critical, it shares its dedicated core with other system tasks. After optimization, the vision front end produced visual pose estimates at a stable 50 Hz rate.

4.2.3.2 Map-Free Approach

Our inertial-optical flow (IOF)-based approach is designed to keep the MAV airborne at all times in a fail-safe manner. Thus, it has to have low-computational cost requiring low system resources and it has to be fail safe.

We implement IOF on our 12 g Odroid-U2 platform and use similar NEON optimization instructions than for the above explained map-based approach. We use the same (FAST) feature extraction method but simplified the matching process by not warping patches. Since we only use two consecutive images at high frame rate, the distortion is small and warping is not required.

Computing the normalized camera velocity vector requires normalizing all optical flow vectors with their scene depth. As detailed in [65], this normalization uses a computational complex SVD per feature i including the optical flow $\dot{\mathbf{x}}_i(t)$, the feature direction vector $\mathbf{x}_i(t)$, and the camera velocity direction vector $\mathbf{v}(t)$. The unknowns are the feature scale factor and scale factor change $\dot{\lambda}_i(t), \lambda_i(t)$ and velocity normalization factor η:

$$\dot{\lambda}_i(t)\mathbf{x}_i(t) + \lambda_i(t)\dot{\mathbf{x}}_i(t) = \eta\mathbf{v}(t). \tag{4.3}$$

which can be stacked to a feature×features matrix M containing optical flow and velocity vector measurements ($\mathbf{x}_i(t)$, $\dot{\mathbf{x}}_i(t)$, $\mathbf{v}(t)$) and λ is the solution vector containing a scale factor per feature ($\dot{\lambda}_i(t)$, $\lambda_i(t)$) and a scale factor η for the velocity vector. λ is a solution up to an arbitrary global scale (which will be estimated in the EKF using the IMU). Thus without loss of generality, we can set $\eta = 1$ and use the block sparsity of M to efficiently compute the SVD in a block-wise parallel fashion on the Odroid-U2. The optimized code runs at 50 Hz with an image resolution of 572×480 (WVGA) on the Odroid-U2 using only about 20 % of the overall computation capacity of system.

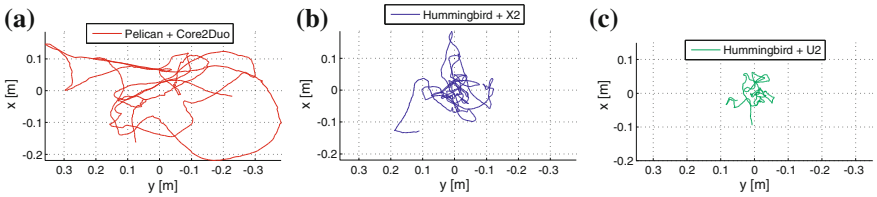

Fig. 4.4 Hover performance of the Pelican with Mastermind (**a**), the Hummingbird with a heavier Odroid-X2 evaluation board (**b**), the Hummingbird with modified U2 (**c**)

4.2.4 Experimental Evaluation: Map Based

To evaluate the influence of the reduced weight on the control stability of the platform, we executed a position hold maneuver with all three vehicle/flight computer configurations, where the MAV was controlled only with position estimates from our pose estimation software (the vision front end was again executed at a frame rate of 30 Hz).

Neglecting the influence of different flight performances of the two quadrotor systems, the reduced gross weight resulted in significantly better control performance: the hovering ellipse was reduced from ±35 cm for the heavy Asctec Pelican with Mastermind (RMS(x y z) = [8.3 cm 15.8 cm 1.5 cm]) to about ±15 cm for the Hummingbird with the X2 (RMS(x y z) = [5.4 cm 5.7 cm 1 cm]) and to ±7 cm for the Hummingbird with the final stripped down version of the U2 (RMS(x y z) = [2.9 cm 3.0 cm 0.8 cm]) (Fig. 4.4). Extensive tests in different environments were done in [63].

4.2.5 Experimental Evaluation: Map Free

We showed in [66] that we can control the MAV with IOF drift free in metric velocity, full attitude, and metric scene distance. Being able to keep the MAV constant in heading and scene distance is crucial for automatic initialization of more powerful algorithms (e.g., VSLAM) to control the vehicle in full 6DoF pose. Our IOF approach is sufficiently robust to estimate the vehicle pose even in drastic motion as it occurs when tossing the MAV in the air.

We start our IOF-based state estimation at $t = 38$ s and toss it at $t = 42$. The 4 s of "initialization" are sufficient to stabilize the MAV after the throw. After about 1 s the vehicle stabilizes already in attitude and in velocity. The convergence of the scene depth requires about 6 s longer. This is due to the wrong initialization of the metric scale factor which generally converges slower than the other states in the system.

Once all states are converged and the vehicle fully stabilized (after about 7 s in the test in Fig. 4.5), we have time to initialize a full VSLAM system as shown in [65].

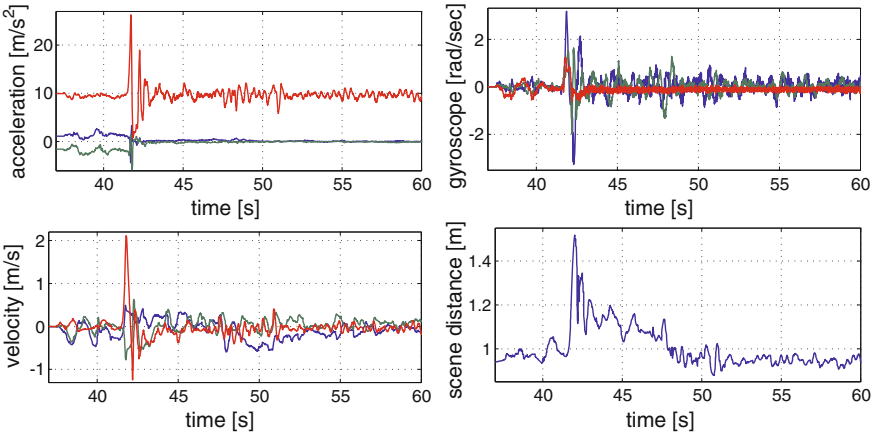

Fig. 4.5 Acceleration and velocity in x (*blue*), y (*green*) and z (*red*) when throwing the MAV in the air. At the throw, the MAV experiences an acceleration of 16.5 m/s^2 (*top left*). The angular velocities when throwing the MAV rise to 190 °/s (*top right*). And the velocity rises up to 2.3 m/s (*bottom left*). The scene depth is estimated correctly at all times (*bottom right*) and that the MAV can maintain it drift free after convergence. This allows good initialization of a subsequent, more powerful VSLAM algorithm

4.3 Obstacle Detection and Avoidance

The ability to sense obstacles and avoid collisions autonomously is a critical safety component for MAVs flying in cluttered environment and at low altitude (Fig. 4.6). Driven by payload and processing constraints and the requirement to accurately detect every obstacle within collision range, new algorithms have to be developed that reflect the limited capabilities of such small platforms. In the following section, we present our obstacle avoidance subsystem that uses stereovision for range perception, and a polar-perspective, inverse-range world representation (disparity space) for collision checking. The perception system is used by a closed-loop RRT motion planner, to navigate around detected obstacles, incorporating vehicle dynamics.

4.3.1 Related Work

Significant progress has been made recently on deliberative obstacle avoidance using active optical range sensors, such as single-axis scanning lidar and RGB-D structured light [5, 56]. The lack of a second axis for scanning lidar is a significant limitation and structured light sensors are ineffective in sunlight. Very compact, electronically beam-steered radar with large maximum range is under development for this application, but again has a single-axis scan [52]. Optical flow can cover a wide field of view and has been used for reactive obstacle avoidance [14, 25]; however, this

Fig. 4.6 Asctec Pelican quadrotor flying through forest

only provides information when the aircraft is moving, and poor estimation of time to collision near the focus of expansion limits the ability to perceive obstacles in the direction of motion. Stereo vision can provide 3D perception around the focus of expansion whether the aircraft is moving or not, but so far with relatively short look-ahead distance [18]. Stereo and optical flow have been used together in purely reactive obstacle avoidance based on sparse perception with point features [23].

The most common obstacle representations for MAV are image space data products that serve reactive obstacle avoidance [14, 25, 50] and Cartesian voxel data structures that serve deliberative planning [5, 18, 56]. Reactive obstacle avoidance with image space data structures use very little memory and computation, but have limited ability to reason about 3D structure and vehicle dynamics. Cartesian voxel data structures enable much greater 3D reasoning, have mature temporal fusion algorithms for error reduction, and have been used to plan high speed, aggressive maneuvers [48]; however, they use much more memory and computing time. Uniform voxel sizes are also problematic for representing both very near and very far objects, which can lead to more complex, multiresolution data structures. Polar representations parameterized by azimuth and range have been used in a few efforts because they naturally capture range-dependent variations in angular and range resolution [7, 68]; however, these efforts were only tested in simulation and [68] only represented sparse, discrete obstacles. Some stereo vision-based navigation systems for ground vehicles have had characteristics that are interesting for MAVs. A simple version of collision testing and path planning with the stereo disparity image was done in [44]. A 2D polar grid-based representation in the ground plane was used in [6], where the radial axis was parameterized as inverse range; this matched the angular and range resolution characteristics of stereo and gave a compact representation of all space from a minimum range to infinity. This was used to represent and reason about distant obstacles, while a 2D Cartesian map was used for nearby obstacles. Inverse range is equivalent to nearness fields that have been used for reactive MAV obstacle avoidance with optical flow [25].

Reactive obstacle avoidance controllers have been based on image space nearness fields computed from optical flow [25] and trained from human behavior via imitation learning [50]. Recent deliberative planners have used techniques including anytime, incremental A* for nonsymmetric vehicles moving slowly in cluttered spaces [35] and RRT* with path optimization [48] and lattice search with precomputed motion primitives [45] for fast, aggressive maneuvers.

4.3.2 Vision-Based Autonomous Navigation System

Traditional deliberative motion planning approaches usually implement a 3D grid-based world representation to expand trajectories and check for collisions [5, 18, 56], and more or less assume that a planned trajectory would be accurately followed by a relatively slow moving vehicle. Applying such an approach to a micro air vehicle generates several issues. Computational resources usually do not permit processing large 3D grid representations in reasonable time, and the agility of the system requires a complex planning approach which incorporates additional vehicle states to reflect fast vehicle dynamics. In our approach, we introduce two key features to mitigate these issues. To reduce complexity of path verification, our system uses an image-based world representation generated from stereo vision with a fixed memory footprint, and to allow planning in a low-dimensional planning space, trajectories are planned over closed-loop vehicle dynamics.

Figure 4.7 gives an overview of our system architecture. 3D perception follows a stereo vision pipeline. When images are acquired with a forward-looking stereo camera head, a stereo disparity map is calculated with a real-time stereo algorithm, and then expanded into configuration space (C-space), which is used for collision checking. Stereo, C-space expansion, and collision checking all take place within an image-based representation: 3D world points are characterized by their polar-perspective image coordinates in the frame of the reference camera and an assigned stereo disparity value (disparity space). The resulting 2.5D inverse-depth representation is very well suited for fast obstacle avoidance: close range where accurate object

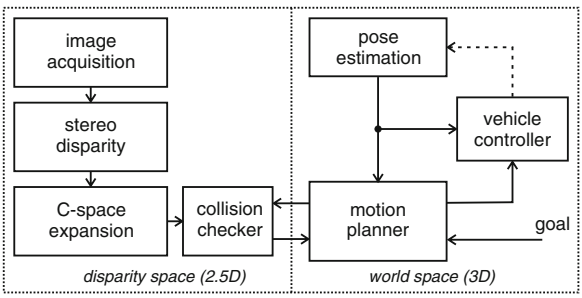

Fig. 4.7 Autonomous navigation system architecture

reconstruction is essential, is resolved with the highest accuracy, whereas accuracy for far distances decreases. At the same time, the polar-perspective character of an image-based representation significantly reduces the memory foot print of a world representation, making this method suitable for small hardware platforms.

The vehicle navigation system follows a standard control loop scheme: the motion planner module plans 3D vehicle trajectories in world space based on vehicle pose and a predefined goal input, and issues control commands to a vehicle controller, which maneuvers the vehicle. For collision checking, 3D trajectory segments are projected into disparity space and verified using the C-space map.

In our simulated experiments, we use simulated vehicle positions as pose estimation inputs. Our onboard implementation uses a vision-aided pose estimation approach [65] to provide pose. As any pose estimation framework on a real system will incorporate pose errors, we evaluate the robustness of our planning approach in Sect. 4.3.7.

In the following, we describe the individual parts of the approach in more detail.

4.3.3 Image-Based Collision Checking

For efficiency reasons, collision checking is performed directly in disparity space. When a new disparity image is obtained from stereo, C-space expansion is applied in the disparity domain, allowing to treat the MAV as a single point in space for planning purposes. During motion planning, small trajectory segments are verified by projecting them into disparity space and comparing the reconstructed disparity values along the segment with the corresponding C-space disparity values to detect collisions.

4.3.3.1 C-Space Expansion

C-space expansion is implemented as an image processing function (Fig. 4.8). To illustrate this operation, we first project a pixel of the stereo disparity map $p(u, v, d)$ into world coordinates using the stereo base b_s and the focal length f (in pixels), assuming rectified images and a disparity map that corresponds to the left camera view:

$$z_{\mathrm{w}} = -f b_{\mathrm{s}}/d \tag{4.4}$$

$$P(x_{\mathrm{w}}, y_{\mathrm{w}}, z_{\mathrm{w}}) = [uz_{\mathrm{w}}/f, vz_{\mathrm{w}}/f, z_{\mathrm{w}}]^T \tag{4.5}$$

Considering an expansion sphere S around $P(x_{\mathrm{w}}, y_{\mathrm{w}}, z_{\mathrm{w}})$ with the expansion radius r_{v}, we calculate the position of the rectangle that perfectly hides the sphere from the viewpoint of the camera (Fig. 4.9) and assign to it a disparity value that corresponds to the distance to the point on S that is closest to the camera origin.

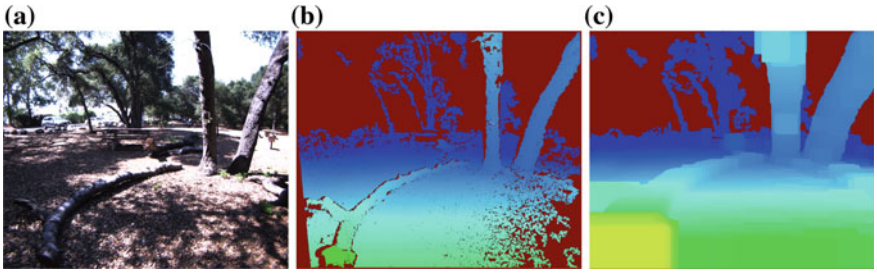

Fig. 4.8 C-space expansion in disparity space: original left view image (**a**), stereo disparity map (**b**), C-space expanded disparity map (**c**), pixels with warmer colors are located closer to the observer

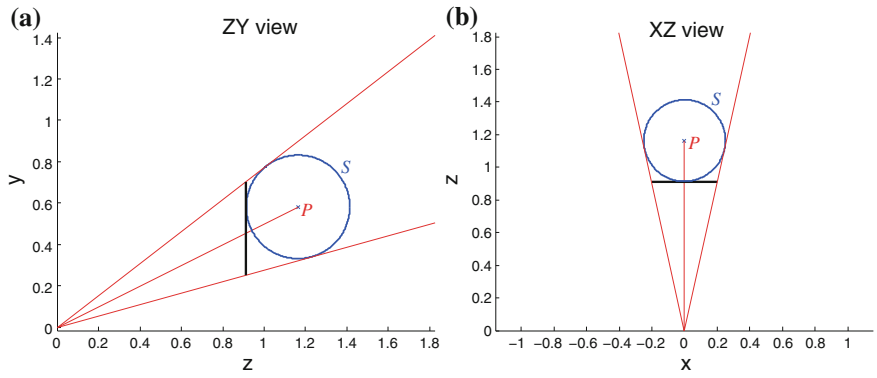

Fig. 4.9 C-space expansion example: The expansion *square* covers the expansion *sphere* from the camera view point (0, 0, 0) completely—its disparity is constant. **a** side view in ZY plane, **b** top down view in XZ plane

Technically, this expansion operation increases the expansion volume around a world point, since the correct projection of a sphere into the image is a circle, but this method has the advantage that the operation now is separable into two 1D operations—a horizontal expansion along image scan lines and a vertical expansion along image columns—reducing computational cost significantly.

The horizontal and vertical expansion limits depend on the viewing angle of each pixel and the expansion radius r_v. The horizontal viewing angle α to a point P in world coordinates is defined as

$$\alpha = \tan^{-1}(z_w/x_w) \tag{4.6}$$

and the horizontal angular field of view γ of the expansion sphere S around P is defined by the distance of P from the camera origin and the expansion radius r_v

$$\gamma = 2\alpha_1 = 2\sin^{-1}\left(r_v/\sqrt{z_w^2 + x_w^2}\right) \tag{4.7}$$

Projecting the two rays r_1 and r_2 along the viewing angles $\alpha + \alpha_1$ and $\alpha - \alpha_1$ back into the image defines the horizontal extension of the expansion sphere in the image

$$\begin{aligned} r_1 &= [r_{1_x}, r_{1_y}, r_{1_z}] = [z_\mathrm{w}/\tan(\alpha + \alpha_1), y_\mathrm{w}, z_\mathrm{w}] \\ u_1 &= f r_{1_x}/r_{1_z} \end{aligned} \tag{4.8}$$

$$\begin{aligned} r_2 &= [r_{2_x}, r_{2_y}, r_{2_z}] = [z_\mathrm{w}/\tan(\alpha - \alpha_1), y_\mathrm{w}, z_\mathrm{w}] \\ u_2 &= f r_{2_x}/r_{2_z} \end{aligned} \tag{4.9}$$

The vertical extension can be calculated similarly:

$$\beta = \tan^{-1}(z_\mathrm{w}/y_\mathrm{w}) \tag{4.10}$$

$$\beta_1 = \sin^{-1}\left(r_\mathrm{v}/\sqrt{z_\mathrm{w}^2 + y_\mathrm{w}^2}\right) \tag{4.11}$$

$$\begin{aligned} r_3 &= [r_{3_x}, r_{3_y}, r_{3_z}] = [x_\mathrm{w}, z_\mathrm{w}/\tan(\beta + \beta_1), z_\mathrm{w}] \\ v_1 &= f r_{3_y}/r_{3_z} \end{aligned} \tag{4.12}$$

$$\begin{aligned} r_4 &= [r_{4_x}, r_{4_y}, r_{4_z}) = (x_\mathrm{w}, z_\mathrm{w}/\tan(\beta - \beta_1), z_\mathrm{w}] \\ v_2 &= f r_{4_y}/r_{4_z} \end{aligned} \tag{4.13}$$

To determine the new disparity of the rectangular defined by the image coordinates u_1, u_2, v_1 and v_2 the expansion radius r_v is subtracted from the z-component of P and transformed into a disparity value:

$$z_\mathrm{new} = z_\mathrm{w} - r_\mathrm{v} \tag{4.14}$$

$$d_\mathrm{new} = -f b_\mathrm{s}/z_\mathrm{new} \tag{4.15}$$

To calculate the full C-space map, (4.6)–(4.15) are applied to every pixel in the stereo disparity image. Each (u, v, d) triplet defines a rectangular image region (u_1, v_1, u_2, v_2) with a constant disparity d_new, that is written into an output map. Individual pixels are only updated if the new disparity is larger than the previous disparity value that was generated by a different (u, v, d) triplet.

4.3.3.2 Implementation Aspects

Equations (4.6)–(4.15) can be precalculated over the disparity space volume. Since the calculation of u_1 and u_2 in (4.6)–(4.9) is independent of the y coordinate it suffice to precalculate a look-up table to store the values of u_1 and u_2 for each defined (x, d) combination. Similarly, a look-up table for v_1 and v_2 is calculated for each (y, d) combination, and finally, the values for d_new can be precalculated

for all valid disparities. Because of this separability, the C-space expansion can be implemented very efficiently. In a first step, an intermediate disparity image is generated by horizontally expanding all disparity values from the stereo disparity map using the u_1/u_2 look-up table. In a second step, each pixel in the intermediate disparity image is expanded vertically using the v_1/v_2 look-up table and the expansion column is stored with a new disparity value from the d_{new} look-up table in the final C-space disparity map.

4.3.4 Collision Checking in Disparity Space

The queries from the motion planner come as a sequence of short linear 3D trajectory segments. The collision checker module takes each segment that is defined through its start and end point, projects it into the current C-space disparity map D_{cm}, and checks all pixels that are located on the straight line between the projected start and end point for collision. Collision checking itself depends on the reconstructed disparity value $d(p_{\text{s}})$ of a point p_{s} on the segment and the actual disparity of the underlying pixel p_{cm} in the C-space map. If the disparity of p_{s} is larger than the disparity of p_{cm}, the pixel is classified as *safe*. If the disparity of p_{s} is smaller than p_{cm} the point on the trajectory is located behind an obstacle, and it is classified depending on the disparity difference

$$\begin{aligned} d(p_{\text{s}}) > d(p_{\text{cm}}) &: \text{SAFE} \\ d(p_{\text{s}}) < d(p_{\text{cm}}) \wedge d(p_{\text{s}}) - d(p_{\text{cm}}) < k &: \text{COLLISION} \\ d(p_{\text{s}}) < d(p_{\text{cm}}) \wedge d(p_{\text{s}}) - d(p_{\text{cm}}) \geq k &: \text{OCCLUDED} \\ p_{\text{s}} \notin D_{\text{cm}} &: \text{OUTSIDE} \\ d(p_{\text{cm}}) = \mathit{invalid} &: \text{NO\_DATA} \end{aligned} \tag{4.16}$$

If the difference is smaller than a threshold, a collision occurred. If it is larger, the trajectory point is labeled as *occluded* as shown in Fig. 4.10. If the C-space map contains no valid disparity data at a checked pixel location p_{cm}, the trajectory segment is labeled as *no_data*—which can be caused, e.g., by a nontextured surface when applying a real-time stereo approach.

How the planner uses these different trajectory classifications is explained in the following section.

4.3.5 Motion Planning over Closed-Loop Dynamics

Motion planning of aerial vehicles has several challenges. First, the state space is high dimensional—6DOF position and orientation, and their time derivatives (velocity/angular velocity, etc.), resulting in at least 12 states. Second, the system is very agile and consequently has poor stability, and naively propagating the open-loop vehi-

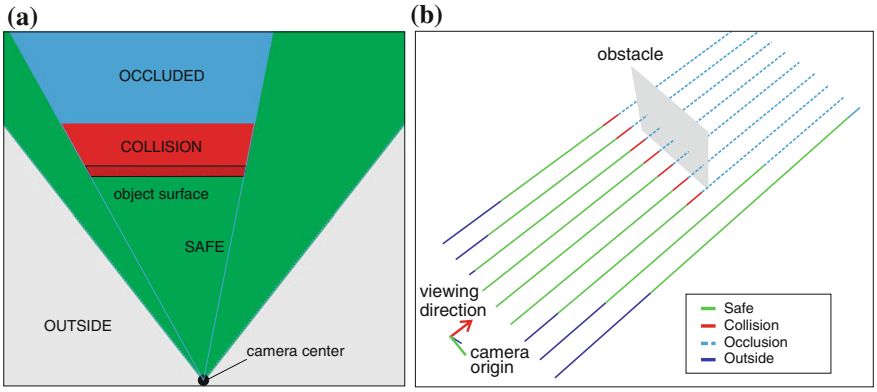

Fig. 4.10 Collision checking logic: **a** trajectory segments inside the viewing volume of the camera are classified as: SAFE if unobstructed (*green*), COLLISION if directly behind an object surface (*red*), and OCCLUDED if behind on object; **b** test trajectories in a simulation: collision checking result for a set of nine *horizontal* trajectories

cle dynamics quickly results in undesirable trajectories. Third, the dynamics become highly nonlinear especially when performing aggressive maneuvers. To address these challenges, we deploy a motion planning approach that incorporates vehicle dynamics by forward-simulating vehicle responses to waypoint control inputs, which effectively reduces the planning space to only 3D.

We extended an approach that was previously used for autonomous urban driving [31] to 6DOF motion with agile vehicle dynamics.

Our planner deploys a closed-loop RRT approach (CL-RRT) to grow a tree of waypoint inputs that are used in a feedback loop to estimate flight trajectories using a low-level controller and the quadrotor model described by How et al. [22]. The low-level controller consists of two layers: a linear *feedback controller* and a *waypoint tracker*. The *waypoint tracker* keeps track of which waypoint to visit next, and when to switch to the next waypoint segment (Fig. 4.11). It also computes the reference position/velocity and the position/velocity tracking errors. Given these tracking errors, the *feedback controller* computes the vehicle inputs $u_{\text{collective}}$, u_{roll}, u_{pitch}, and u_{yaw} to maneuver the vehicle along the trajectory using regular PID controllers for each channel.

The output of the control loop are the predicted vehicle states which define trajectory segments that are used for collision checking. Note, that the planner simultaneously grows a tree of controller inputs (straight lines connecting the controller input of a selected node to a sample, which forms an input to the forward simulation) and a tree of collision-free dynamically feasible trajectories (output of the forward simulation) as illustrated in Fig. 4.12.

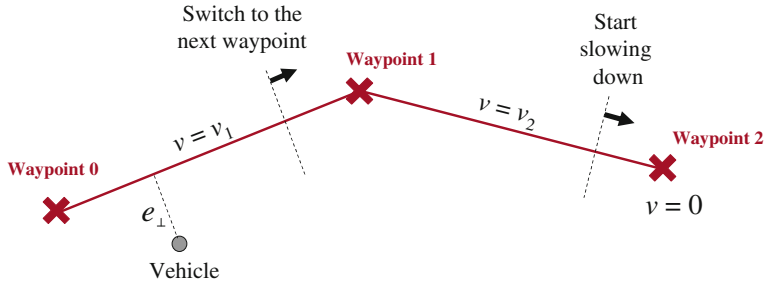

Fig. 4.11 Waypoint tracking logic. A cross-track controller minimizes the cross-track tracking error $e_\perp$, while an along-track controller maintains a commanded velocity along the lines that connect waypoints. The tracker switches waypoints when the along-track distance to the next waypoint becomes smaller than a threshold. Waypoints are marked with $\times$

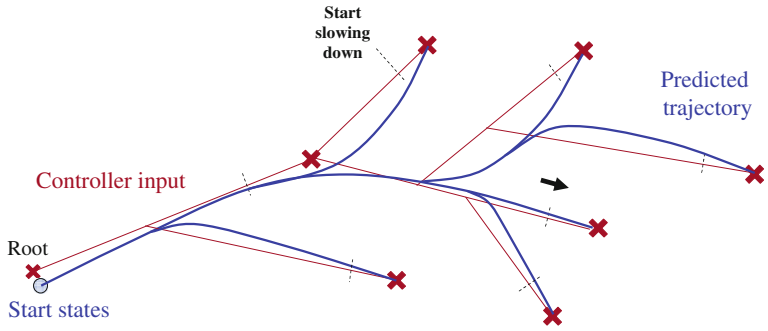

Fig. 4.12 RRT with closed-loop dynamics. The *red lines* represent the input to the controller, which are constructed by connecting a sample (marked with $\times$) to the tree. The *blue lines* represent the predicted vehicle trajectory, which is computed by the forward simulation

The main steps of the RRT algorithm are

1. Randomly sample the 3D world space.
2. Select a node in the tree.
3. Form a controller input from the selected node to the sample.
4. Forward simulate the closed-loop dynamics from the selected node, using the controller input generated in step 3, and obtain a dynamically feasible trajectory.
5. Check if the predicted trajectory collides with or is occluded by obstacles (Sect. 4.3.3). If there is a collision, or the occlusion is in a close range, discard the sample and go to step 1.
6. Add a sample and associated propagated trajectory to the tree. Also generate intermediate nodes on the trajectory, so that it can branch off to another trajectory for future samples.

4.3.6 Motion Planning Strategies

When the low-level controller executes the planned waypoints, different real-world considerations require specific execution strategies.

First, our planner is designed to maintain a safety invariance while the vehicle is flying [54] by adapting vehicle velocities such that the vehicle can come to a hover from the current state without collision. Note, that hovering is an invariant state of the quadrotor—once the vehicle is in a hover state, it can theoretically remain in hover without collision indefinitely in a static environment. Second, paths planned a certain distance behind a perceived obstacle are considered as feasible, which allows to plan into potentially free space behind obstacles. The maintained safety invariance ensures an early replanning should this space not be free. Space with no available data is treated similarly, since we expect that obstacles will have enough surface texture to be captured in the collision checking process.

Third, the predicted trajectory and the actual trajectory flown are generally close [32]. To account for nonzero prediction errors and changes in the environments, the planner repropagates the latest states and changes the trajectory accordingly to ensure collision-free flight from the current position.

Last, if the collision checker rejects a trajectory because it ends in occluded space or outside the field of view, we retain the feasible portion of such a trajectory in the tree, so that RRT can quickly connect future samples and grow trees from it.

4.3.7 Experimental Results

To evaluate our approach, we implemented our navigation algorithm both in a simulation environment (Fig. 4.13) and on a Asctec Pelican quadrotor system (Fig. 4.6). The simulation environment mimics a quadrotor within a virtual 3D world that is composed of cuboids.

At the beginning of a simulated flight, the quadrotor is placed at a starting position above the ground and the planner is executed a few seconds ahead of the controller to allow the construction of an initial tree of decent size. Figure 4.13 gives an example of such a step, where the stationary vehicle has planned trajectories toward the goal at the top of the scene. When the controller is started, the vehicle begins to execute the planned trajectory, replanning simultaneously during flight. As the position of the vehicle changes, new parts of the scene come into view, and trajectories are updated accordingly, until the goal is reached. To verify the robustness of our approach, we repeatedly executed a simulated flight for several example scenes. One standard scenario for performance evaluation of planning stability is the flight through a vertical opening in a wall (Figs. 4.14 and 4.15), which we use to measure the influence of a corrupted pose on the planning approach. In a Monte Carlo simulation, we commanded the vehicle to pass the vertical opening with noise added to the pose estimate and recorded the flown flight paths for 100 flights for each experiment.

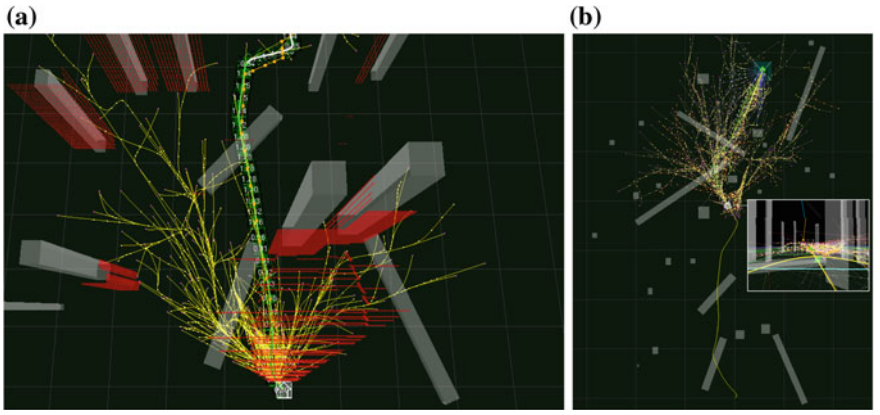

Fig. 4.13 Simulated flight through virtual forest: **a** Initial flight trajectory from the vehicle position at the *bottom* toward the goal at the *top* with overlaid 3D C-space point positions (*red*); the *horizontal red lines* in front of the vehicle correspond to C-space points above the ground; **b** *top-down* view during traverse with overlaid current view of the vehicle

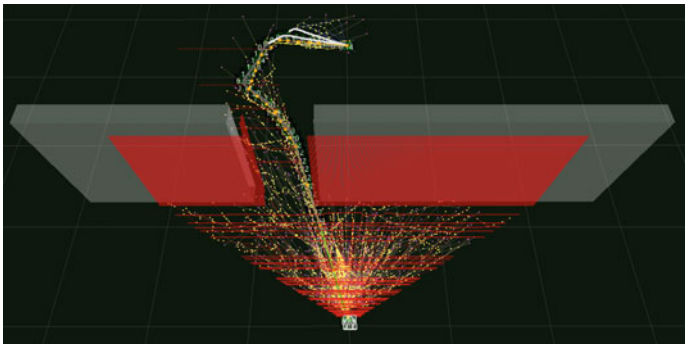

Fig. 4.14 Flying through a door opening with added pose noise: Scene view from the starting position with overlaid C-space point cloud (*red*)

With no pose noise added, the vehicle was able to fly safely through the opening on each run (Fig. 4.15a). Figure 4.15b illustrates the same flight experiment, where limited white noise ($\eta \in [-15\,\text{cm}, 15\,\text{cm}]$) was added to the position of the collision sensor (stereo camera system) prior to each planning cycle. When treating the pose of the collision sensor (with noise overlay) as true vehicle pose, all node positions on the RRT-tree become erroneous, simulating the effect of a noisy pose estimate on the planning process. In this case, collision checking invalidates additional tree branches, due to the added noise, forcing the vehicle to re-plan a valid trajectory when needed. As shown in Fig. 4.15b, additive noise affects the planning result only marginally. The vehicle was able to execute all runs properly and pass the opening at all times, since the added noise had zero mean (no drift).

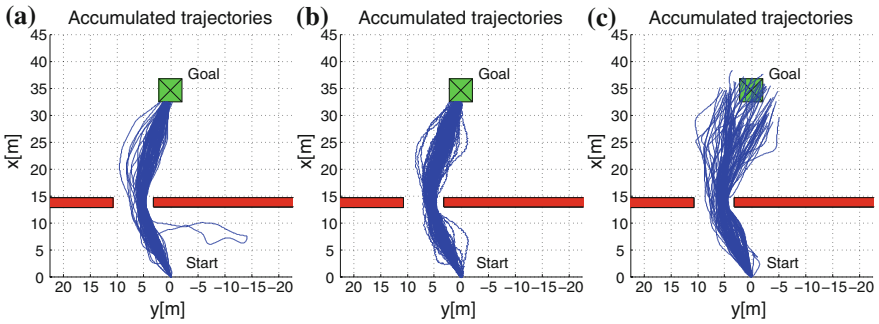

Fig. 4.15 Door experiment: *top-down* view of Monte Carlo simulation with 100 runs with **a** no pose noise, **b** additive white noise in x and y ($\leq$15 cm), **c** random walk bias in x and y (13.4 cm/s, random direction for each run)

This changed when a drifting pose estimate was simulated as shown in Fig. 4.15c. To simulate a random walk bias on the position estimates, a fixed position offset of 1.3 cm (a 10 % drift when operating at maximum speed of 1.3 m/s with a 100 Hz simulation rate) in a random direction within the x/y plane was defined at the beginning of each run, and added as a pose offset prior to each planning step. In this experiment, the vehicle kept a stable orientation to maintain correct heading, which we assume closely related to real flight experiments, since roll and pitch would be stabilized around a drift-free gravity vector and yaw can be assumed to be measured locally drift free by an onboard magnetometer or a vision-aided pose estimation approach. Since all world point coordinates drift with the amount of bias on pose, the goal cannot be reached and serves as a desired flight direction indicator.

The vehicle was able to pass the obstacle without a collision in 96 % of all cases. Collisions only occurred when the vehicle was already within the opening and drifting sideways, so that the camera could not see the closing in obstruction.

Figure 4.16 illustrates the performance of our navigation system in a real-world scenario. We implemented our algorithm onboard an Asctec Pelican quadrotor which was equipped with an Intel Core2Duo, 1.86 GHz processor and conducted flight experiments in a test forest (Fig. 4.6).

Our system setup used the sensor fusion approach from [63] for pose estimation. It fuses IMU and position updates from a visual SLAM algorithm (Parallel Tracking and Mapping (PTAM) [30]) that uses images from a downward-looking camera (752 $\times$ 480, grayscale, 100 FOV).

To generate stereo disparity maps, we mounted a stereo camera system [20] on top of the quadrotor that included an embedded OMAP3730 to off-load the calculation of real-time disparity maps (376 $\times$ 240, 25 Hz, 12 cm baseline, 110 FOV) from the main processor, which only performed postprocessing of disparity maps within the stereo vision pipeline.

In this setup, the pose estimation framework used 59 % of the total resource (camera and PTAM (30 Hz) 48 %, pose filter (30 Hz) 11 %), stereo postprocessing

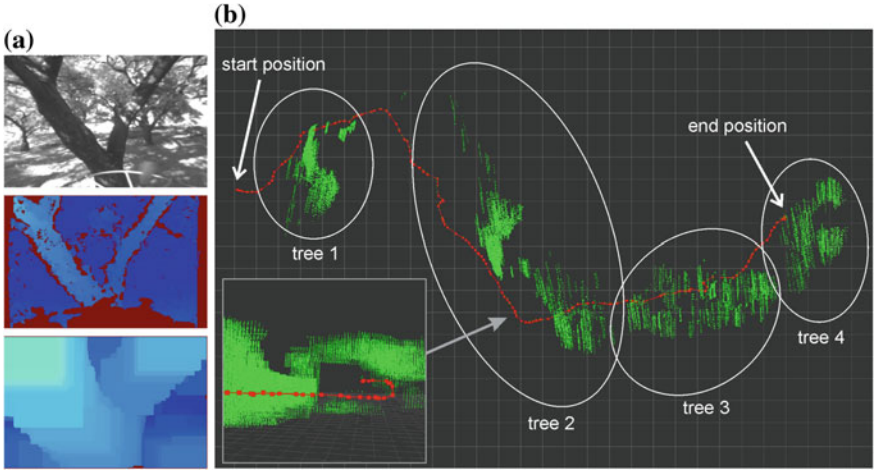

Fig. 4.16 Flying under tree canopy: **a** *top* to *bottom*: left rectified stereo camera view; disparity map; C-space map with expansion size of 60 cm; **b** *top–down* view of flown trajectory (*red*) with overlaid accumulated C-space point clouds (*green*) that are generated from subsampled C-space disparity maps; only points are shown that are more than 2.5 m above the ground and not farther away from the vehicle than 2 m. The *inset* shows a view from behind the vehicle in an upright view, illustrating flight below a tree branch

used 9 %, and motion planning used 65 % (RRT planner 60 %, trajectory server 5 %), generating motion plans at 2 Hz. On average, 6.51 % of the time used by the RRT planner was spent on C-space expansion, and 8.19 % on collision checking, which demonstrates the effectiveness of our approach. Additionally, C-space expansion and collision checking required less than 1.5 MB of memory for processing 376 × 240 disparity images from the stereo system. In total, the navigation frame work used 133 % of the available 200 % processing power of our two core CPU.

Figure 4.16a gives an example of the viewing volume that is used for motion planning when approaching a tree. Figure 4.16b illustrates the flight trajectory of a representative experiment in a top-down view. The vehicle started at the left of the image and was commanded to fly to the right, avoiding four different trees on its traverse.

At the beginning of each experiments, we started generating motion plans at an altitude of 4 m and restricted the planning volume to sample 3D points between 2 and 5 m altitude. The vehicle was able to avoid obstacles autonomously and plan its motion around tree branches. This included a portion of the trajectory where the vehicle flew correctly below an overhead branch (tree 2, Fig. 4.16b).

4.4 Autonomous Landing

In this section, we introduce autonomous landing as a high-level navigation application. Even if autonomous landing is a specific task, its single parts can easily be adapted to a series of different applications. Our approach comprises two research topics, dense monocular 3D reconstruction and visual surface analysis. First, we will describe the related work, then introduce our landing algorithm, followed by the embedded implementation, and concluded with experimental results.

4.4.1 Related Work

Most prior work on autonomous landing of unmanned aerial vehicles addresses landing on terrain, instead of finding elevated perches like rooftops. Due to the severe constraints on size, weight, and power (SWaP) for especially micro aerial vehicles, applicable methods must use much lighter, lower performance sensing and computing resources than available on larger scale systems [53]. Approaches amenable to these SWaP constraints frequently employ monocular [10, 26, 40, 60] and binocular stereo [37, 61] camera systems to map and analyze terrain. Most approaches perform some form of 3D terrain reconstruction, then assess planarity and slope of appropriately-sized terrain patches. Binocular stereo vision approaches are, due to the fixed intercamera geometry, algorithmically simpler, but are limited by the fixed interocular baseline and also heavier due to the additional camera. Three monocular approaches are particularly relevant here. The first tracks point features to estimate homographies from image pairs for predominantly planar terrain, then analyzes correlation coefficients for dense matches to segment in-plane and out-of-plane pixels [10]. The second uses a recursive filter at each pixel, image matching via gradient descent with intensity derivatives, and a plane plus parallax formulation of structure from motion to estimate dense elevation maps from image sequences [60]. Both of these address finding landing sites on the ground. The third uses multiplanar homography alignment with tracked features to segment a planar ground-level surface from an elevated, planar landing site [11].

In the last couple of years, significant progress has been made in dense monocular 3D reconstruction as well. Recently published approaches use Bayesian and variational estimation models with known camera motion [43, 46, 59]. All of them use powerful processing hardware, such as GPUs, to achieve real-time capability. An overview about earlier work can be found in [55].

4.4.2 Algorithm Description

Our approach can find flat landing platforms everywhere in the 3D model and is not limited to dominant planes. The presented algorithm consists of three parts, whereas the whole approach is designed to achieve reasonable short and constant processing time even on limited computing hardware. First, we use a dense motion

stereo method to determine a 3D model of the scene beneath the MAV. We present a frame list approach with variable baseline which enables arbitrarily selection of depth accuracy of the 3D model as long as the motion between an image pair could be found correctly. The scale can be determined from any metric pose estimator or altitude sensor. Here, we use the pose estimator presented in Sect. 4.2. Second, we analyze the 3D model in order to find potential landing candidates. Most of the mentioned work uses the dimensions of the MAV, the size, the planarity, and slope of the landing spot as main criterion of landability. We reduce all these criterions to simple steps which enable efficient onboard implementation. Third, we pick the most promising candidate and approach it, e.g., with a two-waypoint trajectory. Figure 4.17 illustrates the processing pipeline of our autonomous landing approach. Besides experiments where we actually land autonomously in a controlled environment, we present more detailed analysis about the system performance with hand-labeled ground truth data.

4.4.2.1 3D Reconstruction

Dense motion stereo is based on the same principle as conventional stereo, with the difference that the two views of the captured scene are generated by a single moving camera instead of a rigid stereo bar. The extrinsic parameters (rotation R and translation t between the two camera positions) have to be determined for each image pair individually. Translation can be estimated up to scale using visual information only. We assume the intrinsic parameters do not change and calibrate them in advance. We use a CAHVORE camera model [17] to model lens effects and to generate linearized camera models that describe the perspective projection.

For selection of a proper image pair, we maintain a frame list of the last n images. Each element of the frame list consists of camera image, camera pose in the world frame, extracted features (STAR [1], MSURF [9]), and a feature track list to record how often each feature has been found in the frame list. Given this data, we can select image pairs using two criteria. First, since depth accuracy is a function of the stereo baseline, we look for images that are an appropriate distance apart to achieve enough depth accuracy (at ground level) at the current altitude of the MAV. Second, we chose the image which exceeds a minimum number of successive feature matches with the current image. As soon as an image pair is found, we estimate R and t between the images with a multiplanar homography alignment approach [12]. Since we can estimate translation only up to scale from pure visual information (without some metric context), the translation vector is then scaled with the real-world baseline from the visual-inertial state estimator described in Sect. 4.2. Having R and t, stereo rectification can be applied. The quality of the motion estimation strongly depends on the accuracy of the feature locations and, thus, is scene dependent. To discard poor motion estimates in order to prevent wrong 3D reconstruction, we calculate the average 3D reprojection error of the feature pairs and accept only image pairs with an error in subpixel range. Finally, we use a real-time sum of absolute difference stereo matching algorithm to estimate a disparity map from which we generate a 3D point cloud to model the captured scene beneath the MAV.

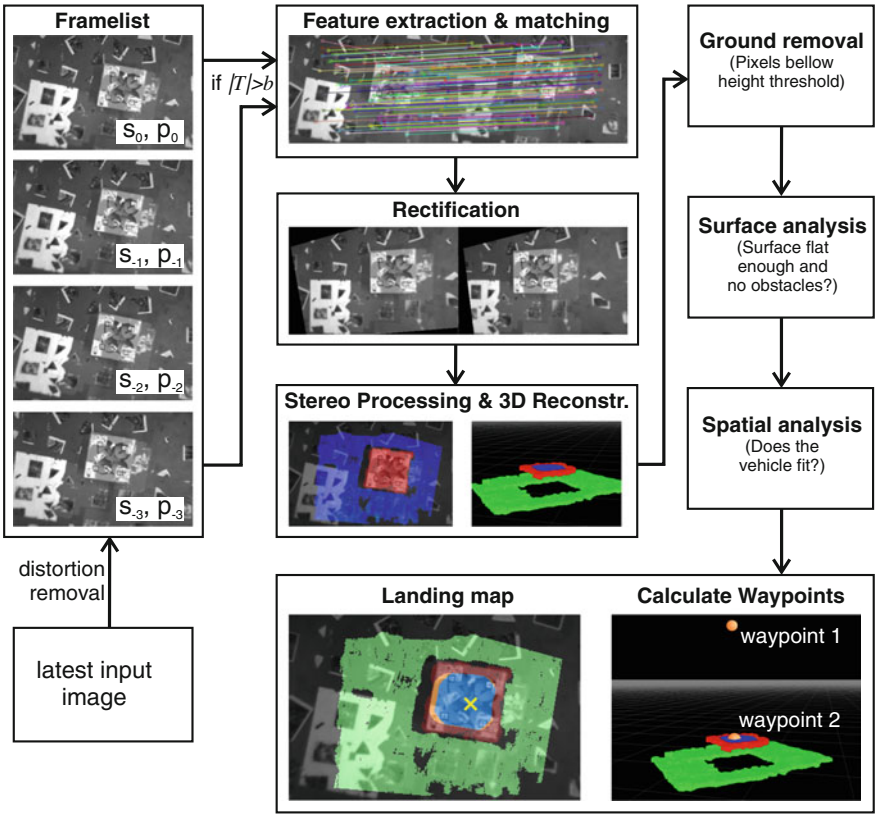

Fig. 4.17 Autonomous landing overview: New input images are stored in a frame list, together with detected features, and camera poses. For selected image pairs, camera motion is estimated, and the 3D model is determined. The result of the landing site detection is a landing map which labels all pixels as: *green* (ground level), *red* (on rooftop but unsafe), *orange* (insufficient space), or *blue* (safe landing area). The location with the highest confidence is labeled by $\times$

4.4.2.2 Landing Site Detection

After 3D reconstruction, the next step is to find potential landing candidates. We define the following requirements for a suitable landing site: (a) A landing site has to be planar, close to parallel to the ground plane, and free of obstacles and hazards, (b) it has to be large enough to fit the MAV, and (c) it has to provide enough free space around it for a safe approach.

To fulfill these requirements, we developed an efficient multistep algorithm which uses the determined range data to reduce the problem to a basic probabilistic model. Since our application is targeted to land on elevated surfaces for surveillance, we first remove all candidates close to the ground level. Then we calculate the standard deviation of the disparity map because the variance of the disparity map along the gravity

vector, after projection into world frame, corresponds to the planarity of the landing surface. The smaller the standard deviation in the disparity map, the more planar the corresponding area and, thus, the higher the landing site confidence (normalized standard deviation). Standard deviation is calculated in an adaptive neighborhood

$$n(d, h) = \frac{r z_{\text{acc}} f}{h^2} d \tag{4.17}$$

which depends on the disparity values d and the MAV's altitude h. The size of the MAV (r is the radius), the focal length f, and the target depth accuracy z_{acc} are constant. The target depth accuracy is the first derivative of z (here h) in respect to the disparity. The needed space for the MAV is its size divided by the lateral resolution. These two equations combined result in Eq. 4.17. In fact, we make sure that the window size (in pixels) for the standard deviation corresponds to the size (in meters) the MAV needs to land. The result of the algorithm is a landing map which labels all pixels (i.e., landing sites) as whether or not they are safe to land. A confidence map assigns each landing candidate a quality value which can further be used to pick the final landing target.

4.4.3 Embedded Implementation

We initially implemented our landing framework on a standard PC and then ported the software to the processing board (see Sect. 4.1.1). First, we applied thorough software optimization techniques to make sure that no redundant calculations are done and all data structures can be accessed fast. Additional software optimization steps were including NEON and ARM specific instruction sets. We also reduced of image resolution to 376×240 which increased the processing speed significantly, while still maintaining sufficient resolution. The decreased depth and lateral resolution, which come along with the smaller image resolution, are not an issue: For depth, our approach compensates this by automatically choosing a larger baseline and the lateral resolution turned out to still be high enough, even for a flight altitude of above 10 m. When running the landing site detection algorithm in parallel with our pose estimation frame work on the Hummingbird/U2, we achieved a frame rate of 1 Hz for landing map updates. Our experiments showed that this frame rate is reasonable for fully autonomous landing site detection.

4.4.4 Experimental Evaluation

To evaluate proper system performance, we ran three indoor experiments and one outdoor experiment. All indoor experiments were conducted with the Asctec Hummingbird carrying the modified U2 flight computer and all outdoor experiments were

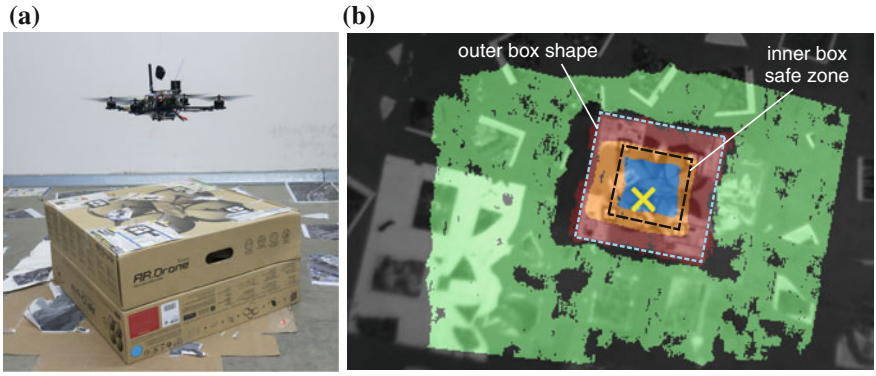

Fig. 4.18 First indoor experiment: **a** *Box* as surrogate rooftop; **b** hand-labeled landing map, *outer box* shape is the edge of the box surface, *inner box* safety zone is the box surface without the 13 cm border region

conducted with the Asctec Pelican carrying the Asctec Mastermind flight computer. Even if the whole processing pipeline runs onboard, for the presented experiments, we recorded the data and processed it offline.

4.4.4.1 Indoor Evaluation

The first indoor experiment was designed to generate quantitative error metrics from hand-labeled ground truth data. We fly our quadrotor system at three different altitudes over a box ($57 \times 57 \times 27\,\text{cm}^3$, Fig. 4.18) to simulate a rooftop landing scenario. In all experiments, the vehicle radius was set to 13 cm to allow for a sufficiently-sized valid landing area in the middle of the box, and the arbitrary ground-level cut off threshold was set to 20 cm. For ground truth, the true landing area in the middle of the box surface was marked by corner marks that were located at 13 cm distance from the box edges, and these were manually identified in the input images (see Fig. 4.18b). The first three rows of Table 4.2 give an overview of the evaluation results. Altitude, baseline, and feature reprojection error during image alignment correspond to the average value for each experiment. For this evaluation, we only considered frames where the box surface was completely visible in the disparity image to avoid border effects, and were valid stereo results could be calculated (enough baseline and more than 40 feature matches for image alignment) ("frames visible"). Within these frames, we defined a successful detection ("frames successful") as frames where at least one valid landing location was detected on the box and no landing location was detected falsely on the ground. For these successful frames, we also calculated false positives (FP) rates (pixels that were classified as valid landing area that are located in the border area), false negatives (FN) rates (not as landing area classified pixels that were located in the correct center area of the box). Note that we only consider pixels with valid disparity values in this metric.

Table 4.2 Evaluation of landing site detection

Altitude (m)	Baseline (m)	Reprojection error (pixel)	Number of frames			FP (%)	FN (%)
			Visible	Successful			
1.47	0.39	0.21	116	106	91.4 %	0.028	39.34
2.19	0.86	0.20	63	62	98.4 %	0.046	43.57
3.34	2.00	0.18	67	63	94.0 %	0.0026	53.14
6.51	1.16	0.28	337	325	96.3 %		
10.15	2.28	0.22	480	433	90.2 %		

Row 1–3 Indoor experiment at three different altitudes ($r = 13$ cm, depth accuracy at ground level = 3 cm), *Row 4–5* outdoor experiment (depth accuracy at ground level = 20 cm)

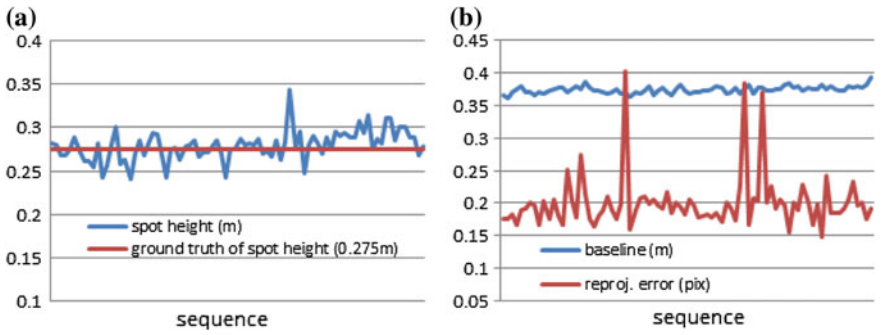

Fig. 4.19 Second indoor experiment: **a** box height calculation for valid landing point with highest confidence. **b** Feature reprojection error of feature matches and estimated baseline (depth accuracy 3 cm)

Our approach is able to robustly detect the landing zone with a success rate of more than 90 % in all experiments and a false positive rate below 0.05 %. The false positive (FP) and false negative (FN) rates are largely defined by the quality of the disparity input. Border-fattening effects (caused by our correlation-based stereo matching) usually increase the FP rate, whereas missing disparity pixels on top of the target increase the FN rates, since we treat missing data as unsafe. To mitigate these two effects, we introduced two thresholds to maximize safety: (1) at disparity edges, we disable all pixels that are located within half a correlation window size to the edge, and (2) we use a percentage threshold which defines the minimum number of pixels with valid disparity around a landing site (98 % in our experiments).

In the second experiment, in order to verify the accuracy of our 3D reconstruction, we plotted the height of the landing point with the highest confidence for one of the sequences (Fig. 4.19). The error follows the expected depth accuracy of 3 cm (min. depth acc.) within the true box height of 27.5 cm. The low average feature reprojection error confirms valid motion estimation results.

The third indoor landing experiment consisted of landing site detection and target approach. In this indoor experiment, the landing target also consisted of a cardboard box to simulate an elevated landing surface. We commanded the quadrotor to fly

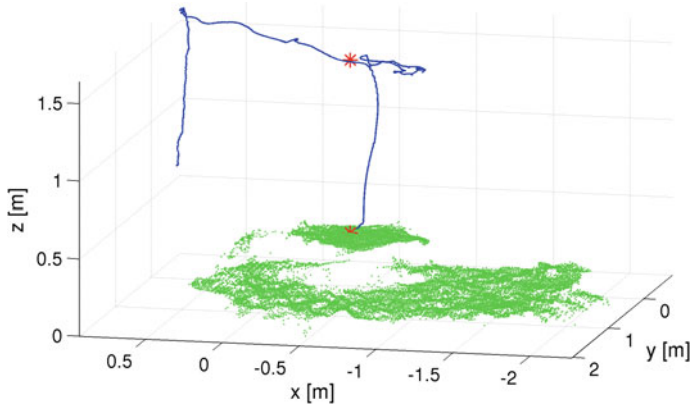

Fig. 4.20 Third indoor experiment: The MAV is commanded to take off autonomously and rise up to 1.5 m. Then we continuously set waypoints to explore the environment (the *solid line* follows the flight trajectory). Once a valid landing spot is detected, the two approach waypoints are generated (*stars*) which are used by the MAV for the landing maneuver. Note that each waypoint includes a tolerance radius (the trajectory does not hit the *red star* at 1.5 m altitude exactly). We reduced this tolerance for the actual landing spot on the surface to enable high-precision landing

over the landing zone by defining manual waypoints, which were approached by the vehicle autonomously while executing the landing site detection algorithm to analyze the area beneath the MAV. As soon as an appropriate landing spot was detected, the two approach waypoints were submitted and executed by the vehicle (Fig. 4.18a).

Figure 4.20 depicts the 3D point cloud of the reconstructed box and ground surface together with the flight trajectory and the issued approach waypoints. The vehicle took off to the left of the illustrated scene and landed correctly on top of the box. Example scene views, together with a resulting landing map are illustrated in Fig. 4.17. All pixels in the middle of the box have been labeled correctly as safe to land (blue), whereas pixels close to the edges of the box are detected as either unsafe (red) or provide not enough space to land on (orange).

4.4.4.2 Outdoor Evaluation

For the outdoor experiments, we conducted overflights over a one story building (Fig. 4.21) and recorded image sequences from the downward-looking camera together with pose data for offline analysis. A quantitative evaluation for two different overflights is given in Table 4.2 row 3–4. The average altitude of the first flight was 6.5 m which lead to an average required baseline of 1.16 m. The second overflight was at a higher altitude of approximately 10.15 m requiring a slightly higher minimum baseline of 2.28 m on average. From all frames, were at least a part of the safe landing zone on top of the building was visible in the disparity images, we could successfully identify a valid landing target in over 90 % for both flights.

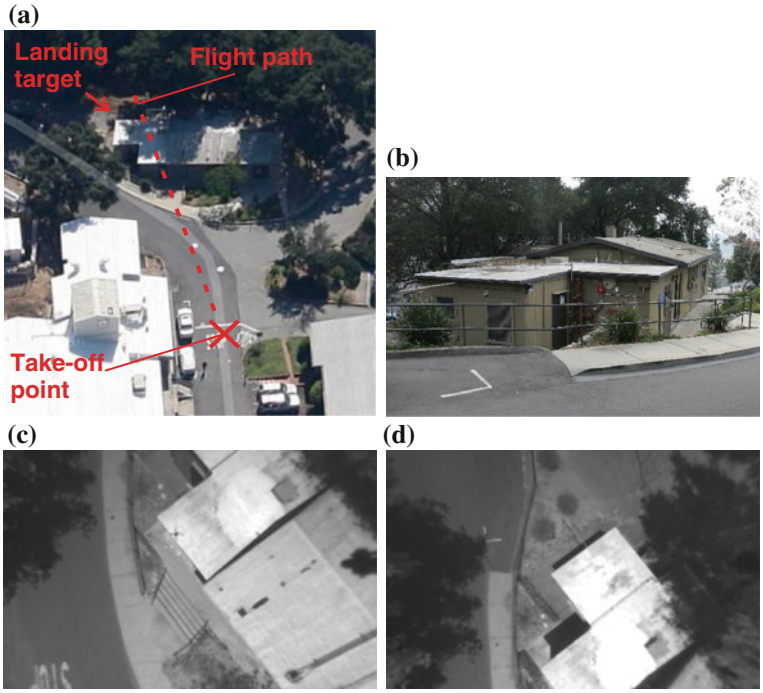

Fig. 4.21 Outdoor experiment environment and data set images. **a** Aerial view of the flight area and target building; **b** the flat area on the roof is the landing target; **c** raw input image with good texture on roof; **d** image with saturation area which leads to missing stereo data

4.5 Conclusion and Future Work

Vision-based navigation algorithms have the potential to become an enabling technology for micro air vehicle autonomy. With the advent of small, low-power processing units and miniature camera modules from the cell-phone sector, low SWaP computing for vision applications is ready to be deployed, enabling fully autonomous navigation of very small platforms for the first time. In this chapter, we presented three different fundamental building blocks for platform autonomy: vision-based pose estimation, onboard obstacle avoidance, and autonomous landing. Fast pose estimation that is independent of external sensor inputs is the basis for safe MAV operations. Our approach fuses accurate map-based localization with a fast map-free approach to estimate vehicle velocities in emergency situations when a map-based approach might fail.

Obstacle avoidance is a key capability for flights in highly cluttered environment or close to the ground. We use frontal stereo vision approach which provides a polar-perspective, inverse-range world representation for obstacle detection and collision checking with low computational complexity, and deploy a closed-loop motion

planning approach that plans collision-free trajectories while accounting for vehicle dynamics.

Our autonomous landing approach finds elevated landing surfaces by executing a dense structure from motion approach, and searching for safe landing zones in the reconstructed terrain.

We implemented all three algorithms on our quadrotor platforms and demonstrated autonomous flights using only onboard resources.

In the future, we plan to further integrate our embedded platform components towards ultimately having a fully capable avionics package (flight computer, camera, and IMU) under 15 g. This will enable fully autonomous control of ultra-small quadrotor systems (as, e.g., the 15 cm, 25 g Bitcraze miniature quadrotor system [16]) that can be operated in highly cluttered environment or confined spaces, indoor and outdoor.

Acknowledgments This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

1. Agrawal M, Konolige K, Blas MR (2008) Censure: center surround extremas for realtime feature detection and matching. Computer vision ECCV 2008. Lecture Notes in Computer Science, vol 5305. Springer, Berlin, pp 102–115
2. Armesto L, Tornero J, Vincze M (2007) Fast ego-motion estimation with multi-rate fusion of inertial and vision. Int J Robot Res 26(6):577–589
3. Armesto L, Chroust S, Vincze M, Tornero J (2004) Multi-rate fusion with vision and inertial sensors. In: Proceedings of the IEEE international conference on robotics and automation, New Orleans, US
4. Ascending Technologies GmbH. http://www.asctec.de/uav-applications/research/products/asctec-mastermind/
5. Bachrach A, Prentice S, He R, Henry P, Huang AS, Krainin M, Maturana D, Fox D, Roy N (2012) Estimation, planning and mapping for autonomous flight using an RGB-D camera in GPS-denied environments. Int J Robot Res 31(11):1320–1343
6. Bajracharya M, Howard A, Matthies L, Tang B, Turmon M (2009) Autonomous off-road navigation with end-to-end learning for the LAGR program. Field Robot 26(1):3–25
7. Bakolas E, Tsiotras P (2008) Multiresolution path planning via sector decompositions compatible to on-board sensor data. In: AIAA guidance, navigation, and control conference
8. Baldwin G, Mahony R, Trumpf J (2009) A nonlinear observer for 6 DOF pose estimation from inertial and bearing measurements. In: Proceedings of the IEEE international conference on robotics and automation, Kobe
9. Bay H, Ess A, Tuytelaars T, Van Gool L (2008) Speeded-up robust features (surf). Comput Vis Image Underst 110(3):346–359
10. Bosch S, Lacroix S, Caballero F (2006) Autonomous detection of safe landing areas for an uav from monocular images. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems, pp 5522–5527
11. Brockers R, Susca S, Zhu D, Matthies L (2012) Fully self-contained vision-aided navigation and landing of a micro air vehicle independent from external sensor inputs. In: Proceedings of the SPIE, 8387:83870Q-1–83870Q-10

12. Cheng Y (2010) Real-time surface slope estimation by homography alignment for spacecraft safe landing. In: Proceedings of the IEEE international conference on robotics and automation, pp 2280–2286
13. Chroust SG, Vincze M (2004) Fusion of vision and inertial data for motion and structure estimation. J Robot Syst 21(2):73–83
14. Conroy J, Gremillion G, Ranganathan B, Humbert J (2009) Implementation of wide-field integration of optic flow for autonomous quadrotor navigation. Auton Robot 27(3):189–198
15. Corke P (2004) An inertial and visual sensing system for a small autonomous helicopter. Int J Robot Syst 21(2):43–51
16. Crazyflie Micro Quadrotor. http://www.bitcraze.se/crazyflie/
17. Di K, Li R (2004) CAHVOR camera model and its photogrammetric conversion for planetary applications. J Geophys Res 109:E04004
18. Fraundorfer F, Heng L, Honegger, D, Lee GH, Meier L, Tanskanen P, Pollefeys M (2012) Vision-based autonomous mapping and exploration using a quadrotor MAV. In: IROS, pp 4557–4564
19. Gemeiner P, Einramhof P, Vincze M (2007) Simultaneous motion and structure estimation by fusion of inertial and vision data. Int J Robot Res 26(6):591–605
20. Goldberg SB, Matthies L (2011) Stereo and IMU assisted visual odometry on an OMAP3530 for small robots. In: 2011 IEEE computer society conference on computer vision and pattern recognition workshops (CVPRW), pp 169–176
21. Hardkernel. http://www.hardkernel.com
22. How JP, Bethke B, Frank A, Dale D, Vian J (2008) Real-time indoor autonomous vehicle test environment. IEEE Control Syst Mag 28(2):51–64
23. Hrabar S, Sukhatme GS, Corke P, Usher K, Roberts J (2005) Combined optic-flow and stereo-based navigation of urban canyons for a uav. In: IROS
24. Huster A, Frew EW, Rock SM (2002) Relative position estimation for AUVs by fusing bearing and inertial rate sensor measurements. In: Proceedings of the oceans conference, vol 3. MTS/IEEE, Biloxi, pp 1857–1864
25. Hyslop AM, Humbert JS (2010) Autonomous navigation in three-dimensional urban environments using wide-field integration of optic flow. Guid Control Dyn 33(1):147
26. Johnson A, Montgomery J, Matthies L (2005) Vision guided landing of an autonomous helicopter in hazardous terrain. In: Proceedings of the IEEE international conference on robotics and automation, pp 3966–3971
27. Jones E (2009) Large scale visual navigation and community map building. PhD thesis, University of California at Los Angeles
28. Jones E, Soatto S (2010) Visual-inertial navigation, mapping and localization: a scalable real-time causal approach. Int J Robot Res 30:407–430
29. Kelly J, Sukhatme GS (2011) Visual-inertial sensor fusion: localization, mapping and sensor-to-sensor self-calibration. Int J Robot Res (IJRR) 30(1):56–79
30. Klein G, Murray D (2007) Parallel tracking and mapping for small AR workspaces. In: Proceedings of the 2007 6th IEEE and ACM international symposium on mixed and augmented reality, ISMAR'07. IEEE Computer Society, p 110
31. Kuwata Y, Teo J, Fiore G, Karaman S, Frazzoli E, How JP (2009) Real-time motion planning with applications to autonomous urban driving. Trans Control Syst Tech 17(5):1105–1118
32. Luders B, Karaman S, Frazzoli E, How J (2010) Bounds on tracking error using closed-loop rapidly-exploring random trees. In: American control conference, Baltimore, MD, pp 5406–5412
33. Lupton T, Sukkarieh S (2008) Removing scale biases and ambiguity from 6DoF monocular SLAM using inertial. In: International conference on robotics and automation, Pasadena, California
34. Lupton T, Sukkarieh S (2009) Efficient integration of inertial observations into visual SLAM without initialization. In: IEEE/RSJ international conference on intelligent robots and systems, St. Louis

35. MacAllister B, Butzke J, Kushleyev A, Pandey H, Likhachev M (2013) Path planning for non-circular micro aerial vehicles in constrained environments. In: ICRA, pp 3918–3925
36. Martin GR (2009) What is binocular vision for? a birds eye view. J Vis 9(11):245–267
37. Meingast M, Geyer C, Sastry S (2004) Vision based terrain recovery for landing unmanned aerial vehicles. In: Proceedings of the IEEE conference on decision and control, vol 2. pp 1670–1675
38. Mei C, Sibley G, Cummins M, Newman P, Reid I (2009) A constant time efficient stereo SLAM system. In: Proceedings of the British machine vision conference (BMVC)
39. Mellinger D, Kumar V (2011) Minimum snap trajectory generation and control for quadrotors. In: Proceedings of the IEEE international conference on robotics and automation (ICRA)
40. Montgomery J, Johnson A, Roumeliotis S, Matthies L (2006) The jet propulsion laboratory autonomous helicopter testbed: a platform for planetary exploration technology research and development. J Field Robot 23(3–4):245–267
41. Mourikis AI, Roumeliotis SI (2007) A multi-state constraint Kalman filter for vision-aided inertial navigation. In: Proceedings of the IEEE international conference on robotics and automation (ICRA)
42. Mourikis AI, Trawny N, Roumeliotis SI, Johnson AE, Ansar A, Matthies L (2009) Vision-aided inertial navigation for spacecraft entry, descent, and landing. IEEE Trans Robot 25(2):264–280
43. Newcombe RA, Lovegrove JS, Davison AJ (2011) Dtam: dense tracking and mapping in real-time. In: IEEE international conference on computer vision (ICCV), pp 2320–2327
44. Otte MW, Richardson SG, Mulligan J, Grudic G (2009) Path planning in image space for autonomous robot navigation in unstructured outdoor environments. Field Robot 26(2):212–240
45. Pivtoraiko M, Mellinger D, Kumar V (2013) Incremental micro-UAV motion replanning for exploring unknown environments. In: ICRA
46. Pizzoli M, Forster C, Scaramuzza D (2014) Remode: probabilistic, monocular dense reconstruction in real time. In: Proceedings of the IEEE international conference on robotics and automation
47. Qian G, Chellappa R, Zheng Q (2002) Bayesian structure from motion using inertial information. In: International conference on image processing, Rochester, New York
48. Richter C, Bry A, Roy N (2013) Polynomial trajectory planning for quadrotor flight. In: RSS workshop on resource-efficient integration of perception, control and navigation
49. Robot Operating System, (ROS). http://www.ros.org
50. Ross S, Melik-Barkhudarov N, Shankar KS, Wendel A, Dey D, Bagnell JA, Hebert M (2013) Learning monocular reactive uav control in cluttered natural environments. In: ICRA, pp 1757–1764
51. Roumeliotis SI, Johnson AE, Montgomery JF (2002) Augmenting inertial navigation with image-based motion estimation. In: Proceedings of The IEEE international conference on robotics and automation, Washington, pp 4326–4333
52. Sarabandi K, Vahidpour M, Moallem M, East J (2011) Compact beam scanning 240 GHz radar for navigation and collision avoidance. In: SPIE, vol 8031
53. Scherer S, Chamberlain L, Singh S (2012) Autonomous landing at unprepared sites by a full-scale helicopter. Robot Auton Syst 60(12):1545–1562
54. Schouwenaars T, De Moor B, Feron E, How J (2001) Mixed Integer Programming for Multi-Vehicle Path Planning. In: Proceedings of the European control conference, Porto, Portugal
55. Seitz SM, Curless B, Diebel J, Scharstein D, Szeliski R (2006) A comparison and evaluation of multi-view stereo reconstruction algorithms. In: IEEE computer society conference on computer vision and pattern recognition, 2006, pp 519–528
56. Shen S, Michael N, Kumar V (2011) 3d indoor exploration with a computationally constrained mav. In: Robotics science and systems
57. Shen S, Michael N, Kumar V (2011) Autonomous multi-floor indoor navigation with a computationally constrained MAV. In: Proceedings of the IEEE international conference on robotics and automation

58. Strelow D, Singh S (2003) Online motion estimation from image and inertial measurements. In: Workshop on integration of vision and inertial sensors (INERVIS), Coimbra, Portugal
59. Stühmer J, Gumhold S, Cremers D (2010) Real-time dense geometry from a handheld camera. In: Proceedings of the 32nd DAGM conference on pattern recognition, pp 11–20
60. Templeton T, Shim DH, Geyer C, Sastry SS (2007) Autonomous vision-based landing and terrain mapping using an MPC-controlled unmanned rotorcraft. In: Proceedings of the IEEE international conference on robotics and automation, pp 1349–1356
61. Theodore C, Rowley D, Hubbard D, Ansar A, Matthies L, Goldberg S, Whalley M (2006) Flight trials of a rotorcraft unmanned aerial vehicle landing autonomously at unprepared sites. In: Forum of the American helicopter society, Phoenix
62. Weiss S (2012) Vision based navigation for micro helicopters. PhD thesis, ETH Zurich, March 2012
63. Weiss S, Achtelik MW, Lynen S, Achtelik MC, Kneip L, Chli M, Siegwart R (2013) Monocular vision for long-term micro aerial vehicle state estimation: a compendium. Field Robot 30(5):803–831
64. Weiss S, Achtelik MW, Chli M, Siegwart R (2012) Versatile distributed pose estimation and sensor self-calibration for an autonomous MAV. In: IEEE International conference on robotics and automation (ICRA)
65. Weiss S, Achtelik MW, Lynen S, Chli M, Siegwart R (2012) Real-time onboard visual-inertial state estimation and self-calibration of MAVs in unknown environments. In: IEEE International conference on robotics and automation (ICRA)
66. Weiss S, Brockers R, Matthies L (2013) 4dof drift free navigation using inertial cues and optical flow. In: IEEE/RSJ International conference on intelligent robots and systems (IROS), pp 4180–4186
67. Weiss S, Siegwart R (2011) Real-time metric state estimation for modular vision-inertial systems. In: Proceedings of the IEEE International conference on robotics and automation (ICRA)
68. Yu H, Beard RW (2013) A vision-based collision avoidance technique for miniature air vehicles using local-level frame mapping and path planning. Auton Robots 34(1–2):93–109