Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

Collision Avoidance for Unmanned Aerial Vehicles using Monocular Vision

---

Gert Hutter

Graz, Austria, September 2013

Thesis supervisors

Univ.-Prof. Dipl.-Ing. Dr. techn. Horst Bischof

Dipl.-Ing. Dr. techn. Andreas Wendel

All truths are easy to understand once they are discovered; the point is to discover them.

Galileo Galilei

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

(date)	(signature)

Abstract

Unmanned Aerial Vehicles (UAVs) have become considerably popular over the last years due to cheap and powerful available hardware and a widespread field of application. An important step towards autonomous flight capabilities is to detect obstacles and to avoid potential collisions during flight. In this master thesis we present a novel system for collision avoidance on UAVs based on visual input from the monocular on-board camera. Besides the image stream we utilize data from the Inertial Measurement Unit (IMU) for short-term pose estimation of the UAV. Three dimensional information is extracted with an efficient Structure-from-Motion (SfM) approach based on sparse feature matching. A meshing technique is then utilized to fill gaps between sparse points leading to a more complete reconstruction of the ambient structure. Subsequent reconstructions are integrated into a probabilistic occupancy map that models free, occupied, and unknown space. In case of close obstacles proper reactive actions are taken to avoid a collision. The system is evaluated in terms of accuracy and real-time capabilities and we illustrate results of fully-autonomous and semi-autonomous flights.

Keywords: UAV, collision avoidance, obstacle, autonomous, drone, quad-rotor helicopter, occupancy map.

Kurzfassung

Unbemannte Flugdrohnen, sogenannte UAVs (Unmanned Aerial Vehicles), wurden in den letzten Jahren immer beliebter. Sinkende Hardwarekosten und ausgereifte Technik forcieren zahlreiche Anwendungsgebiete im privaten, militärischen oder akademischen Bereich. Ein wichtiger Schritt in Richtung "autonome Flugfähigkeiten" ist die automatische Erkennung von Hindernissen und die Vermeidung von Kollisionen während eines Fluges. In dieser Abschlussarbeit präsentieren wir ein neuartiges System zur Kollisionserkennung und -vermeidung für unbemannte Flugdrohnen. Das System detektiert potentielle Hindernisse anhand des visuellen Inputs der On-Board Kamera. Neben den Kamerabildern werden Daten der Inertial Measurement Unit (IMU) verwendet um Position und Orientierung der Drohne abzuschätzen. Tiefeninformation wird aus den Kamerabildern durch ein effizientes Structure-from-Motion (SfM) Verfahren gewonnen, welches korrespondierende Bildpunkte dreidimensional rekonstruiert. Die entstehende Punktwolke wird mit Hilfe eines Meshing-Algorithmus verdichtet um ein vollständigeres Bild der Umgebung zu erhalten. Alle Rekonstruktionsergebnisse werden in eine probabilistische Landkarte integriert, welche freie, besetzte und unbekannte Bereiche der Umgebung modelliert. Im Falle von nahen Hindernissen werden dem Anwendungsfall entsprechende Maßnahmen getroffen um einer Kollision vorzubeugen. Das System wurde bezüglich Genauigkeit und Echtzeittauglichkeit evaluiert und es wurden autonome sowie semi-autonome Flüge durchgeführt.

Stichworte: UAV, Kollisionserkennung, Hindernis, autonom, Drohne, Quadrocopter, probabilistische Landkarte.

Acknowledgments

I want to thank my parents Günther and Erika, for their great moral and financial support during my years of studying. Moreover, I want to acknowledge all my fellow-students and my brother Günther for their help and friendship over the years. A special thank you goes from here to my girlfriend Christine for her outstanding patience and motivation during completion of this thesis.

Finally, I want to express my gratitude to my thesis supervisors Univ.-Prof. Dipl.-Ing. Dr. techn. Horst Bischof and Dipl.-Ing. Dr. techn. Andreas Wendel for the great collaboration throughout this thesis. It was a pleasure for me to be part of this highly professional team of Computer Vision experts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Unmanned aerial vehicles (UAVs) have become a topic of great interest over the last years. Especially small models, called micro aerial vehicles or MAVs, have become affordable and constitute an evolving robot platform. Low costs, advancing hardware, and various on-board sensors are only a few reasons for their increasing popularity. Those flying drones are used in manifold applications of both academic and non-academic nature. Common applications include surveillance, photogrammetry, inspection tasks, 3D modeling, or search and rescue operations. To promote powerful UAV applications, autonomous flight capabilities are highly desired. In this work we tackle the problem of detecting obstacles and preventing collisions during a flight, which is known as *collision avoidance*.

1.1 Overview and Motivation

Typical UAV applications require a human operator who ensures safe and target oriented flights. Many processes could be automated by defining waypoints in the three-dimensional space, which the UAV visits subsequently. Autonomous flights could reduce the required human interaction and create opportunities for enhanced applications on unmanned aerial vehicles. To enable safe, autonomous flights it is necessary to take

care of 3 fundamental problems – *localization*, *obstacle detection* and *flight control*.

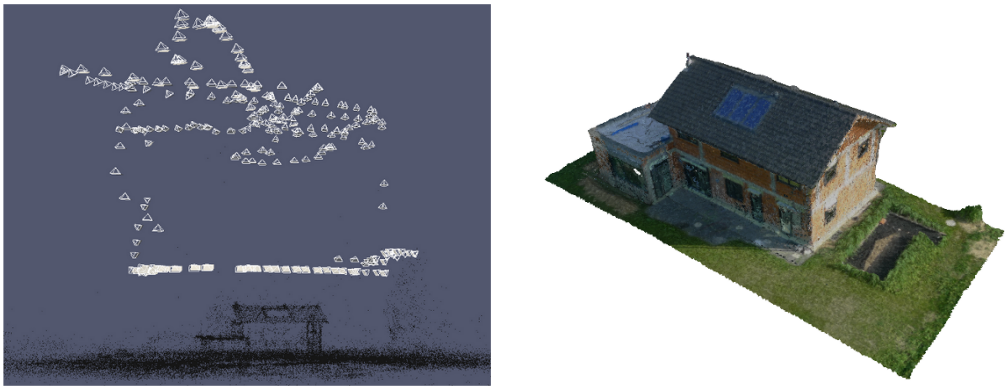
Localization. Localization is the task of estimating the UAV's current pose, i.e. position and orientation, in the three-dimensional space. This task is also referred to as *pose estimation* and is essential for flying along pre-defined paths. Using the global positioning system (GPS) is a widespread method for localization of ground and airborne robots. However, it requires the UAV to be equipped with an appropriate receiver and is restricted to outdoor scenarios. Furthermore, the accuracy of GPS is in the range of a few meters, which is not satisfactory for precise waypoint flights. Localization must therefore be obtained in an alternative way, for example from visual landmarks. Visual methods are beneficial since they only need a camera as input sensor, which most current UAVs are equipped with by default.

Obstacle Detection. Assuming adequate localization capabilities, information about the environment is required to assure collision-free flights. Prior knowledge about the environmental structure can be utilized only in some scenarios. In cluttered or dynamic environments this might not be enough to ensure safe flights. Unexpected obstacles or inaccurate localization can cause collisions despite the presence of prior information. It is therefore beneficial to detect potential obstacles during flight using on-board sensors. Sensing devices as laser range finders or ultrasonic sensors are well-suited but not available on most standard UAVs. Thus, utilizing an available on-board camera for obstacle detection is of great benefit.

Flight Control. An autonomous operating UAV needs a control mechanism to account for obstacles along the desired flight path. Assuming accurate localization and reliable obstacle information, it is possible to employ global path planning algorithms. Such algorithms are able to find collision-free paths in a given environment model. However, incomplete or noisy environment models restrict the feasibility of those methods in practice. Alternatively, low-level control mechanisms as reactive controllers can be applied. Such controllers only perform local path modifications or simply initiate the UAV to stop in case of close obstacles.

Solving those tasks are key requirements to enable autonomous flights. The CONSTRUCT project[1] constitutes one example of an ambitious UAV application that would highly benefit from autonomous flight capabilities [25, 27]. A micro aerial vehicle is used as image capturing platform to obtain imagery of construction sites from various viewpoints. From the set of aerial images a detailed 3D model is created using a sophisticated Structure-from-Motion (SfM) algorithm. Such models are created on a regular basis for progress monitoring and documentation throughout the construction process. Figure 1.1 illustrates sample images captured from the UAV and Figure 1.2 illustrates the resulting 3D model.

Figure 1.1: Sample images captured with the UAV from different viewpoints [25].



(a) (b)

Figure 1.2: The airborne images are utilized to reconstruct high quality 3D models. (a) Sparse reconstructed point cloud (black) and viewpoints from where the images were captured (white frustums). (b) The sparse point cloud is refined to a dense 3D model including texture [25].

To achieve high quality 3D models, the viewpoints from which aerial images are captured

[1]The CONSTRUCT project is a collaborative research project at Graz University of Technology funded by Siemens AG and the Austrian Research Promotion Agency.

must fulfill certain requirements [27]. Therefore, it is reasonable to plan paths in advance that contain those desired positions [25]. Flight planning is also important to fully exploit the limited flight time of the UAV. An autonomous flight along a 3D path would therefore facilitate the image capturing process significantly. However, it is essential that an autonomous operating UAV is able to detect unforeseen obstacles such as trees or construction cranes along its path. Using the camera as primary sensor for collision avoidance requires no additional sensors to be mounted on the UAV. This motivates our work on a system that is able to detect obstacles during a flight from visual input.

1.2 Thesis Goals

The goal of this thesis is to develop a system that detects obstacles close to the UAV during a flight. Moreover, it should be aware of obstacle-free space and be able to perform local path corrections towards a safer region. Obstacles should be detected purely from visual input of the on-board camera without using additional sensing devices, as ultrasonic or laser range sensors. In particular, we use a single, monocular camera and do not use a stereo setup or any kind of depth camera (as Microsoft Kinect$^{\text{TM}}$).

Our system should not be limited to work with one specific application scenario (e.g. modeling construction sites). Instead, it should run independently from the main task and provide a clear interface for inter-task communication. Proper reactive actions can be taken based on the characteristics of the flight mission. In some tasks it might be best to stop and wait for human interaction whereas other tasks allow local modifications of a given flight path.

We aim to use off-the-shelf hardware without any custom hardware modifications. The system design should be flexible enough to be applicable on multiple platforms and with several applications. Importantly, the entire process of detecting obstacles and taking appropriate actions should work in real-time in order to be of practical use.

1.3 Contributions

In this work we present a novel system for UAV collision avoidance based on visual input from the on-board camera. The main contributions are:

IMU-based Localization. We utilize data from the drone's Inertial Measurement Unit (IMU) for short-term localization. Our flexible system design also allows to use other localization methods, such as localization from visual landmarks. Benefits and drawbacks of our IMU-based localization approach are discussed.

Fast 3D Point Extraction. We implemented an efficient method to extract 3D points from the image stream. A meshing technique is used to interpolate 3D points at locations with high obstacle evidence and reject outliers at the same time.

Probabilistic Mapping. The environment is modeled as free, occupied, and unknown space using a three-dimensional occupancy map. This map incorporates 3D information collected during a flight and its probabilistic fashion makes it robust to noisy sensor data. We countervail global pose estimation drift as we only maintain parts of the map that are within a small volume around the UAV's position.

Reactive Controller. A controller periodically checks for obstacles in the probabilistic map. It can initiate reactive actions as guiding the drone towards free space or holding the current position. The actual reactive behavior can be customized to fit the needs of a certain application or environment.

Modular System Design. The core parts of the system are encapsulated into modular components. The individual components communicate over clearly defined interfaces and allow simple extension or exchange of system components.

1.4 Outline

The remainder of this work is organized as follows. In Chapter 2 we introduce topics related to our work and give an overview of recent collision avoidance methods. Chapter 3 focuses on localization and illustrates how we estimate the UAV's pose in a world coordinate system. In Chapter 4 we describe how 3D information is efficiently extracted from the image stream. Later in Chapter 5 we introduce the probabilistic occupancy map that models the environment near the UAV. We show our experiments in Chapter 6, where the core system parts are evaluated and example flights are illustrated. Finally, we conclude the work in Chapter 7 and give an outlook to possible improvements and future work.

Chapter 2

Related Work

Contents

Collision avoidance on unmanned aerial vehicles is a research topic of high interest. Using the camera as primary sensor requires Computer Vision methods to extract information, such as the environmental structure, from camera images. In Section 2.1 we explain Computer Vision principles and show how 3D information can be gathered from images. Subsequently, in Section 2.2 we introduce Simultaneous Localization and Mapping (SLAM) methods, which solve the problems of localization and map building at the same time. In Section 2.3 we describe different 3D map representations and discuss benefits and drawbacks of the individual approaches. Finally, we introduce current methods for collision avoidance on unmanned aerial vehicles in Section 2.4.

2.1 Basics of 3D Reconstruction

Computer Vision is an emerging scientific field including methods to acquire, process, and analyze digital camera images. The goal is to "teach a computer to see", i.e. to gather certain information from images. Cameras and acquired imagery are therefore central components in this field. To comprehend how information can be extracted from images it is necessary to understand how cameras work. The function principle of a

camera can be explained with a simple model, called the *pinhole camera model*.

The pinhole camera model. In a mathematical sense a camera is a mapping between the 3D world and a 2D image [21]. Three-dimensional points are mapped onto a two-dimensional plane by central projection. The center of projection is termed *camera center* and the plane is named *image plane*. Assume the setup in Figure 2.1, with the camera center $\boldsymbol{C}$ at the origin of a world coordinate system. The camera's viewing direction is called *principal axis* (sometimes *optical axis*) and goes along the positive z-axis. The image plane lies normal to the principal axis at a distance f, called *focal length*, from the camera center. Points in 3D space are mapped onto the image plane through a central projection to the camera center. Specifically, a world point $\boldsymbol{X}$ is mapped to image point $\boldsymbol{x}$, where a line from $\boldsymbol{X}$ to $\boldsymbol{C}$ intersects the image plane.

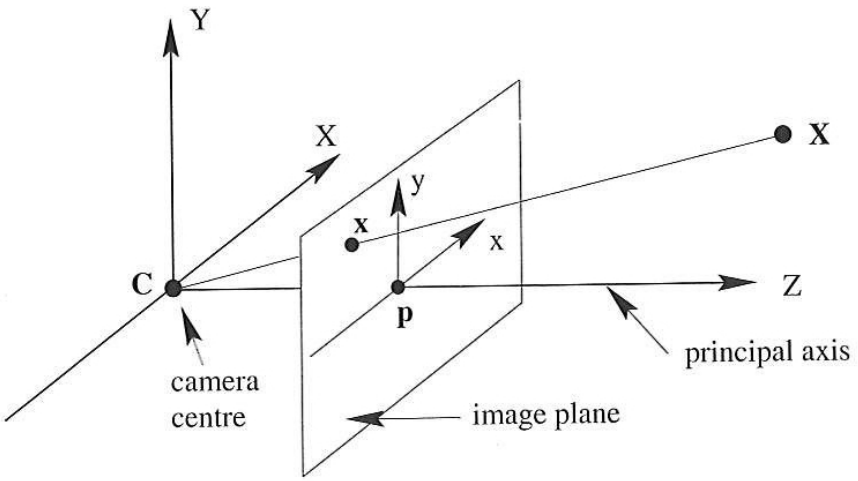


Figure 2.1: The pinhole camera model. World point $\boldsymbol{X}$ is mapped to image point $\boldsymbol{x}$ on the image plane through a central projection to camera center $\boldsymbol{C}$. Image taken from [21].

The camera maps a 3D point $\boldsymbol{X} = (X, Y, Z)^T$ to the point $(\frac{fX}{Z}, \frac{fY}{Z}, f)^T$ on the image plane. Since all points are mapped onto the image plane, i.e. having $z = f$, it is convenient to define a separate 2D coordinate system on this plane, called *image coordinate system*. The simplest method is to omit the z-coordinate, which automatically defines the 2D origin at principal point p, as shown in Figure 2.1. In general, the image coordinate origin is defined with a certain offset from this point, called *principal point offset*. The mapping from 3D space to 2D image is then defined as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} \frac{fX}{Z} + p_x \\ \frac{fY}{Z} + p_y \end{bmatrix} \tag{2.1}$$

where p_x and p_y denote the principal point offset in the image coordinate system. The parameters f, p_x, and p_y that describe this camera can be combined into a matrix

$$\boldsymbol{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{2.2}$$

which is termed *camera calibration matrix*. The matrix $\boldsymbol{K}$ allows a compact representation of the mapping from 3D to 2D according to

$$\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{I}|\boldsymbol{0}] \cdot \boldsymbol{X}, \tag{2.3}$$

assuming homogeneous coordinates. The camera calibration matrix can be extended from three to five parameters, which is beneficial when modeling real-world cameras. f is divided into f_x and f_y to model non-quadratic pixels [2] and the skew parameter s is introduced to account for non-perpendicular x- and y-axis. This leads to the general form of the camera calibration matrix

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix} \tag{2.4}$$

with 5 parameters. Those parameters in $\boldsymbol{K}$ are called the *intrinsic camera parameters*. In general, a camera can be translated and rotated with six degrees of freedom (6DOF) in the world coordinate system. Three degrees of freedom account for the camera's position and three for its orientation respectively. Position and orientation together determine the *pose* of a camera. The pose can be specified mathematically by a 3×3 rotation matrix $\boldsymbol{R}$ and a translation vector $\boldsymbol{t}$, which constitute the *extrinsic camera parameters*. Intrinsic and extrinsic parameters can be combined to form the 3×4 camera projection matrix

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]. \tag{2.5}$$

The mapping from world point $\boldsymbol{X}$ to image point $\boldsymbol{x}$ is fully specified by $\boldsymbol{P}$, according to

$$\boldsymbol{x} = \boldsymbol{P} \cdot \boldsymbol{X}. \tag{2.6}$$

Throughout this work we will represent a camera by its corresponding projection matrix $\boldsymbol{P}$.

When a point is mapped from 3D space to a 2D image, information regarding the point's depth is lost. However, the depth can be reconstructed if there exist two or more images showing the same point from different views. The according camera views must be known in terms of their camera projection matrices. This technique of depth reconstruction is known as *triangulation* or *multi-view reconstruction*.

Reconstruction from 2 cameras. Assume two cameras in a world coordinate system with known projection matrices $\boldsymbol{P}$ and $\boldsymbol{P'}$. A world point $\boldsymbol{X}$ is mapped to image point $\boldsymbol{x}$ by the first camera and $\boldsymbol{x'}$ by the second camera respectively. The image point $\boldsymbol{x}$ restricts the three-dimensional position of $\boldsymbol{X}$ to a ray from camera center $\boldsymbol{C}$ through $\boldsymbol{x}$. Another ray is defined by the second camera center $\boldsymbol{C'}$ and image point $\boldsymbol{x'}$ accordingly. The point $\boldsymbol{X}$ can be reconstructed in 3D space at the intersection point of both rays, as shown in Figure 2.2.

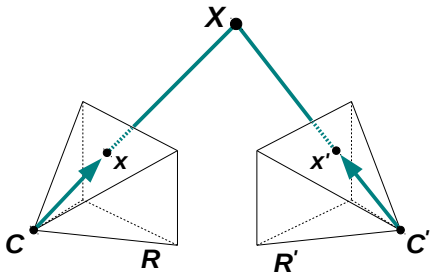


Figure 2.2: Triangulation principle. Both $\boldsymbol{x}$ and $\boldsymbol{x'}$ define a ray from the according camera center to restrict the position of $\boldsymbol{X}$. The intersection of two rays determines the 3D position of $\boldsymbol{X}$ and reconstructs depth information. Image modified from [46].

In practice, one has to face the problem of imperfectly measured image points $\boldsymbol{x}$ and $\boldsymbol{x'}$. Therefore, two rays will be skew and not intersect in a point in 3D space generally. In this case an approximate solution $\hat{\boldsymbol{X}}$ can be obtained in a least squares manner. This requires the definition and minimization of a suitable cost function. A simple solution is to reconstruct $\hat{\boldsymbol{X}}$ as the point closest to both rays, as exemplified in Figure 2.3. Better results are generally obtained by minimizing the *reprojection error*. It can be obtained by

projecting the approximated point $\hat{\boldsymbol{X}}$ back into both images at $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}'$. The reprojection error is then defined as $d(\boldsymbol{x}, \hat{\boldsymbol{x}})^2 + d(\boldsymbol{x}', \hat{\boldsymbol{x}}')^2$, where $d(*, *)$ denotes the Euclidean distance between two image points [21].

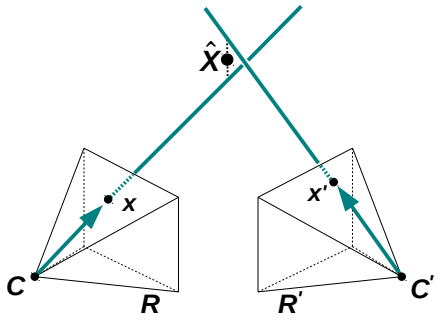
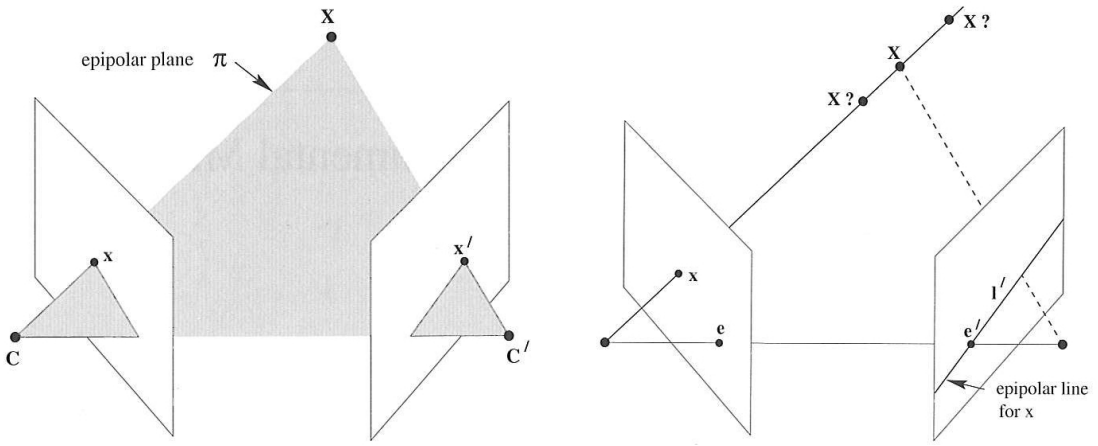


Figure 2.3: Triangulation in practice. The point $\hat{\boldsymbol{X}}$ minimizes the distance to both rays and constitutes an approximate reconstruction result. Image modified from [46].

When using approximation methods, the extension of reconstruction to more than two views is straightforward [21]. Reconstruction from more than two views can improve results considerably. In case of two views, the triangulation angle, i.e. the angle between back-projected rays, is a good indicator for the expected reconstruction quality. Small triangulation angles generally yield poor results but depth uncertainty decreases significantly with larger triangulation angles [44].

The great challenge in 3D reconstruction is to find image correspondences $\boldsymbol{x} \leftrightarrow \boldsymbol{x}'$, where both image points result from the *same point* $\boldsymbol{X}$ in 3D space. While correspondence identification among images is a simple task for humans it is still a challenging problem for computer algorithms. Finding correspondences is furthermore a computationally expensive problem. Generally, for a given image point $\boldsymbol{x}$ it is necessary to search over the entire corresponding image for point $\boldsymbol{x}'$. This search space can be reduced significantly if the relative geometry between both cameras is known, which is called *epipolar geometry*. The epipolar geometry describes the relation between two camera views. It is independent of the captured scene and only depends on the intrinsics and the relative pose between both cameras. Importantly, it allows to derive constraints where corresponding image points $\boldsymbol{x}$ and $\boldsymbol{x}'$ can be found, as depicted in Figure 2.4.

The point $\boldsymbol{X}$ is mapped to $\boldsymbol{x}$ from the first and $\boldsymbol{x}'$ from the second camera. World point $\boldsymbol{X}$, both image points and both camera centers are coplanar and lie on a plane π.



(a) (b)

Figure 2.4: (a) The two cameras are indicated by their camera centers $\boldsymbol{C}$ and $\boldsymbol{C}'$ and image planes. The 3D world point $\boldsymbol{X}$ and its image points $\boldsymbol{x}$ and $\boldsymbol{x}'$ lie on a common plane π. (b) Image point $\boldsymbol{x}$ back-projects to a ray through $\boldsymbol{C}$ and $\boldsymbol{x}$. This ray maps to the epipolar line $\boldsymbol{l}'$ in the second image. Images taken from [21].

Assuming that only image point $\boldsymbol{x}$ is known, it can be seen that the corresponding point $\boldsymbol{x}'$ has to lie on a line $\boldsymbol{l}'$. This line is the image of the back-projected ray from the first view and is called *epipolar line* of $\boldsymbol{x}$. Thus, the epipolar geometry results in a map $\boldsymbol{x} \mapsto \boldsymbol{l}'$ and the actual correspondence point $\boldsymbol{x}'$ is restricted to lie on epipolar line $\boldsymbol{l}'$.

The point to line mapping can be formulated mathematically with the *fundamental matrix* $\boldsymbol{F}$. This 3×3 matrix has 7DOF and encapsulates the relative geometry between both views. Corresponding image points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are restricted to point pairs that satisfy the equation

$$\boldsymbol{x} \cdot \boldsymbol{F} \cdot \boldsymbol{x}' = \boldsymbol{0}, \tag{2.7}$$

assuming homogeneous image coordinates. The fundamental matrix can be obtained uniquely from known camera projection matrices $\boldsymbol{P}$ and $\boldsymbol{P}'$. Conversely, $\boldsymbol{P}$ and $\boldsymbol{P}'$ cannot be determined uniquely from $\boldsymbol{F}$. This is because $\boldsymbol{F}$ only comprises information regarding the *relative* pose between cameras (lacking any absolute pose information) and there exists an overall scale ambiguity.

A known relative pose between two cameras can be ensured, if two physical cameras in a fixed setup are used, which capture images simultaneously. Such a rigid camera configuration is called a *stereo camera setup* or *stereo rig*. The fixed relative pose and known intrinsics uniquely determine the fundamental matrix $\boldsymbol{F}$, which in turn simplifies the correspondence search. A special case of a stereo configuration is the *canonical* stereo setup. In this setup both cameras are arranged side-by-side having parallel principal axes, as shown in Figure 2.5. The distance between both camera centers is called *baseline*. In a canonical stereo setup the epipolar lines run horizontally and hence allow efficient and robust correspondence search. Many current stereo reconstruction algorithms assume this configuration [19]. Next, we will investigate how reconstruction can be achieved if only one camera is present.

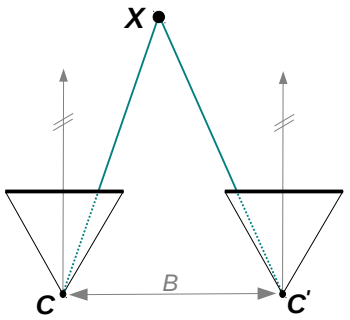


Figure 2.5: Top view of a canonical stereo setup. Both cameras are arranged side-by-side and have aligned orientations. The distance between camera centers is called baseline B. This setup restricts the correspondence search space to horizontal lines.

Reconstruction from a single camera. Typical stereo reconstruction requires the presence of two physical cameras in a fixed setup. In cases where only one camera is present, one can apply the same methods to images that were captured at different points in time. However, reasonable results can only be achieved if image pairs fulfill the following requirements:

1. The camera has to move between capturing the first and second image.
2. The motion must be in a way such that images exhibit sufficient parallax and overlap.
3. The scene must remain mostly static during capturing.

Since a motion of the single camera is required, this approach is also known as *Structure-from-Motion*. In contrast to a stereo setup, the relative pose between two views of the

same camera is unknown. To reconstruct 3D points it is therefore necessary to estimate the unknown relative pose between the views. It has been shown that a known camera geometry, represented as fundamental matrix $\boldsymbol{F}$, can be exploited to restrict the correspondence search space. Reversely, it is possible to estimate $\boldsymbol{F}$ from 8 or more known correspondences $\{\boldsymbol{x}_i \leftrightarrow \boldsymbol{x}'_i\}$ using the normalized 8-Point algorithm [20]. If the intrinsic camera parameters are known, the problem is simplified and only 5 correspondences are required. This can be accomplished robustly with the Five-Point algorithm [38]. As previously explained, the relative pose between cameras ($\boldsymbol{R}$ and $\boldsymbol{t}$) can be obtained from $\boldsymbol{F}$ up to an overall scale ambiguity. To get a metrically correct reconstruction, it is necessary to determine the missing scale factor as well as the absolute pose of one camera additionally.

2.2 Simultaneous Localization and Mapping

Systems for autonomous robot navigation in unknown environments must solve two main tasks, namely *localization* and *mapping*. Localization is the problem of estimating a robot's 6DOF pose with respect to a world coordinate system. Mapping denotes the task of perceiving the environment structure from sensor readings and model information in an appropriate representation. Both tasks are highly dependent on each other and are usually solved simultaneously using SLAM (Simultaneous Localization and Mapping) methods.

SLAM methods build a map in an unknown environment while keeping track of the current robot position relative to this map. Methods that utilize a camera as input sensor are called visual SLAM (VSLAM) methods. Early VSLAM systems used filtering techniques based on an Extended Kalman Filter (EKF) or particle filter respectively. Davison et al. [13] utilized an EKF to update probability distributions of both pose estimates and visual landmarks at frame rate. However, the EKF assumes motion and observation model to exhibit Gaussian noise, which is often not the case in practice. Particle filters, as used in [37] and [40], use a set of independent hypotheses that are weighted by their likelihood. All filtering methods enforce tracking and mapping to be thoroughly linked since pose estimates and landmarks are updated together at each frame. This is computationally expensive and the high redundancy between subsequent frames cannot be exploited. Motivated by this observation a novel approach to visual SLAM, called Parallel Tracking and Mapping (PTAM), was introduced in 2007.

Parallel Tracking and Mapping. PTAM is a real-time VSLAM method introduced by Klein and Murray [31]. It decouples the tasks of tracking camera pose and mapping the environment into two separate threads. The key observation is that subsequent video frames contain highly redundant information, thus using every single frame for mapping is not beneficial. Instead, only a few useful frames, called *keyframes*, are utilized for map building. Pose estimation on the other hand is performed at full frame rate in a separate thread. This thread tracks known map points among frames and derives the 6DOF camera pose from it.

PTAM's tracking thread requires an initial map, which is built at startup and needs user interaction. The user has to press a button, then smoothly translate the camera and perform a second key-press. This initiates the system to capture the first two keyframes. Image correspondences are obtained by tracking features between first and second key-press and an initial map of 3D points is triangulated. After map initialization the tracking thread constantly tries to find map points in the current frame. In each frame, FAST corner points [43] are detected at four levels of an image pyramid and 8x8 pixel patches are extracted. The 3D map points are projected into the current image, based on the last pose estimate and a decaying velocity model. Image patches that are stored together with each map point are warped and scaled according to the projection. Such a patch is compared against spatially close patches extracted from the current frame in terms of zero-mean SSD scores. If a correspondence is found within a fixed search radius of its expected position, the map point is considered successfully tracked. A camera pose update is then estimated from the set of tracked points in a two-stage coarse-to-fine procedure. Tracking quality is constantly monitored by observing the ratio of feature observations that have been matched successfully. Two threshold values allow to classify tracking quality to be either "good", "poor" or "lost". Whenever tracking is considered "lost", no pose estimates can be generated and a recovery routine is initiated.

PTAM's mapping thread extends and maintains the initial map. New 3D points are generated only after a new keyframe has been added. A keyframe is added if

- tracking quality is "good",
- the last keyframe has been added at least 20 frames ago, and
- the current camera position has a minimum distance to the nearest map point.

This leads to high quality keyframes and new map points are generated through triangulation with previously added keyframes. During times where no keyframe is processed the map is refined using local and global bundle adjustment [48]. PTAM achieves outstanding results for small AR environments and is considered a state-of-the-art VSLAM method. In the upcoming section we investigate how 3D environments can be modeled properly.

2.3 Map Representations

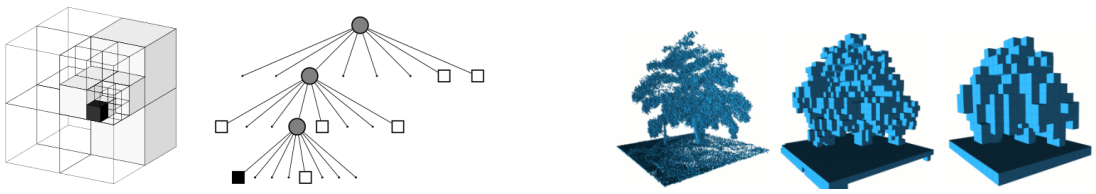
The term *mapping* is often used as synonym for obtaining three-dimensional information from sensor readings. However, mapping also comprises the modeling of 3D information in an appropriate representation. The simplest representation is a three-dimensional point cloud, i.e. an unordered set of 3D points. Each point results from a distance measure between a sensor to a certain object point. A major drawback of a point cloud representation is that no information regarding free or unknown space is stored in the map. Moreover, incorporation of noisy sensor data leads to inexact maps so point clouds should mainly be used with high precision sensors. Memory consumption of point clouds increase linear with the number of measurements and the simple representation is not well-suited for further processing. Therefore, representations other than point clouds are often beneficial.

Surface representations, such as Digital Surface Models (DSMs) [45] or Elevation Maps [22], constitute another way to model the environment. The idea is to model environmental structures as a continuous surface instead of a three-dimensional point set. Such a representation is often better suited for further processing, compared to a point cloud that exhibits gaps between individual points. Surfaces can be created by merging range images or applying meshing techniques to a given point cloud. However, models with a continuous surface such as simple DSMs can not deal with complex structures as bridges or tunnels. More flexible representations as multi-level surface maps [47] exist that allow more general surface-based modeling. Surface maps have like point clouds the drawback that free and unknown space are not explicitly modeled. Moreover, noisy sensor data often requires computationally expensive optimization steps to achieve decent results.

Occupancy maps [14, 52] are a probabilistic representation where the environment

is modeled as volumetric blocks. The 3D space is subdivided into fixed size volume elements, called *voxels* or *cells*. Each cell is assigned a probability that an obstacle is present in this space. The cell probabilities are obtained from integrating measurements into the map. Discretization and the probabilistic nature of the map makes this representation robust to noisy sensor data. Voxel elements can be classified according their particular cell probability to be "free" or "occupied". A third state "unknown" can also be used for cells where no information regarding obstacles is present, e.g. undiscovered space. Distinguishing free, occupied, and unknown space gives a more complete representation of the environment and the probabilistic nature allows to deal with sensor noise. Therefore, we utilize an occupancy map framework called OctoMap [52] in our system for environment mapping.

OctoMap. OctoMap[2] is an efficient implementation of a 3D occupancy mapping framework. It is based on octrees [35, 50], a tree-based data structure for 3D space subdivision. An octree is a tree with up to 8 child nodes representing the subdivision of a volume into 8 subvolumes. This recursive subdivision ends at a given minimum voxel size, known as *resolution*. The tree structure implicitly gives a multi-resolution representation since cutting the tree at any depth leads to a coarser subdivision of the space, as depicted in Figure 2.6.



(a) (b)

Figure 2.6: (a) An octree storing free (shaded white) and occupied (black) cells. The volumetric and the corresponding tree representation are illustrated. (b) Coarser resolutions can easily be obtained in OctoMap by cutting the tree at any depth level. Images taken from [52].

OctoMap was used in our work because we found it to be the best suited map representation. The probabilistic map structure allows to deal with noisy sensor readings, which

[2]Downloaded from `http://octomap.github.io`

is expected when using monocular vision. Modeling of occupied, free, and unknown space gives a complete representation of the 3D environment. Maps are built incrementally without the need to specify the map extent in advance. Furthermore, OctoMap allows to work on different resolution levels, which can be exploited to meet speed or accuracy requirements. The efficient implementation produces maps that are compact in size and allow fast map operations. Details on the usage of OctoMap in our system are described in Section 5.1.

2.4 Current Research on UAV Collision Avoidance

A lot of research is focusing on collision avoidance and autonomous navigation of unmanned aerial vehicles. We will now give a short summary of recent methods and discuss merits and drawbacks of the individual approaches. Collision avoidance methods can be divided based on the primary sensor into *non-vision based* and *vision based* techniques.

Non-vision Based Techniques. Such techniques use other than visual sensors to detect obstacles in the environment. Bouabdallah et al. [10] used four ultrasound sensors on a quad-rotor helicopter for obstacle detection. Distance measurements are conducted to a flight controller that aims to stay distant to obstacles or performs evasive maneuvers when obstacles are already close. Ultrasound sensors give robust distance measures but they constitute an additional payload for the UAV. Bachrach et al. [3] utilized a laser range finder (LRF) for obstacle detection and fused measurements using an extended Kalman filter. Other recent works [5, 15, 18] also demonstrated the feasibility of laser sensors on UAVs. Like ultrasound sensors, LRFs represent an additional payload and hence reduce flight times and influence flight dynamics. Moreover, laser sensors are expensive and have a small field of view compared to cameras.

Vision Based Techniques. Techniques based on vision use one or more cameras as primary sensor for obstacle detection. Meier et al. [36] utilized a stereo camera setup to detect obstacles and used artificial markers for robust localization. Additionally, they exploit IMU sensor data to refine pose estimates relative to the markers. The fast on-board computer allows to perform obstacle detection and localization on the UAV. Stereo is also used by Hrabar in [26] where dense reconstructions are obtained with 25 Hz using stereo on chip (STOC) technology. 3D points are

integrated into a three-dimensional occupancy map. A global path planning algorithm based on probabilistic roadmaps [29] is used to navigate in cluttered environments. The problem of localization and limited load capacity is not tackled since the approach is evaluated on an air vehicle simulator. Methods based on stereo vision yield fast and accurate reconstruction results but require two physical cameras on the UAV.

Another popular approach is to use RGB-D cameras on aerial vehicles such as the Microsoft KINECT ™ sensor. Bachrach et al. [4] presented a real-time SLAM system on a micro aerial vehicle based on KINECT. A visual odometry algorithm runs on-board to obtain real-time pose estimates while costly tasks as global pose correction or loop closure detection run decoupled on a ground station computer. They also integrate measurements into a three-dimensional occupancy map for a robust and compact environment representation. The KINECT sensor is cheap, lightweight, and provides real-time depth images. Unfortunately, it uses an IR structured light pattern for depth perception that restricts the sensor to indoor usage.

Recent work also includes methods for collision avoidance that use a conventional monocular camera. Call et al. [11] extracted Harris corner points and tracked them over several frames to triangulate a map of sparse three-dimensional points. They used an agglomerative clustering algorithm to group points to obstacles. Point clusters that are spatially close to the UAV are avoided using a reactive sliding mode control law. A similar approach was published by Lee et al. in [32]. They extracted multi-scale oriented patches (MOPS) at Harris corner points and combined them with SIFT [34] feature points. Points of both kind are triangulated to a sparse 3D point cloud. Their method presumes that MOPS patches are mainly extracted at the obstacle boundaries while SIFT features appear mostly at the obstacle's interior. Volumetric obstacle information is gathered by merging both types of 3D points following this assumption. Both mentioned methods assume accurate reconstruction of sparse 3D points but do not address problems as outlier handling, small triangulation angles or lacking texture.

In contrast to map building methods, Ross et al. [42] proposed a system for autonomous MAV navigation, which does not extract 3D information from images. Instead, they use a state-of-the-art imitation learning strategy and train a reactive flight controller directly on 2D images. The controller learns a strategy to mimic

actions that a human pilot would take in the same situation. Training data (sample images) and supervision (human pilot's commands) are collected during an iterative training phase. The method allows autonomous MAV navigation through woods. However, the control strategy is limited to environments that have been learned during training. Bills et al. [8] also navigated a small MAV through indoor scenes without map building. They first classify which type of environment is currently observed, e.g. corridor, by extracting and analyzing Hough lines [16] in the images. Based on the classified environment, the desired flight direction is obtained from perspective cues. In corridors for example, they estimate the vanishing point of lines and navigate towards this point to progress through the corridor. Their method is restricted to a few scene categories and can not deal with unforeseen obstacles on the path.

In this Chapter has been shown how 3D information can be obtained from camera images. Furthermore, we have investigated the related topics of localization and mapping and discussed different kinds of 3D map representations. Finally, we have summarized current methods that address collision avoidance and autonomous UAV operations. In the following chapters we describe our proposed collision avoidance system.

Chapter 3

Localization

Contents

Localization is the task of estimating the aerial vehicle's 6DOF pose, which is required for two reasons. First, relative poses between multiple views are needed to reconstruct 3D information from images. Second, the pose is essential to navigate with a map that collects the gathered 3D information.

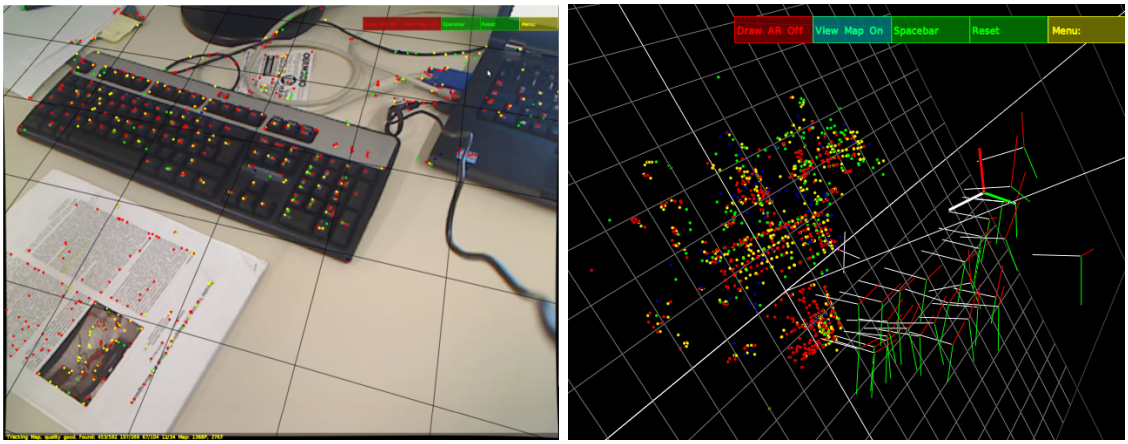
An intuitive method is to exploit the global positioning system (GPS) for this purpose, but its feasibility is severely limited. The GPS signal can only be utilized outdoors and even there it might be disturbed by high buildings or walls. Moreover, GPS can only determine the position but not the orientation of the UAV. Another major drawback is the limited accuracy, which is in the range of a few meters. This makes it hard to navigate in cluttered environments and impossible to estimate accurate camera poses for multi-view reconstruction. Thus, more accurate localization methods are required.

Visual Simultaneous Localization and Mapping (VSLAM) methods can build a map from visual landmarks and estimate the camera pose with respect to that map. The PTAM system is a state-of-the-art implementation and has proven to deliver accurate 6DOF pose estimates in real-time. Its internal map of registered 3D points also contains valuable information of the environmental structure. This motivates the strategy to obtain localization from PTAM. In Section 3.1 we describe how this can be achieved, but also illustrate limitations and problems with this method. Therefore, we subsequently

present an alternative localization method that overcomes many of those problems. It is based on the Inertial Measurement Unit (IMU) of the drone and is described in Section 3.2.

3.1 Localization from PTAM

Parallel Tracking and Mapping, shortened PTAM, is currently one of the most promising VSLAM systems. It was mainly designed for augmented reality (AR) applications. A typical AR scenario includes a moving handheld camera that faces a small scene. As the camera moves, PTAM triangulates three-dimensional points and integrates them into its internal map. Simultaneously, it estimates the camera pose at frame rate based on visible map points. The pose estimates and the environmental structure are then utilized to overlay the video stream with virtual objects in typical AR applications. Figure 3.1 shows the system operating in a desktop scene.



(a) (b)

Figure 3.1: PTAM in a typical AR environment. (a) The colored dots show map points that are currently tracked. The grid depicts the dominant plane, which could be used to draw virtual objects on the table. (b) The three-dimensional point map is built incrementally from triangulation between spatially distributed images, called *keyframes*. The coordinate frames depict the camera poses of those keyframes. Images taken from [28].

The two main tasks of PTAM – tracking and mapping – highly depend on each other. The tracking thread estimates the current camera pose by identifying and tracking

known map points among camera images. Simultaneously, the mapping thread tries to extend and refine this map based on *known camera poses*. This can be seen as a "chicken-and-egg problem" and requires to define a starting point. For this purpose PTAM employs an initialization procedure at startup to create an initial map of 3D points.

Standard Initialization. The standard initialization procedure of PTAM works the following way. The user faces the initial scene with the camera and presses a key. Then the user translates (and maybe rotates) the camera smoothly and again presses the same key. This results in the first two keyframes to be added. The translation between the keyframes is necessary to provide a sufficient parallax between both images. Corresponding points in the image pair are determined by tracking salient points from the first to the second keyframe during the smooth motion. Successfully tracked points constitute a set of image correspondences $\{\boldsymbol{x}_i \leftrightarrow \boldsymbol{x}_i'\}$ that are triangulated to an initial map of 3D points.

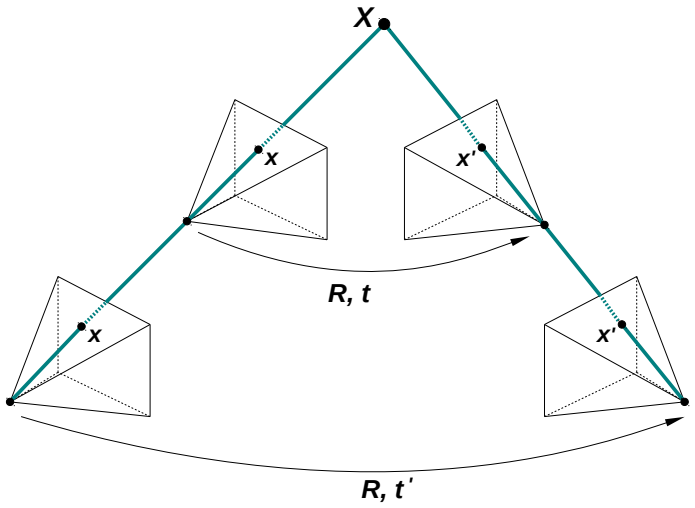


Figure 3.2: Standard PTAM initialization. The user translates and possibly rotates the camera between the first two keyframes according to $\boldsymbol{R}, \boldsymbol{t}$. Without prior knowledge it is not possible to determine this initial motion with correct scale. A motion according to $\boldsymbol{R}, \boldsymbol{t}$ is not distinguishable from any scaled version $\boldsymbol{R}, \boldsymbol{t}'$. This leads to arbitrarily scaled map points and pose estimates.

The problem with this initialization procedure is that initial map and pose estimates can only be determined up to scale. The arbitrary scale is a consequence of the unknown initial camera motion employed by the user. Without additional knowledge it is not possible to obtain this scale factor from a set of image correspondences, as shown

in Figure 3.2. While a correct (metric) scale might not be of importance in certain AR applications, it is of vital importance in our system. Therefore, the initialization procedure had to be modified to obtain the missing scale factor.

Metric Initialization. The correct scale can be assessed if the relative motion between first and second keyframe is known in terms of $\boldsymbol{R}$ and $\boldsymbol{t}$. For this purpose we put the UAV on a board that contains two marked positions. The relative pose between the two positions is known and incorporated into the modified initialization procedure. Furthermore, we replaced the tracking from first to second keyframe with an epipolar search strategy given the known relative pose. This leads to more robust results and does not require a smooth and trackable initial motion. The resulting initial map is metrically scaled and so are the pose estimates from the tracking thread. Figure 3.3 illustrates the metric initialization procedure on the unmanned aerial vehicle.

(a) (b)

Figure 3.3: Metric PTAM initialization. (a) The UAV is placed on the left marked position from where the first keyframe is captured. (b) The second keyframe is captured from the right position, which is also marked on the board. The relative motion between the keyframes is known in terms of $\boldsymbol{R}$ and $\boldsymbol{t}$, which results in a metrically scaled initial map.

Fixing the relative motion between first and second keyframe leads to an initial map with metric scale. Consequently, the pose estimates from the tracking thread exhibit correct scale, which is required in our context. An estimated pose describes position and orientation of the on-board camera with respect to a world coordinate system. The world coordinate system is chosen to be aligned with the first keyframe

pose. Mathematically spoken, the extrinsic camera parameters corresponding to the first keyframe are $\boldsymbol{t} = [0, 0, 0]^T$ and $\boldsymbol{R} = \boldsymbol{I}$, where $\boldsymbol{I}$ denotes the 3×3 identity matrix. To localize an unmanned aerial vehicle using PTAM it is necessary to employ the initial procedure on the ground. Once the initial map is created, the UAV is ready to take off and the pose of its on-board camera can be estimated at frame rate. Figure 3.4 shows a scenario where PTAM is employed on the unmanned aerial vehicle.

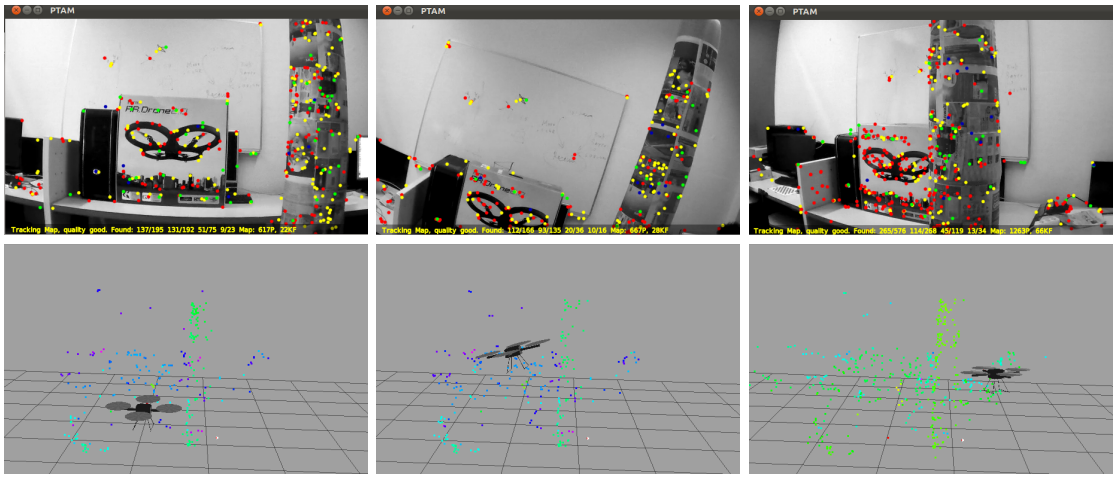


Figure 3.4: PTAM used for localization of an unmanned aerial vehicle. Top row: 3 views from the on-board camera after take off. Tracked points are illustrated as colored dots in the image. Bottom row: The corresponding pose estimates of the UAV relative to the three-dimensional map points.

Limitations. The example above illustrates a scenario where localization with PTAM works considerably well. The shown scene is static, limited in depth, and provides enough texture for successful image feature tracking. Such conditions can be assumed in many augmented reality scenarios, for which PTAM was originally designed. However, those optimal preconditions cannot be assumed for UAV applications in general, as exemplified in Figure 3.5. Problems arise in several situations, especially in case of:

- **Large Scenes:** Scenes with a depth exceeding a few meters are difficult to handle. Small triangulation angles lead to inaccurate maps. Consequently, the pose estimates relative to this map are of low quality.
- **Dynamic Scenes and Occlusions:** PTAM incrementally builds a map of 3D points and implicitly assumes that the environment remains static. Major scene

changes or occlusions from moving objects violate this assumption.

- **Lack of Texture:** Image features can only be found and tracked in well-textured regions. Homogeneous areas (e.g. sky, untextured walls) provide no interest points for tracking.

- **Sharp Turns:** Sharp turns of the UAV during flight cause all tracked points to vanish from the field of view (FOV). Mostly, this leads to an inevitable loss of tracking and no pose estimates can be generated.

(a) (b)

Figure 3.5: Some situations showing the limited feasibility of PTAM on unmanned aerial vehicles. (a) Large scenes cannot be mapped accurately due to small triangulation angles. (b) Lack of texture or missing map points in the field of view cause PTAM to *lose localization*.

Conclusion. Those limiting scenarios are likely to happen with UAV applications, especially when operating outdoors. When PTAM cannot find enough points to track it *loses localization*. In this state it is not able to provide pose estimates at all. Although the map of PTAM is extendable it would require highly strategic flight planning to extend the map systematically without losing localization. The mentioned limitations are clearly not acceptable for a reliable collision avoidance system. Hence, we need a more robust way of estimating the UAV pose, which is described in the subsequent section. Nevertheless, our modular system design allows to utilize PTAM in scenarios where those limitations are admissible. Summarized, PTAM is not an essential component but a valuable extension to our overall system.

3.2 Localization using the Inertial Measurement Unit

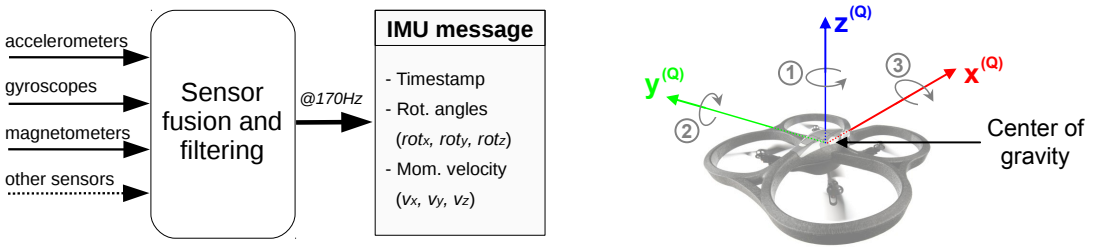
In the previous section has been shown that localization from visual landmarks is problematic in certain situations. When PTAM loses localization it cannot deliver any pose estimates until the system recovers (which is not guaranteed to happen). While lost, PTAM cannot perceive 3D information either, due to lacking information regarding the current motion. So the entire system would be "blind" whenever localization is lost. Therefore, we need an alternative method that can deal better with the aforementioned situations.

Unmanned aerial vehicles, such as quad-rotor helicopters, require certain internal sensors to ensure stable flight behavior. UAVs are therefore equipped with sensors including accelerometers, gyroscopes or magnetometers. The combination of those sensors constitutes the Inertial Measurement Unit (IMU). The IMU reports the vehicle's current state in terms of orientation, velocity, and gravitational forces to an internal controller, which requires this information to control the UAV's dynamics. Many platforms allow to intercept and utilize IMU data for external tasks. In our case, we exploit this data to estimate the 6DOF pose of the UAV and this section illustrates how this is achieved.

IMU Data Format. In this work we use the Parrot AR.Drone 2.0 quad-rotor helicopter[3] as unmanned aerial vehicle. It is equipped with 3 accelerometers, 3 gyroscopes, a magnetic compass, and 2 ultrasonic altitude sensors. Sensor data is fused and filtered in a black-box manner from a low-level process that runs on the UAV. This process delivers accurate estimates of the quad-rotor's *momentary orientation* and *momentary velocity*. Information is encapsulated into messages, called IMU messages, that we intercept and exploit for pose estimation. Those messages are received at a high rate of approximately 170 Hz. Each message consists of an accurate timestamp, three rotation angles rot_x, rot_y, rot_z, and 3 momentary velocity values v_x, v_y, v_z. Rotation angles and velocity values are with respect to a *quad-rotor coordinate system*, as illustrated in Figure 3.6.

The 3 rotation angles rot_x, rot_y, rot_z are given in radiant from the interval $[-\pi, \pi]$. Each angle specifies the amount of rotation around the corresponding axis. Together they determine the absolute orientation of the quad-rotor helicopter. This representation, known as Euler angles or Tait-Bryan angles [41], is ambiguous unless the order of

[3] http://ardrone2.parrot.com



(a) (b)

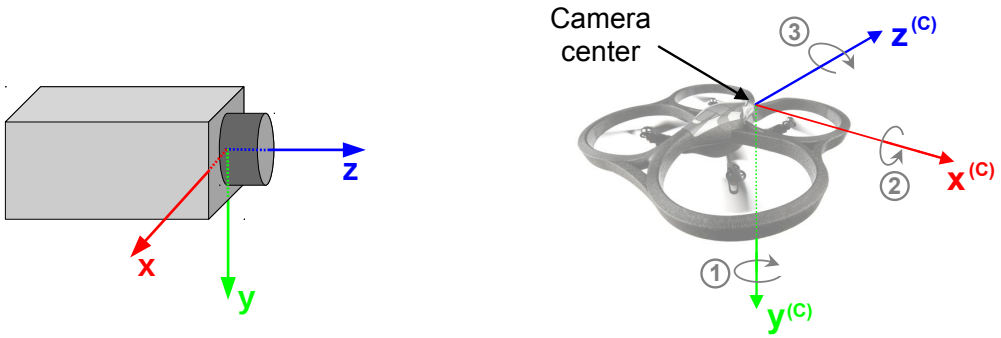
Figure 3.6: IMU messages and the quad-rotor coordinate system. (a) Data from multiple sensors is fused, filtered, and published as IMU message periodically. Each message contains an accurate timestamp as well as momentary rotation angles and velocity. (b) Rotation angles and velocity are with respect to a *quad-rotor coordinate system*, indicated by the index Q. The origin is located at the quad-rotor's center of gravity and the axes configuration (x: front, y: left, z: up) is common in aeronautics.

individual rotations is known. We assessed empirically that the correct rotation order is rot_z, followed by rot_y, followed by rot_x.

The momentary velocity values v_x, v_y, v_z describe the velocity components towards x-, y-, and z-axis respectively. They compose the momentary velocity vector $\boldsymbol{v}^{(Q)} = [v_x, v_y, v_z]^T$. In the next step we describe how we relate this information to the primary sensor for depth perception, the on-board camera, and how the fixed world coordinate system is defined.

Camera and World Coordinate System. The quad-rotor helicopter contains a rigid, forward-looking camera at its front side. Images from this camera will be used to extract 3D information, presuming accurate 6DOF pose estimates. We define a *camera coordinate system* with its origin at the camera center, as illustrated in Figure 3.7. The axes follow a common convention for cameras, where the z-axis is aligned with the optical axis, the y-axis faces downwards, and the x-axis goes to the right hand side.

Since the camera is rigidly mounted on the quad-rotor, there exists a fixed transform between camera and quad-rotor coordinate system. We choose the camera system to be



(a) (b)

Figure 3.7: The camera coordinate system. (a) We follow the common coordinate system convention for cameras, where the z-axis is aligned with the optical axis. (b) Applying this convention to the quad-rotor leads to the *camera coordinate system*, indicated by the index C. The order of individual rotations needs to be considered.

the dedicated object coordinate system for pose estimation. In other words, any position estimate will refer to the camera center rather than to the quad-rotor's center of gravity. This is beneficial since we can utilize camera poses for subsequent tasks as multi-view reconstruction without conversion. Therefore, we express information from IMU messages in terms of the camera coordinate system. The rotation angles with respect to the camera coordinate system can easily be obtained by switching the corresponding axes according to

$$\begin{aligned} rot_x^{(C)} &= -rot_y^{(Q)} \\ rot_y^{(C)} &= -rot_z^{(Q)} \\ rot_z^{(C)} &= rot_x^{(Q)}. \end{aligned} \tag{3.1}$$

Considering the camera coordinate system axes, the individual rotations have to be applied in the order $(y \rightarrow x \rightarrow z)$ to determine the correct camera orientation, as illustrated in Figure 3.7. In the same manner we express the momentary velocity vector in camera coordinates. For pure translational motion the velocity of the camera is equal to the velocity of the quad-rotor's center of gravity and the following equation holds true.

$$\boldsymbol{v}^{(C)} = \begin{bmatrix} v_x^{(C)} \\ v_y^{(C)} \\ v_z^{(C)} \end{bmatrix} = \begin{bmatrix} -v_y^{(Q)} \\ -v_z^{(Q)} \\ v_x^{(Q)} \end{bmatrix} \tag{3.2}$$

For rotational quad-rotor motion this equation is not correct due to the offset between camera center and center of gravity. Unfortunately, the exact offset distance is hard to determine and thus a small error is inevitable. Since the distance is in the range of a few centimeters, we approximate the camera's momentary velocity according to Equation 3.2.

Additionally, a fixed *world coordinate system* is required, which constitutes the reference frame for localization. This system can be defined arbitrarily. We follow the example of PTAM and fix the world coordinate system to the initial camera pose at system startup. World and camera coordinate system are illustrated in Figure 3.8.

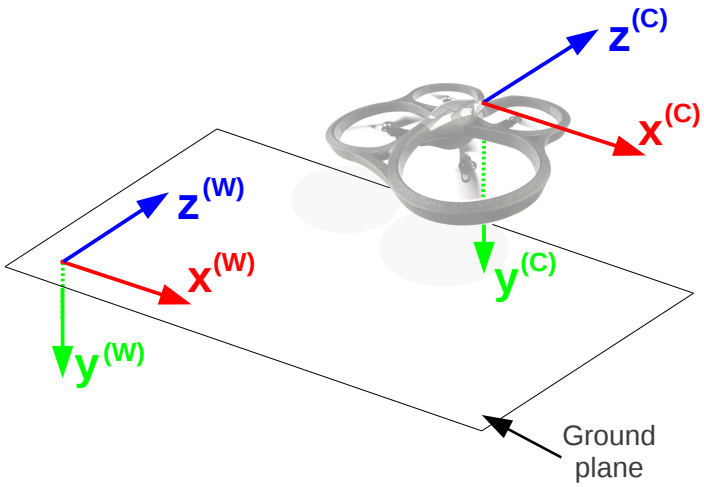


Figure 3.8: The world coordinate system, indicated by the index W, is fixed to the initial pose of the quad-rotor's camera. Therefore, the xz-plane of the world system constitutes the ground plane approximately. A pose estimate specifies the transformation between world and camera coordinate system.

Pose Estimation from IMU Data. So far we have defined a world and a camera coordinate system and receive rotation angles and momentary velocity from periodic IMU messages. The goal is to exploit this data and estimate orientation and position of the camera from it. Orientation can be determined directly from the rotation angles $rot_x^{(C)}$, $rot_y^{(C)}$, and $rot_z^{(C)}$. By definition, the initial camera

pose is aligned with the world coordinate system. Therefore, the initial rotation angles (before take-off) constitute a constant offset that must be subtracted from the current angles. Considering the correct order of rotation, the angles are converted to a 3×3 rotation matrix $\boldsymbol{R_{W,C}}$, which fully specifies the orientation of the camera.

Besides orientation, the camera position needs to be determined. IMU messages provide accurate estimates of the momentary velocity. The idea is to obtain the absolute position by integrating velocities over time. Since velocity values can only be estimated relative to the current orientation, we first have to convert from camera to world system. Given the velocity vector $\boldsymbol{v}^{(C)}$ in camera coordinates and current orientation $\boldsymbol{R_{W,C}}$, the velocity vector in world coordinates is calculated as

$$\boldsymbol{v}^{(W)} = \boldsymbol{R_{W,C}} \cdot \boldsymbol{v}^{(C)}. \tag{3.3}$$

Velocity is defined as the rate of change of an object's position over time. Conversely, the position can be determined by integration of velocity. This way, we obtain the camera's absolute position $\boldsymbol{x}^{(W)}$ as we integrate over the velocity $\boldsymbol{v}(t)^{(W)}$ over time. We approximate this integration by a sum of piecewise constant velocity values $\boldsymbol{v}_i^{(W)}$ multiplied by the corresponding duration Δt_i. The duration results from the timestamp difference between two consecutive IMU messages at times t_i and t_{i-1}. Thus, the absolute position $\boldsymbol{x}^{(W)}$ at time step N is calculated as

$$\boldsymbol{x}^{(W)} = \sum_{i=1}^{N} (\boldsymbol{v}_i^{(W)} \cdot \Delta t_i). \tag{3.4}$$

The absolute position $\boldsymbol{x}^{(W)}$ and the rotation matrix $\boldsymbol{R_{W,C}}$ fully specify the 6DOF pose of the camera with respect to the world coordinate system.

Conclusion. The proposed localization method utilizes IMU sensor data to determine the camera pose in a world coordinate system. In contrast to PTAM, this method is not dependent on the presence of visual features. Therefore, it can be applied in scenarios where visual SLAM methods are not applicable. The drawback of this method is that no global reference is present and the position estimate tends to drift. In other words, the position error accumulates over time due to the successive integration process. It is

therefore to expect that real and estimated position drift apart, making this approach inaccurate for large-scale localization. However, to detect and avoid *close* obstacles it is not necessary to achieve accurate localization over large distances. Unless we aim to utilize a global path planner, we are only interested whether obstacles are present inside of a small volume around the UAV. Within a small spatial region we expect the effect of drift to be negligible, which will be verified in our experiments. In the map building stage, described in Section 5, we will countervail the effects of drift by discarding parts of the map that are spatially distant from the UAV. This makes the IMU-based approach the preferred localization method in our collision avoidance system.

Chapter 4

3D Reconstruction

Contents

The main task of the collision avoidance system is to detect obstacles that are spatially close to the quad-rotor helicopter. This requires to obtain three-dimensional information of the environment. If PTAM is employed for localization, it would be possible to access its internal map of 3D points. However, since PTAM is only feasible in constrained scenarios (as described previously), we do not presume 3D information to be available at this point. Thus, we created our own method for efficient 3D reconstruction.

Extracting three-dimensional information from a single, moving camera requires to match images captured at different points in time. It has been discussed that reconstruction can only yield good results if such images exhibit an appropriate baseline. Otherwise, small triangulation angles cause high depth uncertainty in the reconstruction process. In order to promote high quality results it is necessary to select appropriate images from the video stream. Those images are called *key images*[4] and the selection process is described in Section 4.1 elaborately.

The actual 3D reconstruction process utilizes those key images. Whenever such an image is selected, a reconstruction step is triggered immediately. The reconstruction works on consecutive key image pairs, i.e. each key image and its predecessor are processed in a

[4]The idea of selecting a set of spatially distributed images is closely related to the keyframe concept of PTAM. In order to avoid confusion we use the term *key image* instead.

stereo fashion, as described in Section 4.2. The entire reconstruction process is illustrated in Figure 4.1.

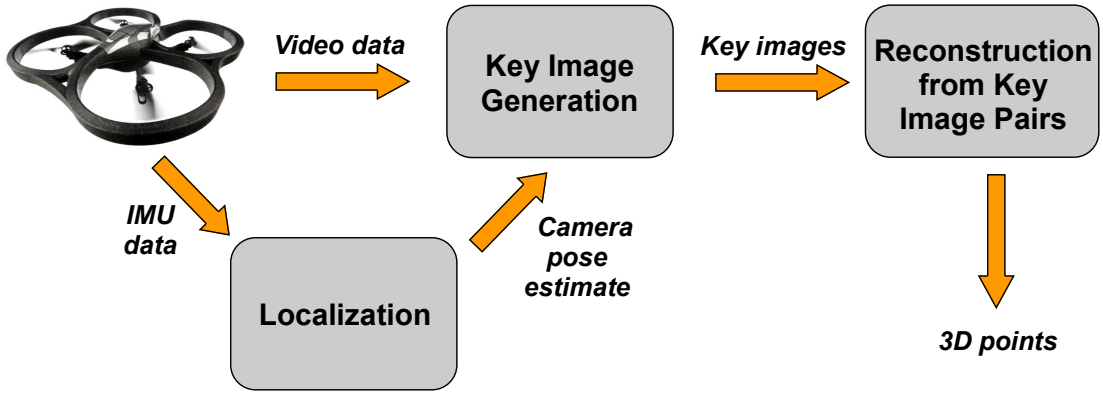


Figure 4.1: Overview of the 3D reconstruction process. The aerial vehicle supplies video and IMU data. The key image generation routine selects appropriate images based on the estimated camera pose. Subsequent image pairs are utilized to extract 3D information in a stereo fashion.

4.1 Key Image Generation

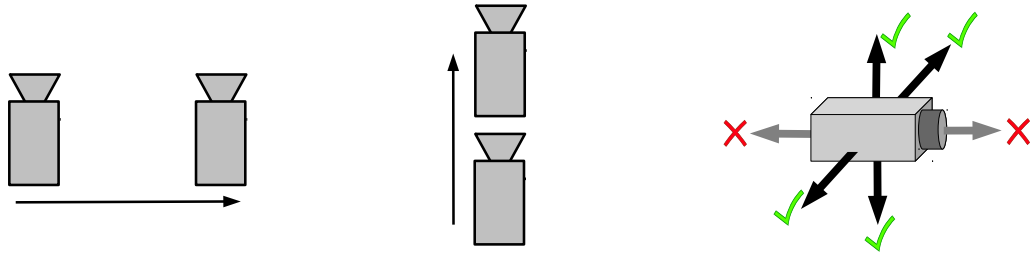
During operation of the unmanned aerial vehicle we employ a routine that constantly selects key images from the video stream. For the following explanation, the i-th selected key image is denoted as $\mathcal{I}_i$. Whenever the routine selects a new key image, it is immediately used for reconstruction together with its predecessor. This means, the pair $\{\mathcal{I}_{i-1}, \mathcal{I}_i\}$ is treated like a stereo image pair, from which 3D information is extracted. Accordingly, the successor key image $\mathcal{I}_{i+1}$ will form an image pair with $\mathcal{I}_i$ after its selection. This way, we successively extract 3D information on regular bases. The question is now: *When should we generate the next key image $\mathcal{I}_{i+1}$ given the current key image $\mathcal{I}_i$?*

Assume that $\mathcal{I}_i$ was captured from camera pose $\mathcal{P}_i = [\boldsymbol{R_i}|\boldsymbol{t_i}]$. For each camera frame the routine has to decide whether it becomes the next key image or not. This depends on $\mathcal{P}_i$ as well as on the current camera pose $\mathcal{P}_{cam} = [\boldsymbol{R}_{cam}|\boldsymbol{t}_{cam}]$. In particular, we are interested in *how far* and *in which direction* the camera has been translated from $\mathcal{P}_i$. Therefore, the translation vector $\boldsymbol{t}^{(\boldsymbol{W})} \in \mathbb{R}^3$ that describes this motion in world coordinates is determined. Since the motion direction is of interest, we express this translation with respect to the

camera coordinate system of $\mathcal{P}_i$, as

$$\boldsymbol{t^{(C)}} = \boldsymbol{R_i}^{-1} \cdot \boldsymbol{t^{(W)}}. \tag{4.1}$$

As already mentioned, good reconstruction results require an appropriate baseline between the images. Besides the translation distance, the motion direction needs to be considered, as shown in Figure 4.2. A pure forward motion (along the optical axis) for example does not lead to increased triangulation angles, whereas horizontal or vertical motion increases the baseline and promotes good reconstruction results.



(a) Horizontal motion (b) In-depth motion (c) Motion overview

Figure 4.2: (a) Horizontal camera motion promotes good triangulation angles whereas in-depth motion, as in (b), does not. (c) Summarized, left-, right-, up- or down-motion improve the baseline and forward- or backward-motion does not.

The translation vector $\boldsymbol{t^{(C)}} = [t_x^{(C)}, t_y^{(C)}, t_z^{(C)}]^T$ is used to assess if an appropriate baseline has been achieved. Following our coordinate system conventions, horizontal motion (right/left) is indicated by $t_x^{(C)}$, vertical motion (up/down) by $t_y^{(C)}$, and in-depth motion (forward/backward) by $t_z^{(C)}$. Since horizontal or vertical motion is desired, a distance threshold is set on the according components of the translation vector. If either $|t_x^{(C)}|$ or $|t_y^{(C)}|$ becomes greater than this threshold value, the current frame is selected as new key image $\mathcal{I}_{i+1}$. The according threshold is called *minimum baseline* and denoted as B_{min}. It constitutes an important parameter in our system and its value affects rate and quality of generated key images. A small value leads to fast key image generation, whereas a large value results in fewer images with greater baselines. Figure 4.3 summarizes the entire key image generation process. In the upcoming section we show how 3D information is obtained from pairs of key images.

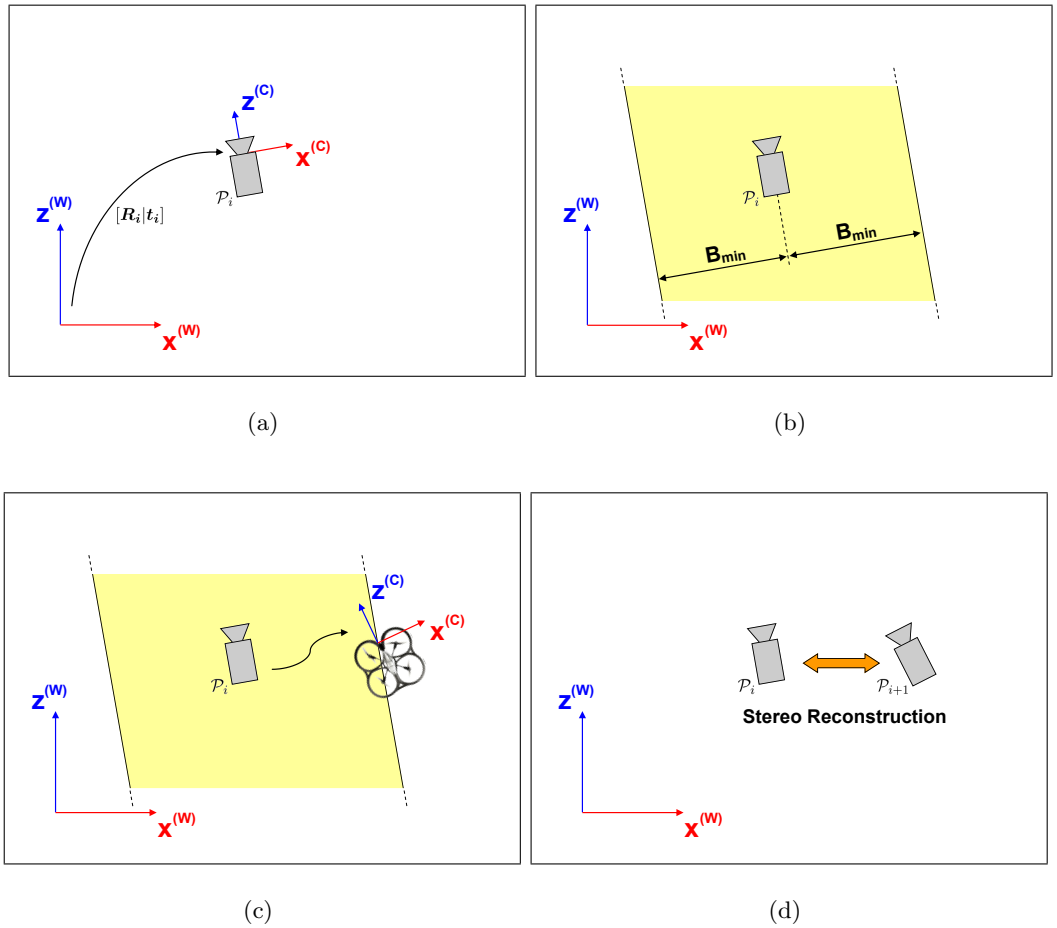


Figure 4.3: Key image generation process (top view). (a) Key image $\mathcal{I}_i$ has been captured from pose $\mathcal{P}_i = [\boldsymbol{R_i}|\boldsymbol{t_i}]$. (b) The minimum baseline B_{min} defines a region relative to this pose (yellow). No key image is generated as long as the camera remains within this region. (c) As soon as the horizontal (or vertical) translation exceeds B_{min} to either side, the next key image $\mathcal{I}_{i+1}$ is added. (d) The new key image $\mathcal{I}_{i+1}$ and its predecessor $\mathcal{I}_i$ form an image pair for stereo reconstruction.

4.2 Reconstruction from Key Image Pairs

Whenever a key image has been selected, we immediately utilize it for 3D reconstruction together with its predecessor. From the corresponding camera poses and the intrinsic camera parameters (which are assessed in a one-time calibration step), the camera projection matrices are determined. Given a pair of images with known camera matrices allows to apply sophisticated stereo reconstruction methods. There exist various algorithms for stereo reconstruction and a good overview is given on benchmark dataset

sites, such as KITTI[5] or Middlebury[6]. Some state-of-the-art algorithms have been assessed and tested for feasibility in our system.

Most proposed algorithms create *dense* reconstruction results that obtain a 3D point for each pixel in an image. Although this gives a more complete representation than triangulating individual correspondence points, the amount of data is hardly manageable in a real-time system. While this issue could be countervailed by a down-scaling of the input images, the major problem is that most real-time capable methods presume a canonical stereo setup. The Efficient Large-Scale Stereo Algorithm [19] creates reconstruction results from canonical stereo images very efficiently. Small deviations to a canonical stereo configuration can be compensated by an according image rectification step [33]. However, it has been discovered that rectification is not applicable if the epipolar geometry differs significantly from such a canonical configuration. We also investigated stereo algorithms that can handle arbitrary stereo configurations, such as the planesweep algorithm [12]. Unfortunately, the results of this algorithm are quite noisy and require a total-variation (TV-L1) based smoothing [49]. Such global smoothing operations are computationally expensive and therefore not feasible in a real-time system. For those reasons, we created our own reconstruction method and specifically designed it to meet the requirements in our system.

Our 3D reconstruction method consists of the following steps: First, local image features are extracted from both key images and correspondence points are obtained. We exploit those correspondences to further refine the relative pose between the cameras using the Five-Point algorithm [38]. This improves the results of the next step, which is the triangulation of 3D points. The reconstructed 3D points are *densified* to get a more complete representation of the environment. For this purpose, we employ a meshing algorithm on the sparse point cloud, called Ball Pivoting Algorithm [7]. The individual steps for reconstruction are now elaborated in more detail.

4.2.1 Feature Extraction and Correspondence Matching

Stereo reconstruction requires to find corresponding points in the image pair. For this reason, local image features need to be extracted from both images. Feature extraction includes *detection* and *description* of interest points. Detection is the task of finding

[5] `http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo`
[6] `http://vision.middlebury.edu/stereo/data/`

a set of image points at recognizable locations (e.g. at corner points). Each point is then described based on its local appearance, utilizing local information such as color, shape or texture. It depends on the used feature descriptor which kind of information is actually incorporated. As a result, each point is described as individual feature vector.

Feature Extraction. There exist numerous methods for image feature extraction. The feasibility of a certain method is always related to the task that needs to be solved. PTAM for example detects FAST corner points [43] and describes each point as 8×8 pixel patch (i.e. as vector of intensity values). This method is highly efficient and allows PTAM to extract and track feature points at frame rate. Efficiency is important for our reconstruction method as well, although the performance requirements are weaker. This is because we neither employ tracking nor need to perform reconstruction at full frame rate, since key images come at a much lower rate. It enables us to use a less efficient but highly discriminative feature descriptor, namely SIFT [34].

SIFT (Scale Invariant Feature Transform) is known to be an extraordinarily robust feature descriptor. It works on Difference of Gaussian (DoG) interest points, which are detected at different scales of an image pyramid. Around the interest point location, the image gradient is calculated in terms of magnitude and orientation. Gradient orientations are weighted with a Gaussian function and accumulated into multiple gradient orientation histograms. The histograms describe the distribution of edge directions in a certain sub-region. After normalization, all histogram values are combined into a 128-dimensional vector representing the interest point, as illustrated in Figure 4.4. Obtained feature vectors are very robust to noise or illumination effects and can handle significant changes of viewpoint and scale. The highly discriminative description allows to achieve outstanding matching results.

Extracted SIFT features are highly discriminative and robust, but the extraction process is computationally expensive. However, significant speedups can be achieved if the parallel processing power of a Graphics Processing Unit (GPU) is exploited. An efficient GPU implementation for SIFT feature extraction exists, which is called SiftGPU [51]. GPU processing is usually not possible on-board of an unmanned aerial vehicle, but in case of a distributed system design, this task can be carried out on an appropriate ground station. In our system we employ SiftGPU on a ground station computer to speed up the feature

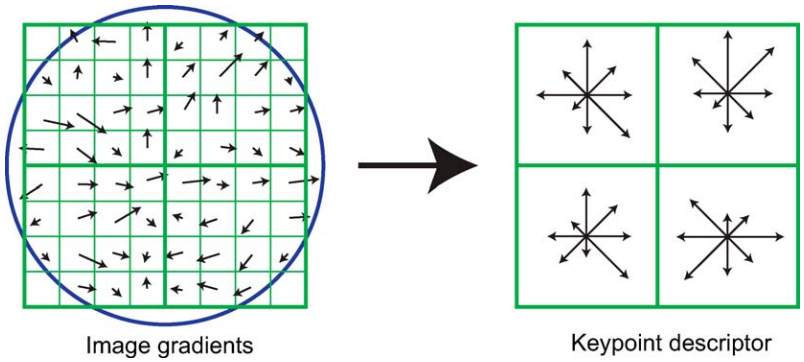


Figure 4.4: The SIFT image descriptor. Gradient magnitude and orientation are computed in a region around the interest point, as shown on the left. The blue circle indicates the Gaussian smoothing to avoid boundary effects. The gradients are accumulated into orientation histograms that summarize the contents over subregions, as shown on the right. This Figure shows a 2×2 array of histograms, each having 8 directional bins. In its standard configuration, SIFT creates a 4×4 histogram array with 8 bins, leading to a 128-dimensional feature vector. Image taken from [34].

extraction process. A CPU variant is also implemented, such that a GPU-capable machine is not a necessary requirement. We evaluate the effective speedup that can been achieved with SiftGPU in Section 6.4. Interest points that have been detected in a pair of key images are shown in Figure 4.5. The next step is to match features between images to find corresponding image points.

Figure 4.5: SIFT feature extraction on key images. The colored circles show the image locations where DoG interest points have been detected. The size of the circle indicates the scale, based on the DoG image pyramid level. A 128-dimensional feature vector is extracted at each interest point location.

Correspondence Matching. After extracting SIFT feature vectors from both images corresponding points need to be identified. Therefore, each descriptor vector $\{\boldsymbol{v}_i\}_{i=1\ldots N}$ in

the first image is compared against all descriptor vectors $\{\boldsymbol{v}_j\}_{j=1\ldots M}$ in the second image. The Euclidean distance between two feature vectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ is calculated as

$$d_{i,j} = \|\boldsymbol{v}_i - \boldsymbol{v}_j\|_2. \tag{4.2}$$

The discriminative SIFT descriptor is expected to achieve small distances if $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ show the same point in both images, and large distances for non-corresponding points. The simplest strategy is to match each point in an image to the point that gives the smallest distance $d_{i,j}$ in the other image. However, this leads to inevitable mismatches despite the highly discriminative feature descriptor, as shown in Figure 4.6.

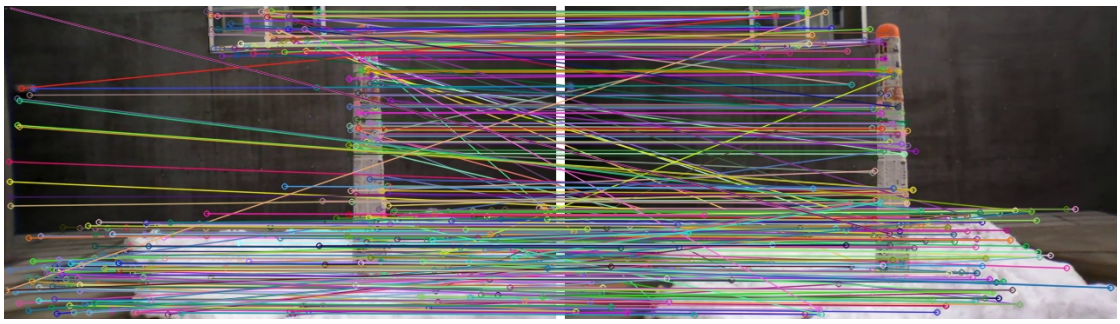


Figure 4.6: Simple feature matching. Matching each image point to the point with the smallest feature distance leads to numerous mismatches.

In order to reduce mismatches we use a more sophisticated method, namely a *ratio matcher with backmatching*. This means that a potential match is confirmed only if

- the ratio between smallest and second smallest distance is below a threshold T_{ratio} and
- the same potential match is found in both directions (from first to second image and vice versa).

This restriction rejects ambiguous matches and mainly correct correspondences remain. We empirically assessed that best results are achieved with a distance ratio threshold T_{ratio} of 0.7. Further rejection of mismatches *could* be achieved by a spatial match verification, for example by verifying if epipolar constraints are met. However, the proposed matcher achieved results with very few mismatches making further verification unnecessary, as illustrated in Figure 4.7. Feature matching has a complexity of $\mathcal{O}(N \cdot M)$, where N and M denote the number of extracted features in each particular key image.

This task can also be performed on a GPU in a parallelized manner. Again, we evaluate which speedup can be achieved with a GPU implementation in our experiments.

Figure 4.7: Using a ratio matcher with backmatching leads to high quality correspondences. We used a distance ratio threshold T_{ratio} of 0.7.

4.2.2 Epipolar Geometry Refinement

At this point a pair of key images, $\mathcal{I}_1$ and $\mathcal{I}_2$, and a set of high quality image correspondences $\{\boldsymbol{x}_i \leftrightarrow \boldsymbol{x}_i'\}$ have been established. Furthermore, the camera projection matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ according to the images are known. Basically, this is all the information required to triangulate 3D points. However, the accuracy of the estimated camera poses in $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ depends on the quality of localization. As explained in the previous chapter, localization might be inaccurate due to drift effects. Thus, we exploit the image correspondences $\{\boldsymbol{x}_i \leftrightarrow \boldsymbol{x}_i'\}$ to refine the epipolar geometry leading to more accurate 3D reconstruction results.

Epipolar geometry refinement is based on the Five-Point algorithm [38]. This algorithm estimates the relative pose $[\boldsymbol{R}|\boldsymbol{t}]$ between two cameras from 5 image correspondences, assuming known camera intrinsics. If more than 5 correspondences are given, a robust estimate on a consensus set is found using RANSAC [17]. As previously explained, the relative pose can only be estimated up to scale from image correspondences. To obtain the missing scale factor we exploit the pose estimates in $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. Precisely, the scale factor is chosen such that the Euclidean distance between the camera centers is preserved. This way, we determine a *refined epipolar geometry* based on image correspondences, which is usually more accurate than the epipolar geometry specified by $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. Taking absolute pose information into account, we effectively determine

a refined second camera $\boldsymbol{P_2}'$ adopting a relative pose $[\boldsymbol{R}|\boldsymbol{t}]$ to the first camera $\boldsymbol{P}_1$. Triangulation is employed from cameras $\boldsymbol{P}_1$ and $\boldsymbol{P_2}'$, as illustrated in Figure 4.8. We show that this refinement step leads to improved reconstruction quality in Section 6.3.

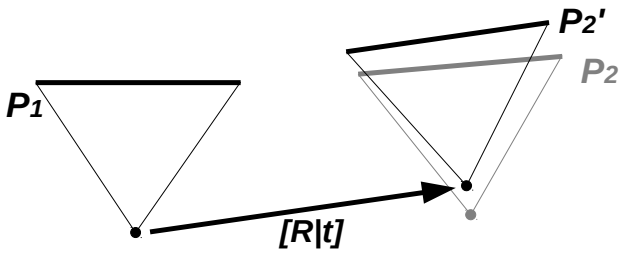


Figure 4.8: Epipolar geometry refinement. The cameras specified by $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are given and constitute a prior estimate on the epipolar geometry. Using the Five-Point algorithm, we obtain the relative pose $[\boldsymbol{R}|\boldsymbol{t}]$ between the cameras, where the scale is determined from the priors. Triangulation is based on camera $\boldsymbol{P}_1$ and the refined camera $\boldsymbol{P_2}'$ adopting the assessed relative pose.

4.2.3 Triangulation and Point Cloud Densification

After the refinement step we triangulate [21] correspondences to 3D points from the camera pair specified by $\boldsymbol{P}_1$ and $\boldsymbol{P_2}'$. We assure good triangulation results by demanding a minimum triangulation angle of 2° (otherwise the point is rejected). This threshold constitutes a good tradeoff between accurate reconstruction and sufficient scene depth. Finally, we ensure that each reconstructed point lies in front of both cameras and can actually be projected into both images. This way, we obtain a high quality 3D point cloud in world coordinates, as illustrated in Figure 4.9.

A sparse point cloud is not a good representation to detect obstacles because the lack of surface information between individual points is problematic. Each reconstructed point origins from a region in the image where an interest point has been detected. Consequently, no 3D points are obtained in regions where interest points are lacking (e.g. homogeneous areas). Further processing is required to close gaps and get a more meaningful environment model. For this reason, we employ a triangular meshing algorithm on the sparse points. This mesh is then used to interpolate additional points and establish a *densified* point cloud.

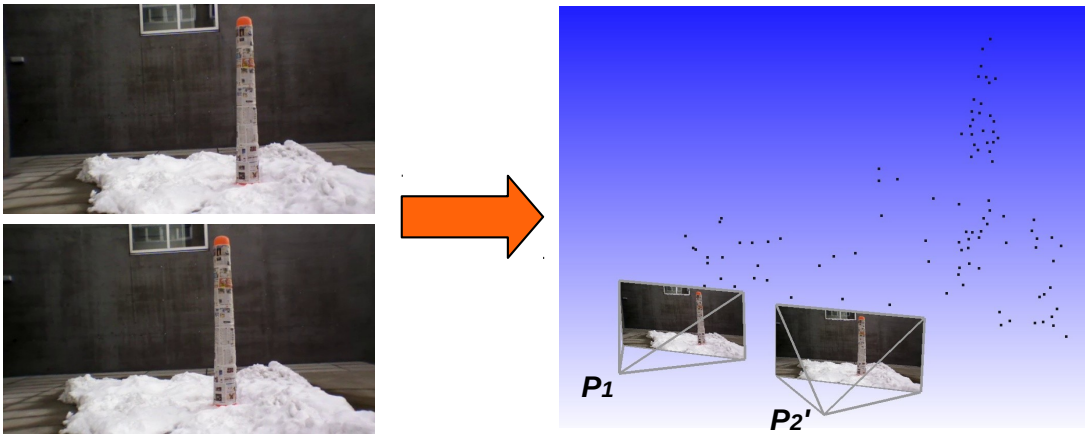


Figure 4.9: Sparse 3D point reconstruction. Sparse points are obtained from triangulation of image correspondences based on cameras $\boldsymbol{P}_1$ and $\boldsymbol{P_2}'$. The few points cannot be used directly for reliable obstacle detection and are therefore densified.

The first step of the densification process is to create a triangular mesh upon the sparse 3D points. Mathematically, a triangular mesh is represented as a set of vertices $\mathcal{V}$ and a set of faces $\mathcal{F}$. Each vertex $v \in \mathcal{V}$ represents a 3-dimensional point in $\mathbb{R}^3$ and each face $f \in \mathcal{F}$ defines a triangle spanned by three vertices. The process of finding a set of faces $\mathcal{F}$ for a given vertex set $\mathcal{V}$ is called *Meshing* or *Surface Reconstruction*. There exist manifold algorithms to establish a set of faces from a given point cloud. Individual surface reconstruction algorithms differ in computation time, size and shape of the created triangles, as well as completeness and connectivity of the resulting mesh.

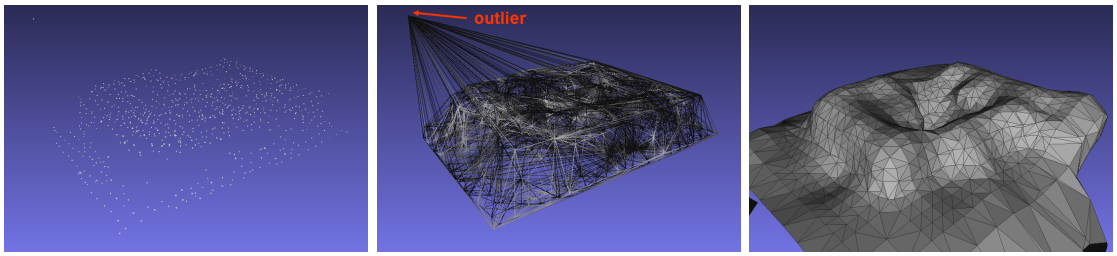
Surface Reconstruction Algorithms. A widely used surface reconstruction algorithm is the 3D Delaunay Triangulation[7] described in [9]. It is an extension of the 2D Delaunay Triangulation, where 2D points are connected to triangles such that no point is inside the circumcircle of any other triangle. Adapted to 3D, it partitions the convex hull of points into tetrahedra with the condition that no point is inside the circumsphere of any other tetrahedron. However, this approach requires to carve away numerous triangles that are not part of the surface and is significantly influenced by noisy outliers [24, 39].

Another popular method is the Poisson Surface Reconstruction algorithm [30], which creates a watertight, triangular surface mesh. First, the algorithm requires to estimate

[7]Obtaining a triangular mesh from a point set is sometimes referred to as 3D *triangulation*. It should not be confused with the process of reconstructing a 3D world point from two views.

the normal vectors of all given points. Then, it finds the indicator function $\chi : \mathbb{R}^3 \to \mathbb{R}$, whose gradient best approximates the normal vector field, by solving a Laplacian equation iteratively. Finally, the surface is extracted as an appropriate isosurface of χ. The obtained surface mesh is watertight and looks visually appealing. However, one fully connected mesh is often not suitable to model multiple, non-connected objects. Furthermore, finding normal vectors and the indicator function are computationally expensive tasks.

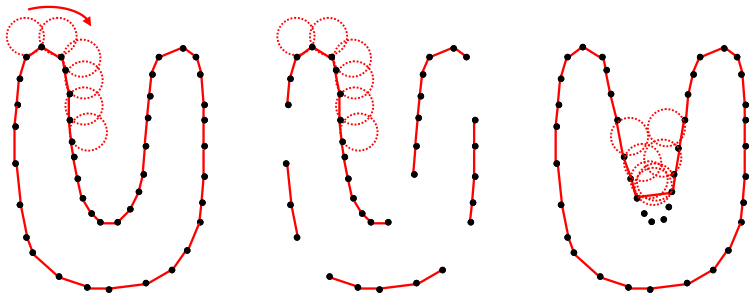
A meshing algorithm for our aims should connect gaps at regions with high point density to emphasize the evidence of an obstacle there. Contrarily, it should not create faces at regions with low point density (*i.e. allow holes in the mesh*) and ignore isolated outliers (*i.e. allow non-connected vertices*). Moreover, the algorithm needs to perform efficiently, which can be quite challenging in the three-dimensional domain. After investigating different meshing algorithms we found the *Ball Pivoting Algorithm* [7] to best meet those requirements.



(a) Given point cloud (b) 3D Delaunay (c) Poisson

Figure 4.10: Comparison of common surface reconstruction methods. (b) The 3D Delaunay Triangulation is sensitive to outliers and creates many inner triangles that need to be carved away. (c) The Poisson Surface Reconstruction achieves visually appealing results, but the watertight surface is too restrictive in many scenarios.

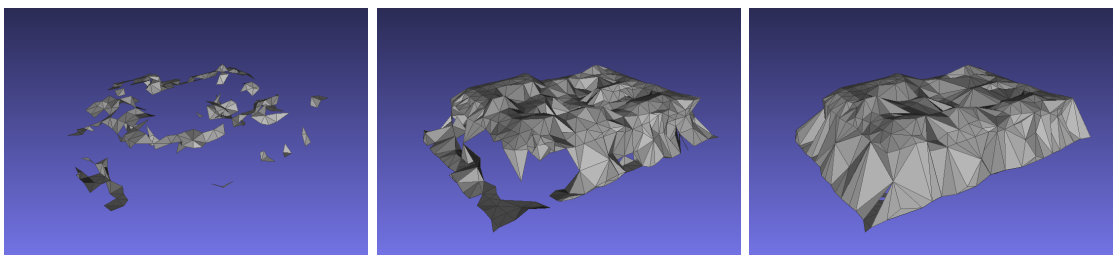
The Ball Pivoting Algorithm. The idea of the Ball Pivoting Algorithm (BPA) is fairly simple. Three vertex points form a triangle if a ball with a user-specified radius ρ is able to touch them simultaneously. Starting at a seed triangle, the ball pivots around an edge while still touching both edge endpoints. When the ball touches another point, a triangle is created by adding two edges to this point. The algorithm converges after all edges have been traversed. The idea is illustrated in 2D in Figure 4.11.



(a) (b) (c)

Figure 4.11: The Ball Pivoting Algorithm in 2D. (a) A ball with fixed radius ρ pivots from point to point and connects them with edges (red lines). (b) At regions with reduced sampling density, some edges will not be created and holes remain. (c) Some points might not be reached where the curvature of the manifold is larger than $1/\rho$ resulting in missing details. Images modified from [7].

The Ball Pivoting Algorithm is very efficient in terms of execution time and memory consumption. It exhibits linear time performance $\mathcal{O}(n)$ with respect to the number of input points. A great advantage is that only one parameter needs to be specified, which is the ball radius ρ. A large ball radius results in more and larger triangles whereas a small radius leads to smaller triangles and more gaps in the piecewise connected mesh, as shown in Figure 4.12. Thus, the ball radius can be adjusted to create faces only at regions where points are sufficiently dense while ignoring isolated points.

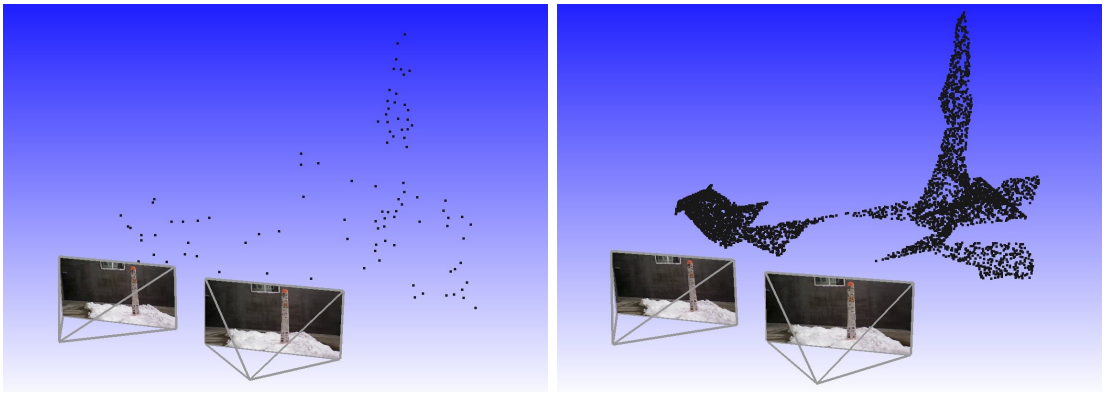


(a) $\rho = 0.25m$ (b) $\rho = 0.50m$ (c) $\rho = 0.75m$

Figure 4.12: The same point cloud is meshed with the Ball Pivoting Algorithm, which only creates triangles between spatially close points. The distance threshold is implicitly determined by the chosen ball radius ρ.

Basically, the faces introduced with the BPA connect 3D points with a small spatial distance compared to the fixed ball radius ρ. Assuming obstacles to be sufficiently textured[8], it is likely that the created faces connect points at the obstacle's surface and (partially) reconstruct the obstacle shape. 3D points that lie very isolated from other points will not become part of any triangular face. Such points are mostly outliers and should remain unconnected to the created surface elements. Summarized, the BPA creates triangular surface elements at obstacle-evident regions and rejects isolated outliers at the same time.

For obstacle detection it is not necessary to obtain geometrically correct surface reconstructions of obstacles. The purpose of creating faces is to emphasize the evidence of obstacles at certain regions and close gaps between points there. Although a surface mesh is a more compact representation than a point cloud, it is beneficial to convert the created faces into points, which is achieved by dense surface-sampling of the triangular faces. The reason for this sampling step is that points are easier to fuse and integrate into a common map than surface elements. Surface-sampling of faces leads to a *semi-dense point cloud*. This means, the resulting point cloud exhibits high point density at some regions whereas no points are present in other regions, as depicted in Figure 4.13.



(a) Sparse point cloud (b) Semi-dense point cloud

Figure 4.13: Visualization of the point cloud densification. The semi-dense point cloud contains more valuable information and is beneficial for obstacle detection.

[8]Every algorithm based on feature extraction and matching is only feasible in scenes, where enough discriminative features can be extracted.

Summary. In this chapter we have shown how 3D information is obtained from the video stream. Starting from an initial key image, we wait until the camera has moved by a sufficient horizontal or vertical distance B_{min} and select the next key image. We extract discriminative SIFT features from both key images and establish high quality correspondences using a ratio matcher with backmatching. The correspondences are used to further refine the epipolar geometry between cameras from which sparse points are triangulated. We interpolate between spatially close points using the Ball Pivoting Algorithm and obtain a semi-dense point cloud. The next chapter will show how semi-dense point clouds are fused into a common map, which is used for collision avoidance.

Chapter 5

Mapping and Reactive Control

Contents

The reconstruction process establishes semi-dense point clouds from pairs of camera images. The next step is to fuse those point clouds by incorporating them into one common map. This map models the environment and constitutes the basis for collision avoidance. As a map representation we use a 3D occupancy grid to model free, occupied, and unknown space in a probabilistic fashion. In Section 5.1 we give a detailed explanation about how the map is gradually built and maintained throughout the operation of the unmanned aerial vehicle.

The occupancy map combines the environmental information that is obtained throughout a flight. Obstacle information can be retrieved from this map via *map queries* over a clearly defined interface. Querying the map and triggering appropriate actions is accomplished by a dedicated *reactive controller* process. This process needs to be closely related to the main task of the UAV (e.g. power pylon inspection) in order to initiate adequate *reactive actions*. We implemented two types of reactive controllers, which are explained in Section 5.2. The first one enables autonomous operations of the aerial vehicle and the second one assists on human-operated flights. An overview of the mapping and control tasks and its relation to the reconstruction process is given in Figure 5.1.

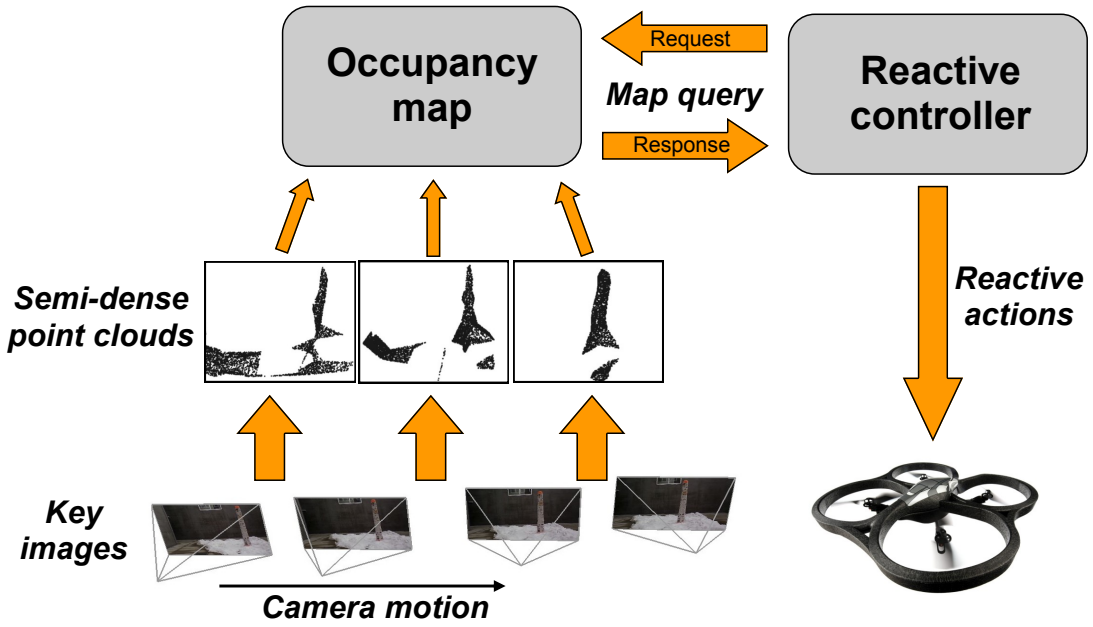


Figure 5.1: Mapping and control overview. The reconstruction process generates semi-dense point clouds from key image pairs. All point clouds are incorporated into a common occupancy map constituting the basis for collision avoidance. An independent controller process retrieves information from this map and sends appropriate reactive actions to the UAV.

5.1 Probabilistic Occupancy Grid Mapping

To provide a reliable map for collision avoidance, we integrate 3D points into a three-dimensional occupancy map. For this purpose we utilize the OctoMap [52] framework, which has been described in Chapter 2. OctoMap has proven to be a fast and flexible framework for occupancy grid mapping. It subdivides the 3D space recursively into discrete cells and each cell n is assigned an *occupancy probability* $P(n)$. The individual cell probabilities result from integrating 3D points into the map, which is subsequently described in detail.

Map Integration. The occupancy map is built of cubic cells with a fixed size. The cube edge length is known as the *resolution* of the map. Each cell has a probability value of being occupied that results from integration of points. When a 3D point is integrated, the occupancy probability of the cell that contains this point increases. Mathematically, the probability $P(n|z_{1:t})$ of cell n being occupied, given the measurements $z_{1:t}$ and prior

probabilities $P(n)$ is

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \cdot \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \cdot \frac{P(n)}{1 - P(n)}\right]^{-1}. \tag{5.1}$$

Calculations can be simplified by using *log odds* notation, which is an alternative way of representing probabilities. Probabilities P can be converted to log odds L and vice versa, according to

$$L = ln\left(\frac{P}{1 - P}\right). \tag{5.2}$$

Using log odds notation and assuming uniform prior probabilities of $P(n) = 0.5$ simplifies Equation 5.1 to

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t). \tag{5.3}$$

The term $L(n|z_t)$ represents the update of cell n according to the current measurement z_t. Its value results from the *inverse sensor model* that is used. We use a *beam-based inverse sensor model*, where a ray from the sensor origin to a reconstructed 3D point is inserted. Since each point is obtained from two camera views, we insert one ray from each camera center to the 3D point. All cells along a ray are determined efficiently using a 3D version of the Bresenham algorithm [1] and updated as follows:

$$L(n|z_t) = \begin{cases} l_{occ}, & \text{for the cell that contains the 3D point} \\ l_{free}, & \text{for all other cells along the ray} \end{cases} \tag{5.4}$$

The beam-based model increases the occupancy probability of the cell containing the 3D point and decreases the probabilities of all other cells along the ray, as illustrated in Figure 5.2. The two parameters l_{occ} and l_{free} determine the impact of a measurement and have been fixed to values of $l_{occ} = 0.4$ and $l_{free} = -0.4$. After a certain cell has been updated multiple times, its probability value might become very high (occupied) or very low (free). In order for the map to remain updateable (e.g. due to changing environment) an upper and lower bound of cell probabilities is defined. This *clamping update policy* [53] avoids map overconfidence by ensuring that cell confidences $L(n|z_{1:t})$ remain in the interval $[l_{min}, l_{max}]$. We set those values to $l_{min} = -1.4$ and $l_{max} = 1.4$ respectively.

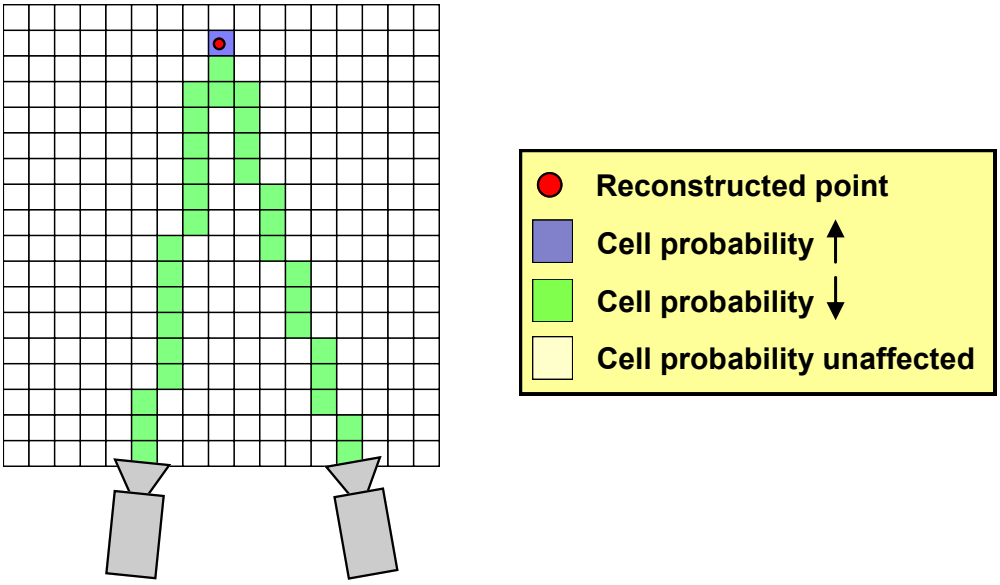
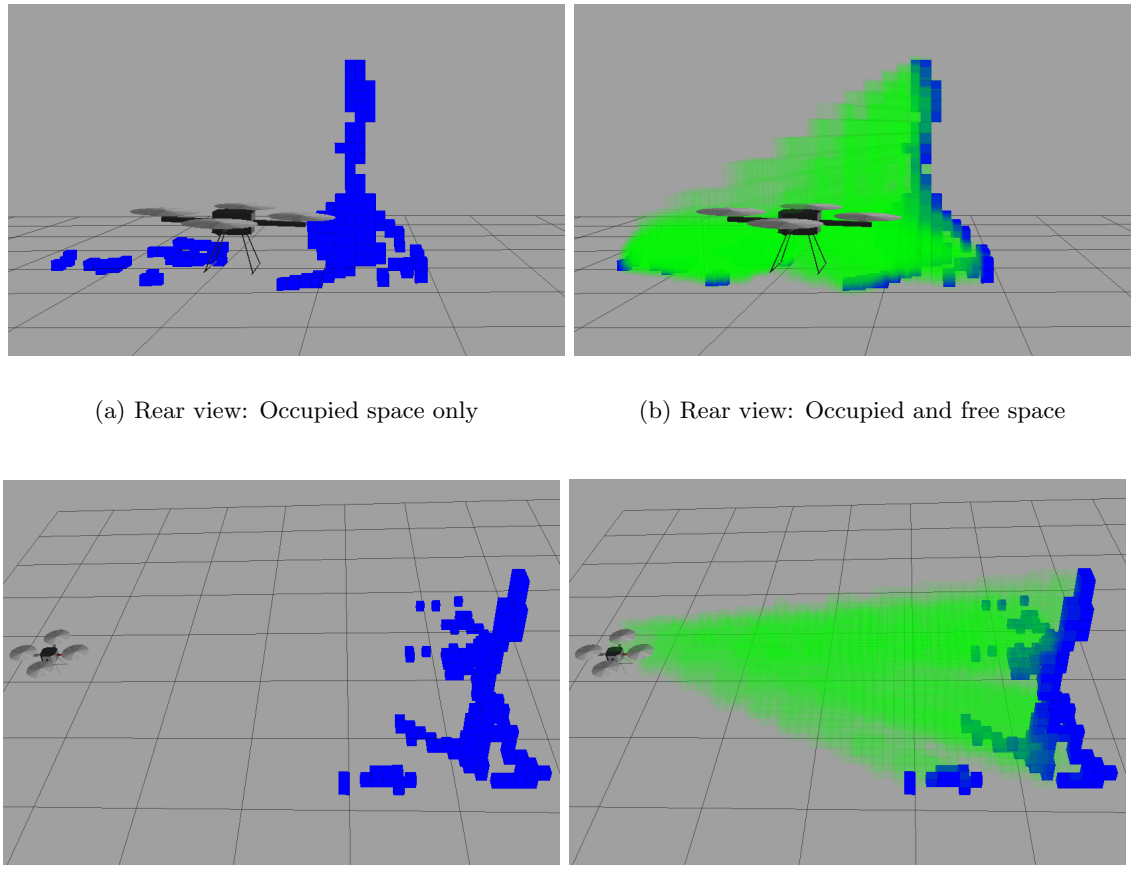


Figure 5.2: Integration of a 3D point into the occupancy map (top view). According to the beam-based sensor model, one ray is inserted from each camera center to the reconstructed 3D point. The probability of the cell that contains the point is increased. Probabilities of all other cells along the rays are decreased.

In this manner, two rays are inserted for each point of a reconstructed semi-dense point cloud. OctoMap updates the occupancy probabilities of all affected cells according to the beam-based sensor model. All cells that have not been affected by any ray are assumed to be *unknown* space. Unknown cells do not require any memory, which is important for efficiency and flexibility, i.e. the spatial extent of the map adapts to the integrated points. The map can be visualized in a nice way by classifying each cell having a probability lower than 0.5 as *free* and otherwise as *occupied*. Figure 5.3 illustrates the map after one semi-dense point cloud has been integrated. As a next step we investigate effects that arise when multiple point clouds are inserted and describe the need for *map cropping*.

Map Cropping. Whenever a semi-dense point cloud has been reconstructed, it is immediately integrated into the occupancy map. Multiple point clouds usually show a lot of redundant information, i.e. the same environmental structures. Assuming perfect localization and reconstruction, those structures are mapped to the exact same cells in the occupancy map each time. Unluckily, we have to face the problem of pose estimation uncertainty and erroneous 3D reconstruction results. While the probabilistic nature of the map allows to deal with reconstruction errors, the pose uncertainty constitutes a



(a) Rear view: Occupied space only

(b) Rear view: Occupied and free space

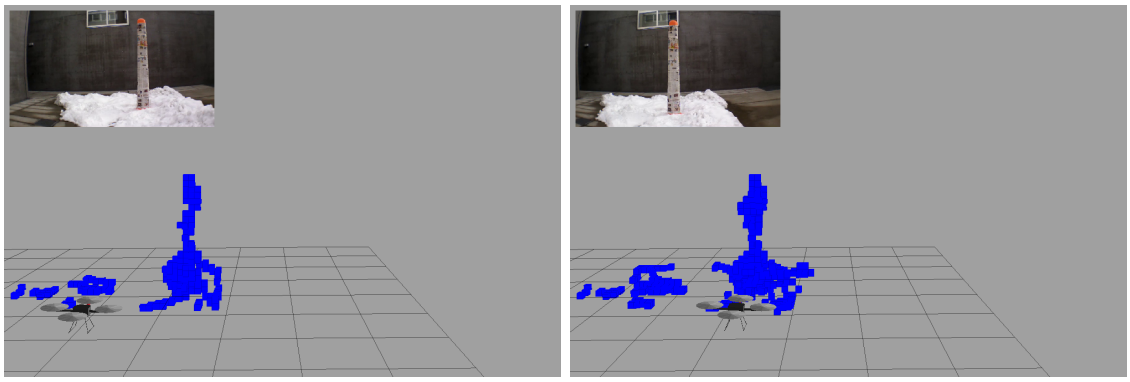
(c) Side view: Occupied space only

(d) Side view: Occupied and free space

Figure 5.3: The occupancy map after integrating one semi-dense point cloud. Occupied cells are shown in blue and free cells are shown in green. All remaining space is considered to be unknown.

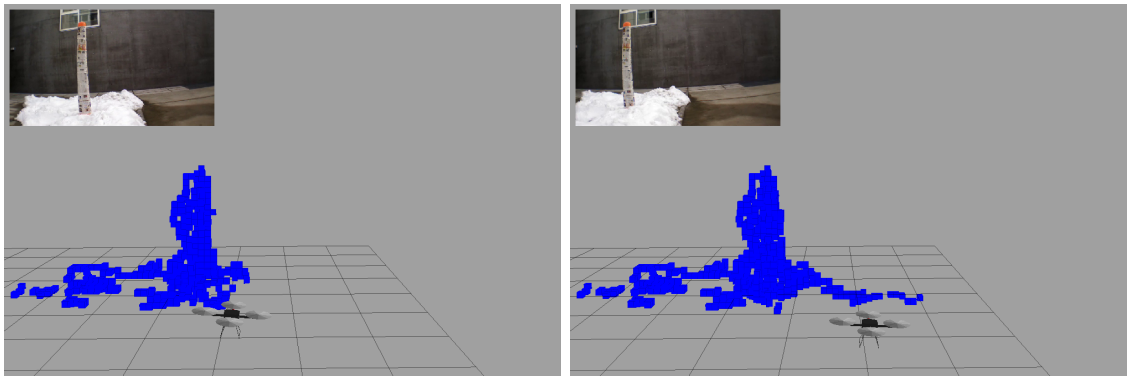
severe problem. It has been shown that global pose estimates are drifting, which leads to misaligned structures or multiple occurrence of structures in the map, as exemplified in Figure 5.4.

This degeneration of the map caused by drift needs to be countervailed. For this purpose we *crop* the map each time before a new point cloud is integrated. The cropping provokes that only cells within a small volume around the UAV's current position remain, while cells outside the volume are rejected (i.e. reset to the initial unknown state). This strategy is motivated by two observations. First, there is no need to build a large-scale map of the environment (except when global path planning is desired). In our system we employ a



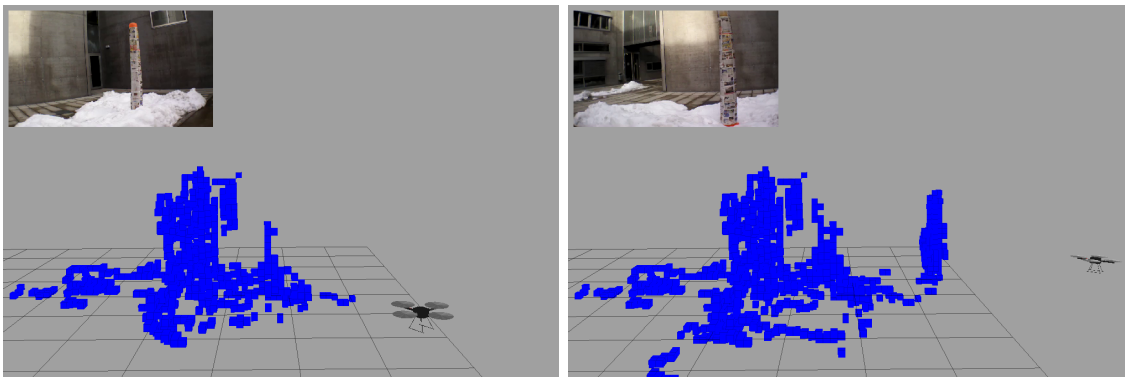
(a) 1 point cloud inserted

(b) 3 point clouds inserted



(c) 5 point clouds inserted

(d) 7 point clouds inserted



(e) 11 point clouds inserted

(f) 15 point clouds inserted

Figure 5.4: The effect of pose estimation drift on the occupancy map. Integration of multiple point clouds worsens the map due to drift of the estimated camera poses.

reactive controller that only takes local environment information into account. Second, the effect of drift increases with the distance that the UAV has covered. This means that the estimated pose is fairly accurate for short-range motion only. Thus, before integrating a semi-dense point cloud, we clear all cells outside of a fixed size volume around the current position, as shown in Figure 5.5. This volume is called *volume of interest (VOI)* and its size determines the "spatial memory" of the occupancy map.

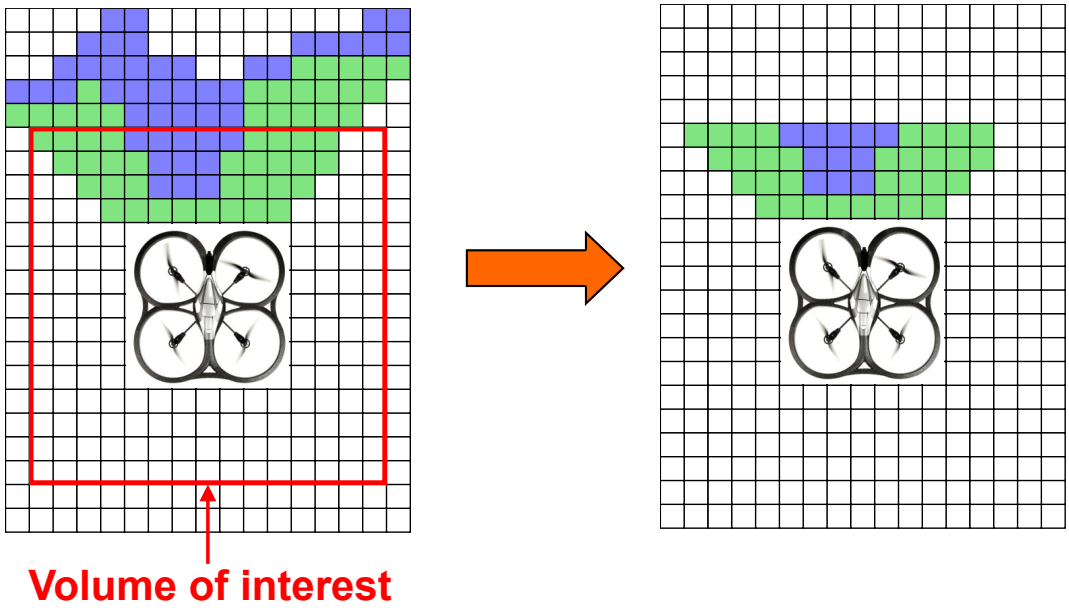


Figure 5.5: Map cropping with respect to the volume of interest (VOI). Only a small volume around the UAV is of interest for collision avoidance. Before integrating a point cloud into the occupancy map, all cells outside the VOI are reset to the unknown state. Shifting this volume through the map countervails the effect of global pose drift.

Summarized, this crop-before-insert strategy reduces the effects of localization drift on the occupancy map. Hence, the occupancy map only adheres local environment information. Rejecting spatially distant cells also keeps the memory consumption bounded to support fast map operations. As a next step an adequate way to query the map for obstacles needs to be established.

Map Querying. The probabilistic map consists of free, occupied, and unknown cells resulting from continuous point cloud integration and map cropping. Map information needs to be available to a controller process in order to initiate appropriate actions in case of close obstacles. Therefore, the mapping process provides a

service interface to enable map queries. Any controller process can initiate a map query by sending a request to the mapping process. The mapping process handles this request, retrieves information from the occupancy map, and returns abstracted information in a response message to the calling process, as shown in Figure 5.6.

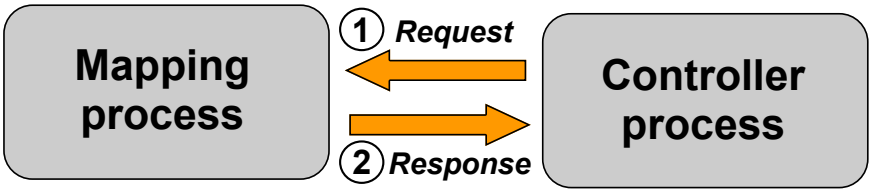


Figure 5.6: Map queries are carried out using a service interface. A controller process sends an appropriate request to the mapping process, which handles the request and returns certain information in a response message.

When the mapping process handles a service request, it abstracts information from the occupancy map by *casting rays*. A ray in 3D space is defined by a start point and a propagation direction. The ray propagates until the first occupied cell is hit and the distance from the start point is determined. It can be thought of measuring a distance with a laser sensor, where only occupied cells reflect the laser beam. Casting a single ray cannot provide reliable information and therefore multiple rays are casted at once. Their start points and the propagation direction are determined from the current pose of the unmanned aerial vehicle and certain information in the query message. In particular, we calculate a bounded rectangle through the center of the UAV, called *collision check rectangle*. The query specifies two angles, α and β, to determine the orientation of this rectangle relative to the UAV's pose. The start points are uniformly distributed on this rectangle and rays are casted normal to it, as illustrated in Figure 5.7.

The start points of all rays are determined from sampling the collision check rectangle. For logical reasons we use a sampling distance equal to the occupancy map resolution (i.e. cell size). Specification of the rectangle's rotational offset (α and β) allows collision checks in arbitrary directions. Furthermore, a query needs to specify a critical distance value d_{crit}. Casting all rays in the desired direction leads to one distance measure per ray. Each ray that hits a cell at a distance closer than d_{crit} is called a *hit*. Rays that

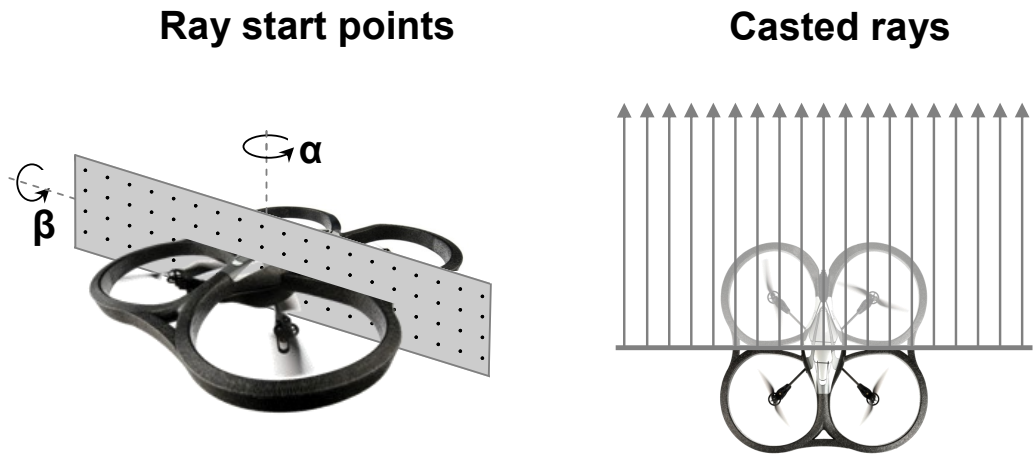


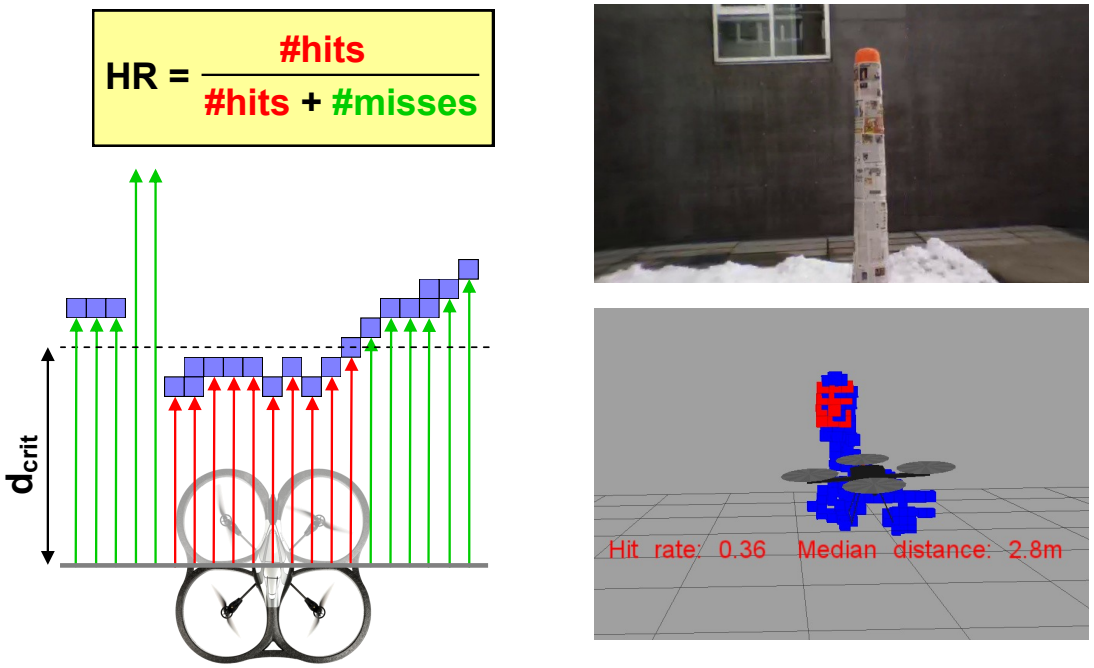
Figure 5.7: The collision check rectangle is an imaginary rectangle through the center of the quad-rotor helicopter. Angles α and β determine the rotation of this plane around the quad-rotor's height or side axis respectively. The rectangle is sampled at equidistant points from where perpendicular rays are casted into the map.

hit a cell at a greater distance or exceed a maximum distance are called a *miss*. As a result, the hit rate HR is obtained, which gives the fraction of hits among all casted rays, as illustrated in Figure 5.8. Additionally, the median distance d_{median} to all hit cells is calculated. Hit rate and median distance constitute the map query result and are returned in an appropriate response message.

Any process can query the map via the provided service interface. The query needs to contain the parameters $\{\alpha, \beta, d_{crit}\}$ and the response consists of hit rate and median distance $\{HR, d_{median}\}$. It is the task of a controller process to initiate appropriate map queries on regular bases and react accordingly. Such a process is called a *reactive controller* and is topic of the upcoming section.

5.2 Reactive Controllers

The final component in our collision avoidance system is a reactive controller that closes the loop to the UAV. Such a controller needs to initiate map queries and influence the flight path accordingly. The behavior of this controller has to be adapted to a given application or flight scenario. Based on the scenario a controller can be optimized by determining



(a) (b)

Figure 5.8: (a) A set of rays is casted from the imaginary rectangle into the map and each ray measures the distance to the closest occupied cell. Rays that hit a cell closer than the critical distance d_{crit} count as hit (red), otherwise as miss (green). The hit rate HR gives the fraction of hits among all rays and indicates the hazard due to obstacles. (b) Visualization of a map query. Hit cells are colored in red.

- at which rate the map is queried,
- which query parameters are used,
- how query results are interpreted, and
- which actions are taken based on the query results.

In this work we implemented two different reactive controllers. The first one enables autonomous forward flights, as desired in surveillance operations for instance, and is described in Section 5.2.1. The second controller assists a human operator and notifies him of hazardous obstacles, which is described in Section 5.2.2. We give a short summary and an outlook towards more sophisticated controller strategies in Section 5.2.3.

5.2.1 Reactive Controller for Autonomous Forward Flights

One goal of this work is to enable autonomous UAV operations using the collision avoidance system. We created a reactive controller to achieve this goal assuming the following scenario.

Scenario. The mission of the unmanned aerial vehicle is to progress forwards safely through an unknown environment. The focus is on safe and collision free navigation and a slow pace is acceptable. As a practical example, one can think of a surveillance drone that gradually moves ahead while recording video footage. The target flight path goes straight forward and deviations should only occur to avoid obstacles. In particular, obstacles should be evaded by guiding the UAV sideways around it, as shown in Figure 5.9.

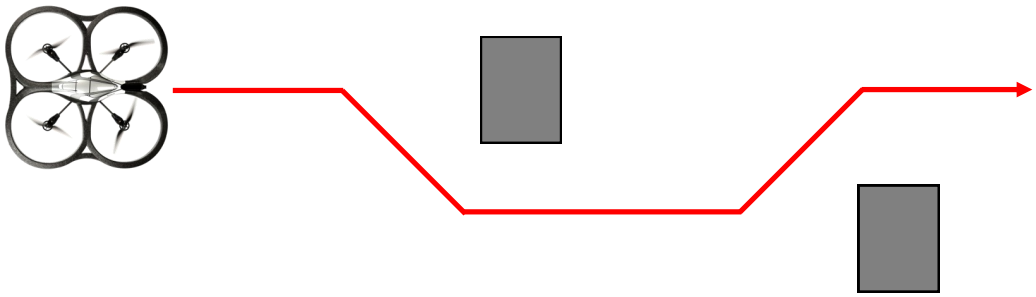


Figure 5.9: Scenario for autonomous forward flights. The target flight path of the UAV is straight forward. In case of obstacles on the target path (gray boxes) the controller guides the UAV sideways around those obstacles.

Controller Implementation. When progressing straight forward, we face the problem that no key images are generated due to insufficient baseline. To overcome this problem the controller periodically initiates a short up- and down-motion, called *baseline motion*. After this baseline motion the controller queries the map in forward direction and progresses if no close obstacle is evident. Motion of the quad-rotor is accomplished by sending *velocity commands* in an open-loop manner. For example, to let the quad-rotor ascend by one meter it can send a velocity command of "$1m/s$ upwards" for one second. We do not have a closed-loop control mechanism to ensure that the

target location is actually reached. However, it is enough to assume that the actual motion based on the velocity commands is fairly accurate.

After the up- and down-motion the controller initiates a map query in forward direction ($\alpha = 0°, \beta = 0°$). Based on the query response it decides whether a forward motion is safe or not. The motion is considered safe if the obtained hit rate HR is smaller than a threshold value HR_{max}. In this case, the controller sends velocity commands to move the UAV a fixed distance forward. If HR is greater than HR_{max}, an obstacle is assumed and an evasive motion has to be performed. For this reason, it checks whether it is safe to move half-right, i.e. 45°, or half-left respectively, by employing two more map queries ($\alpha = \pm 45°, \beta = 0°$). The first query that gives a hit rate smaller than HR_{max} results in an according motion. To keep evasion to both sides balanced, we keep track of how often the UAV moved to either side and first check in the less frequent direction. However, if neither half-left nor half-right motion is safe, no forward motion is initiated. Instead, the controller moves the quad-rotor slightly to the left or right hand side, where the less frequently evaded side is preferred again. After any motion of the UAV, the entire process is repeated. The controller strategy is illustrated as state machine diagram in Figure 5.10.

5.2.2 Reactive Controller for Assisted Flights

Another reactive controller has been implemented in the context of this work. In contrast to the previous controller it does not enable fully autonomous flights, but assists a human operator to achieve collision-free flights.

Scenario. In this scenario a human operator is required to control the UAV manually. The operator utilizes a gamepad for this purpose to generate velocity commands. The reactive controller acts as middleware layer between operator and quad-rotor helicopter. It intercepts the velocity commands and verifies each one with a map query. The command is only forwarded to the quad-rotor if the map query approves the command. If the collision check indicates a hazardous obstacle, the controller provokes the aerial vehicle to hover immediately. The human operator is informed about this circumstance through an acoustic signal and can reset the position hold with a key-press. Figure 5.11 illustrates the described scenario.

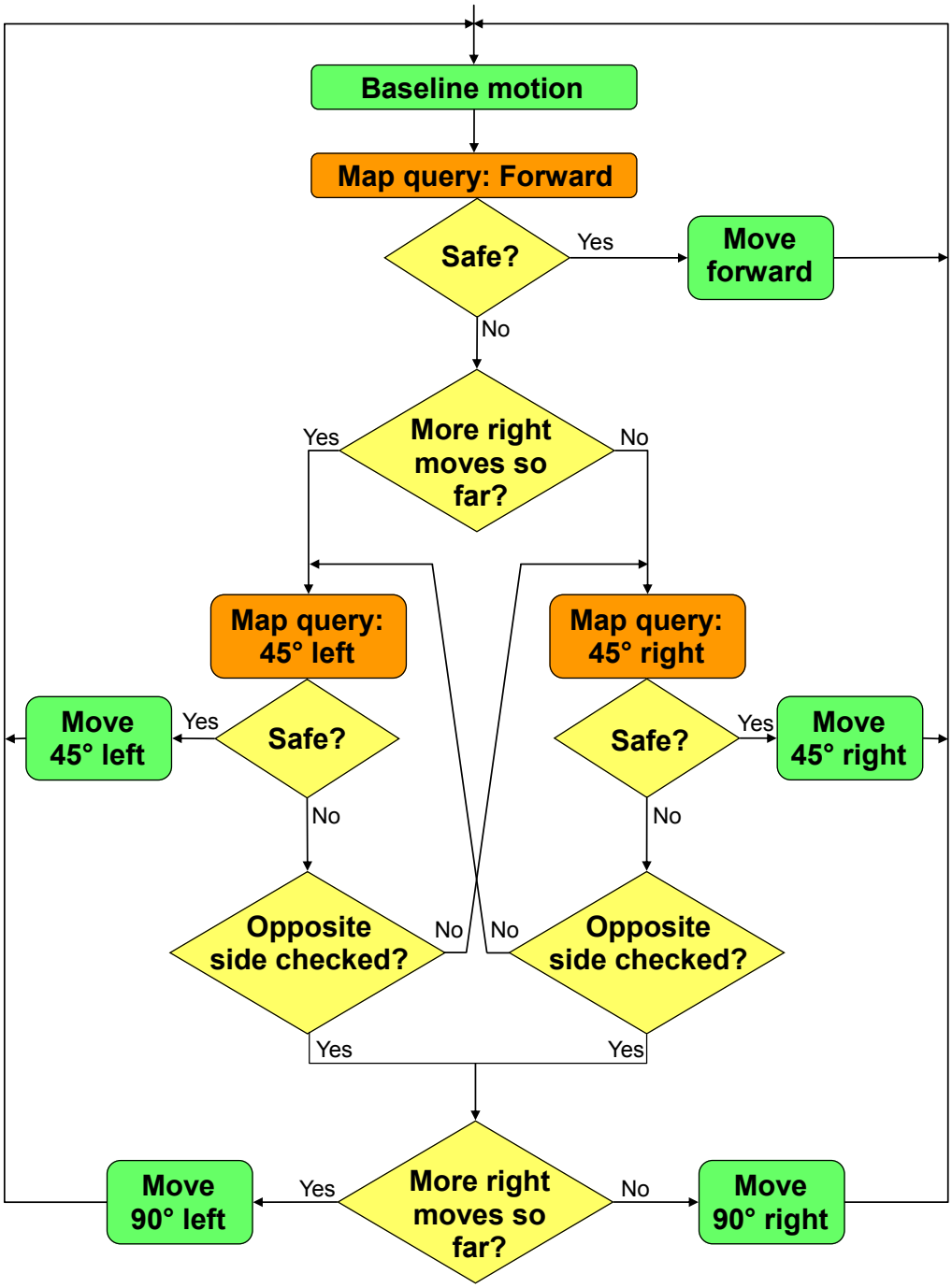
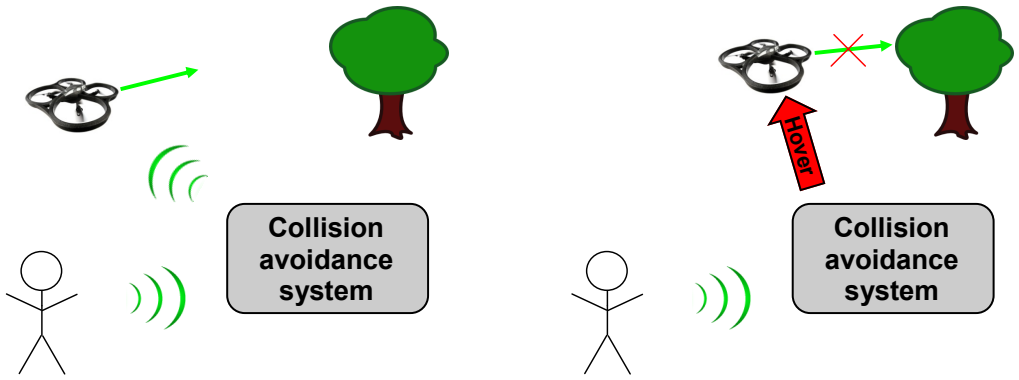


Figure 5.10: Controller for autonomous forward flights. Orange boxes: Map queries. Yellow diamonds: Controller decisions. Green boxes: Resulting motion of the UAV.



(a) (b)

Figure 5.11: Scenario for assisted flights. (a) The collision avoidance system including the reactive controller acts as middleware layer between human and UAV. It forwards the velocity commands (indicated by the green arrow) only after a map query has indicated no hazardous obstacles. (b) If the velocity command is dangerous according to the map query, the reactive controller sends a hover command instead and informs the human operator with an acoustic signal.

Controller Implementation. As with the previous controller, an important requirement is to ensure that key images are generated reliably. Of course, the human operator could take care of this by avoiding pure forward motion and flying zig-zag patterns. However, to simplify the operators task we implemented an *automatic baseline motion* feature. If this feature is activated, a constant up- and down-motion is added to the velocity commands of the user. This way, the UAV constantly ascends and descends slightly to achieve sufficient baseline without requiring user-interaction.

The velocity commands from the operator are analyzed and the desired flight direction is obtained. A map query is initiated to check the map in the assessed direction, by setting angles α and β accordingly. Again, we use a threshold on the hit rate to decide whether the velocity command is safe or not. If the hit rate is smaller than this threshold, the velocity command is forwarded to the UAV immediately. Otherwise, the controller sends permanent hover commands to the quad-rotor and informs the operator with an acoustic signal. The operator can get a glance of the situation and reset the position hold by pressing a key. This also clears the cells of the occupancy map in order to get rid of potential erroneous information. The flight can be continued and the controller verifies

each velocity command with the map again, which is successively rebuilt by integrating new 3D information.

5.2.3 Summary and Future Work

In this chapter we proposed two reactive controller implementations. It has been shown that a reactive controller can be customized for a certain scenario in order to initiate appropriate rective actions. Such reactive actions include stopping the UAV by sending hover commands or guiding it towards safer regions. The proposed controllers are fairly simple but arbitrarily complex strategies can be implemented for a given scenario.

However, developing mature controller strategies is a topic on its own and goes beyond the scope of this work. Such controllers can also exploit additional information from the map queries. For example, the median distance of hit cells d_{median} could be considered to adapt subsequent map queries. Furthermore, the map query service interface can easily be extended to retrieve supplemental map information. Distinguishing between free and unknown cells in ray casting operations or exploiting lower OctoMap resolutions are only a few ideas to promote more sophisticated reactive controllers. We suggest that future work should concentrate on the development of such mature controllers.

Chapter 6

Experiments and Results

Contents

We performed several experiments to evaluate the proposed collision avoidance system. In Section 6.1 we give an overview of the experimental setup. We assess the quality of our IMU-based localization method and compare it against the PTAM system in Section 6.2. Later in Section 6.3 those two methods are compared again, but we evaluate how accurate the *relative motion* between subsequent key images can be estimated. In Section 6.4 the entire system is evaluated in terms of computational speed. First, we break the system down into individual tasks and determine the average processing time of each task. Then, we assess the response time of the system to rapidly emerging obstacles. In Section 6.5 we present the results of fully autonomous UAV flights. Finally, we show the outcomes of assisted flights in Section 6.6 to conclude the experimental chapter.

6.1 Experimental Setup

The unmanned aerial vehicle we use throughout this work is the Parrot AR.Drone 2.0 quad-rotor helicopter. It is equipped with 2 rigidly mounted cameras – one

looking forward and the other one facing downwards. We only utilize imagery from the forward looking camera that supports a resolution of 1280x720 pixels. The quad-rotor's Inertial Measurement Unit (IMU) consists of accelerometers, gyroscopes, and magnetometers. Sensor data is fused and filtered on-board resulting in time-stamped IMU messages, which are exploited for IMU-based localization. During flight, video frames and IMU messages are constantly streamed to a Wi-Fi connected ground station computer. To avoid congestion we stream the image frames with a reduced resolution of 640x360 pixels at 10 frames per second (fps).

The ground station computer is a laptop with a 2.4 GHz Intel Core i7 processor running Linux. It performs all collision avoidance related tasks and can interfere by sending control commands back to the drone. The laptop is equipped with an NVIDIA GeForce GTX 660M Graphics Processing Unit (GPU) that allows to run costly tasks in a parallelized manner. For manual or assisted flights we utilize a Logitech™ gamepad to communicate with the UAV via the ground station, as depicted in Figure 6.1.

The experiments in Section 6.2 and 6.3 require a ground truth of the aerial vehicle's pose. For this reason we utilize a Vicon™ motion tracking system. This outside-in tracking system consists of 15 cameras to track certain markers and assess highly accurate 6DOF pose estimates. Such markers have been mounted on the UAV to establish ground truth poses, as illustrated in Figure 6.1. In the following experiment we evaluate the accuracy of UAV localization.

6.2 Localization Accuracy

In this experiment we compare the localization accuracy of the visual SLAM system PTAM to our IMU-based localization method described in Section 3.2. Throughout a flight, we constantly compare pose estimates from PTAM as well as from our IMU-based approach against the ground truth pose acquired with the Vicon system. First, we employ flight paths that meet the requirements of PTAM, i.e. we constantly face a small, well-textured scene. Then, we violate those constraints by performing a 360 degree turn and assess the effect on either localization method. In the subsequent section we describe the evaluation criteria for this experiment.

(a) (b)

Figure 6.1: Experimental setup. (a) The AR.Drone 2.0 communicates with the ground station laptop over a Wi-Fi link. For manual control we utilize a gamepad to communicate with the drone via the ground station. (b) Special markers on the drone allow highly accurate 6DOF pose estimates using the Vicon motion tracking system.

6.2.1 Evaluation Criteria

The localization accuracy is evaluated by determining *translational error* and *rotational error*. The translational error is measured in terms of the Euclidean distance

$$\varepsilon_{trans} = \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|, \tag{6.1}$$

where $\hat{\boldsymbol{x}}$ denotes the estimated position and $\boldsymbol{x}$ denotes the ground truth position from the Vicon system. The rotational error gives the difference of *orientations* between estimated and real pose. Orientations are represented as quaternions $\hat{\boldsymbol{q}} = [\hat{q}_0, \hat{q}_1, \hat{q}_2, \hat{q}_3]^T$ and $\boldsymbol{q} = [q_0, q_1, q_2, q_3]^T$ respectively. The difference between two orientations can also be represented as a quaternion $\widetilde{\boldsymbol{q}} = \hat{\boldsymbol{q}} \cdot \boldsymbol{q}^{-1}$. The rotational error is obtained from $\widetilde{\boldsymbol{q}} = [\widetilde{q}_0, \widetilde{q}_1, \widetilde{q}_2, \widetilde{q}_3]^T$ as

$$\varepsilon_{rot} = 2 \cdot \arccos(\widetilde{q}_3), \tag{6.2}$$

which gives the minimum required angle to rotate from the first to the second orientation geometrically.

The Vicon system provides ground truth poses at a rate of about 120 Hz. We measure the

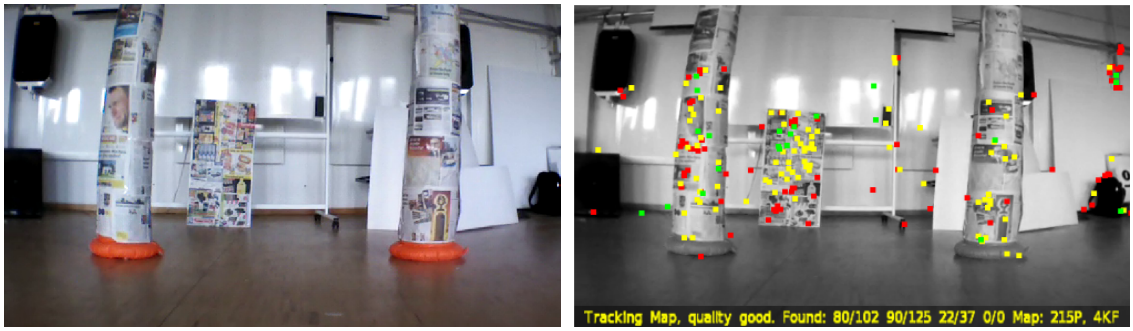
overall performance of an entire flight in terms of the Root Mean Squared (RMS) error, computed as

$$\mathbf{RMS} = \sqrt{\frac{1}{N}\sum_{n} \|\varepsilon_n\|^2}, \tag{6.3}$$

where ε_n denotes the translational or rotational error respectively at time step n. In the first experiment we compare the performance on flight paths that promote good localization with PTAM.

6.2.2 PTAM-friendly Flight Paths

In this experiment we initialized PTAM metrically, while the camera faces a small and well-textured scene. Then we manually controlled the UAV along flight paths, such that the initial scene remained in the camera's field of view, as shown in Figure 6.2. This promotes highly accurate pose estimates of the PTAM system, which are compared against our IMU-based localization method.

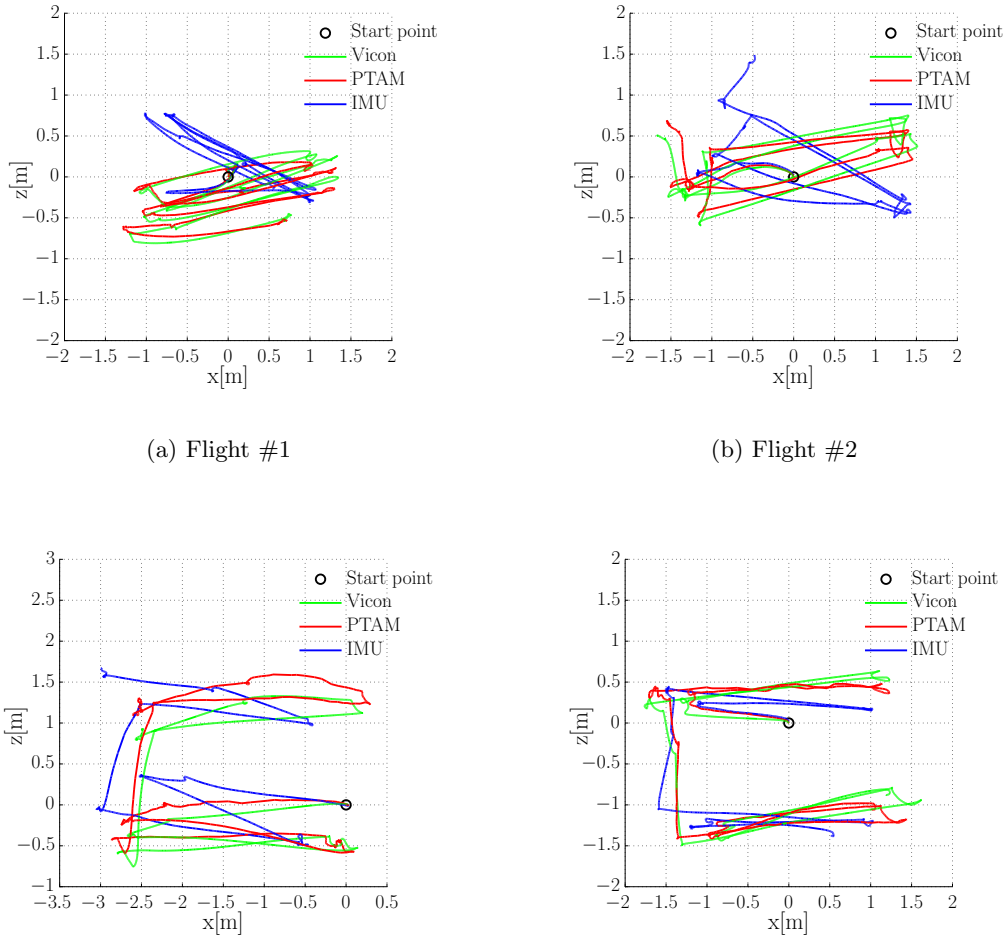


(a) Camera image (b) Tracked feature points

Figure 6.2: The rich-featured scene remains visible throughout the entire flight. Numerous visual features can be tracked to ensure optimal conditions for PTAM localization.

We performed 4 independent flights to assess the pose estimation quality of each method. The trajectories of both approaches together with the ground truth trajectory are illustrated in Figure 6.3. The overall RMS error (combining all 4 flights) is shown in Table 6.1.

This experiment proves that pose estimation with PTAM clearly outperforms the IMU-based localization if the flight path meets certain constraints. Specifically, the initial



(a) Flight #1

(b) Flight #2

(c) Flight #3

(d) Flight #4

Figure 6.3: Trajectories of all 4 flights in the localization experiment (top view). It can be seen that the IMU-based trajectories drift significantly, whereas PTAM estimates remain close to the ground truth position.

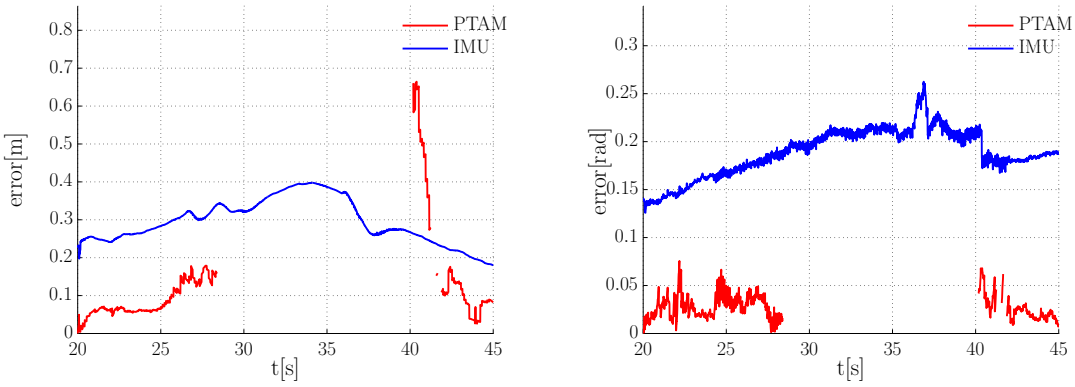
Method	Transl. error [m]	Rot. error [rad]
PTAM	**0.3962**	**0.1871**
IMU-based	0.7888	0.5823

Table 6.1: Overall RMS localization error for flight paths that meet PTAM's requirements.

scene remained visible throughout each flight, such that PTAM's tracking could always find enough points for accurate pose estimation. The trajectories illustrate that the IMU-based method suffers from drift while PTAM remains close to the ground truth. However, the good localization quality is only possible due to the constrained flight paths. In the following experiment we show the effect when those constraints are violated.

6.2.3 Violation of PTAM Constraints

The previous flight paths were specifically designed to meet the requirements of PTAM. In case of more general flight paths those constraints are usually violated. In this experiment we let the UAV turn around its height axis by 360 degree. As soon as the initial scene vanishs from the field of view, PTAM loses its localization because the tracking thread cannot find map points anymore. In this state, PTAM cannot generate any pose estimates until known map points become visible again and the system recovers. Figure 6.4 illustrates the localization error during a 360 degree turn.



(a) Translational error

(b) Rotational error

Figure 6.4: Localization error during a 360 degree turn. The turn starts at $25s$ and PTAM loses localization at $28s$, after known map points have vanished from the camera's field of view. During this time, PTAM is not able to establish pose estimates. Recovery starts at $40s$ and the localization is successfully recovered at $42s$. At $43s$ the turn is complete and the entire initial scene is faced again.

The experiment demonstrates that PTAM cannot handle maneuvers such as sharp turns without losing the localization. This is because its internal map of 3D points only contains world points near the initial scene. Although the map is extendable, it would

require highly strategic path planning to extend the map during a turn without loss of localization. Therefore, the feasibility of PTAM for UAV localization is restricted to constrained flight paths.

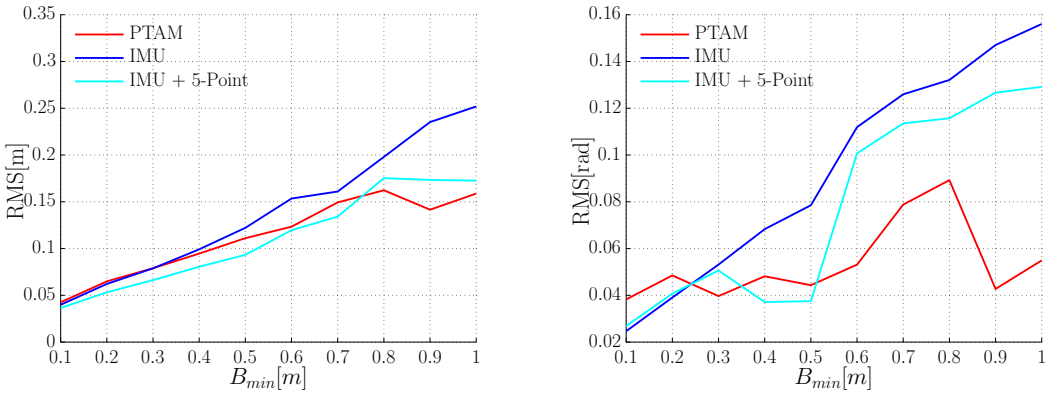
In contrast, our IMU-based localization method is not dependent on the presence of visual features. It is solely based on sensor data from the Inertial Measurement Unit. Since this method lacks any global reference information, it suffers from global pose drift. However, since we do not need to build a globally consistent large-scale map, this drift is acceptable in our system. Nevertheless, it is essential that the relative motion between key images is estimated accurately to achieve high quality reconstructions, which is assessed in the following experiment.

6.3 Relative Motion Estimation

Each multi-view reconstruction method that uses a single camera requires an accurate estimate of the *relative motion* between images to match. The accuracy of this estimate has a major impact on the reconstruction quality. In our system we constantly perform reconstruction between pairs of subsequent *key images*. In this experiment we evaluate the quality of relative motion estimation between those key images. Again, we obtain the ground truth from the Vicon motion tracking system. Since a relative motion can also be specified as a 6DOF pose, we use the same error measures as described in 6.2.1.

We compare 3 different methods against the ground truth. The first method estimates the relative motion using PTAM. Again, we initialize PTAM on a well-textured scene and avoid sharp turns to prevent localization loss during flight. When the i-th key image is selected, the camera pose $\mathcal{P}_i$ according to PTAM is stored. The relative motion is then obtained as the transformation between pose $\mathcal{P}_{i-1}$ and $\mathcal{P}_i$. The second method works accordingly, but the individual camera poses are obtained from our IMU-based localization technique. The difference to the previous experiment is that not the global 6DOF pose is of interest but the relative pose between subsequent key images. The third method refines the purely IMU-based estimate, as described in Section 4.2.2. In particular, the relative motion is estimated from image correspondences using the Five-Point algorithm [38], where the missing scale factor is obtained from the IMU-based estimate.

An important parameter in this experiment is the minimum baseline B_{min}. This value determines the required motion in horizontal or vertical direction that results in a new key image. We varied the minimum baseline between $0.1m$ and $1.0m$ and assessed the relative motion error of all 3 methods. The results are graphically illustrated in Figure 6.5.



(a) Translational error

(b) Rotational error

Figure 6.5: Translational and rotational error of the relative motion estimation. For small values of B_{min} the accuracy of all 3 methods are comparable. Thus, the effect of drift on the IMU-based methods is negligible for short-range motion.

The experiment proves that each method can estimate small relative motions accurately. For small values of B_{min}, the pure IMU-based estimates show comparable accuracy to PTAM. Earlier experiments have shown that IMU-based localization tends to drift. However, small motions can be estimated accurately using IMU data and are not noticeably affected by drift. Refinement of the IMU pose using the Five-Point algorithm leads to slightly improved results. When a small value of B_{min} is used, our proposed method is more accurate and can potentially triangulate better 3D points than PTAM. Furthermore, we do not require expensive tracking and flight paths do not need to be constrained. In the following section we evaluate the entire system in terms of computational speed.

6.4 System Speed Evaluation

Computational efficiency is an essential requirement for our system in order to perform in real-time. We evaluate the speed of the system in two ways. First, the processing time of

each main system task is determined and the impact of exploiting the GPU is assessed. Then we measure the time it takes to detect obstacles that are rapidly placed in front of the UAV in real-world experiments.

6.4.1 Execution Time of System Tasks

In this experiment we assess the average execution time of each task in the system; starting from frame arrival until feedback is sent to the UAV. The collision avoidance system is therefore divided into subsequent tasks and execution time is broken down into those tasks. Our ground station computer is able to perform some tasks on its Graphics Processing Unit (GPU). We investigate the benefits of this ability by comparing execution times when GPU capabilities are enabled and disabled, respectively.

The proposed collision avoidance system can be broken down into the following logical tasks. Tasks that are marked with a star($*$) can be executed on the GPU.

1. **Baseline verification:** The system receives a frame and determines whether it provides a sufficient baseline to the last key image. Only if this is the case, the frame becomes a keyframe and is processed; otherwise the frame is dropped.
2. **Image conversion and undistortion:** If the baseline is sufficient, the key image is converted into an appropriate data format and camera lens distortion is corrected.
3. **Feature extraction*:** Interest points in the current key image are detected and SIFT image features are extracted.
4. **Correspondence matching*:** Extracted feature vectors are matched with features from the previous key image, resulting in a set of image correspondences.
5. **Five-Point pose estimation*:** Given the image correspondences, the relative pose is estimated using the Five-Point algorithm.
6. **Sparse point triangulation:** A set of sparse 3D points is triangulated from image correspondences and the previously assessed relative pose.
7. **BPA densification:** Sparse 3D points are densified to a semi-dense point cloud using the Ball Pivoting Algorithm (BPA).
8. **Map cropping:** The map is cropped such that only cells around the current UAV position remain in the occupancy map.

9. **Ray insertion:** Rays are integrated into the occupancy map by updating cells according to the inverse sensor model.

10. **Collision check:** The map is queried for obstacles by casting rays. Although collision checks are usually initiated by a separate controller, we check right after insertion here to complete a full system cycle.

Execution without GPU Capabilities. First, we analyze the execution times when all tasks are performed on the CPU. Therefore, we recorded 50 seconds of a flight along an arbitrary flight path. The minimum baseline was set to 0.2m, which results in 31 generated key images throughout the flight. The individual execution times of all tasks were logged and the average time of each task was determined, as shown in Figure 6.6.

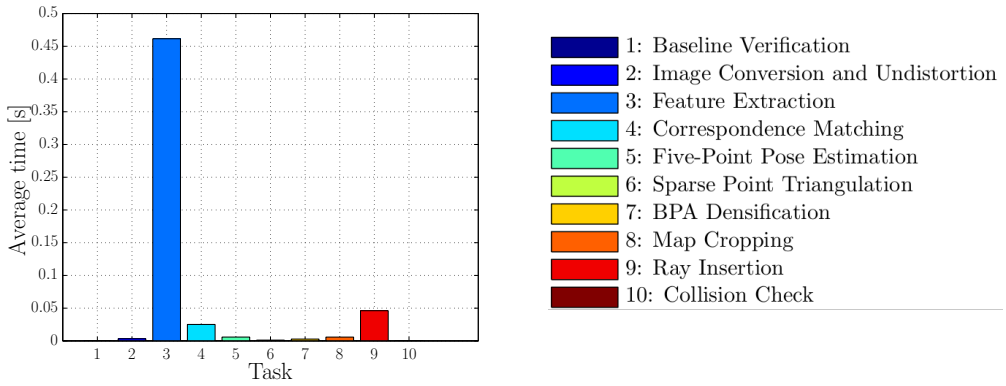


Figure 6.6: Average execution times with disabled GPU-capabilities. SIFT feature extraction is clearly the most costly task. An entire system cycle takes 0.55 seconds.

The most costly task is obviously the extraction of SIFT features from an image. Feature extraction took 0.46 seconds on average, which constitutes more than 80 percent of an entire system cycle. An average cycle took 0.55 seconds so that the system can roughly handle two key images per second. Luckily, our ground station is able to perform some tasks on the GPU and the resulting speedup is assessed subsequently.

Execution with GPU Capabilities. Now we analyze the effect of enabling GPU capabilities on the execution times. As previously explained, three tasks can be carried out on the GPU, which are *feature extraction*, *correspondence matching*, and *five-point pose estimation*. We observed that the pose estimation task runs faster on the CPU and

therefore only the first two tasks were actually executed on the Graphics Processing Unit. We analyzed the exact same 50 seconds of the recorded flight and assessed the execution times, as shown in Figure 6.7.

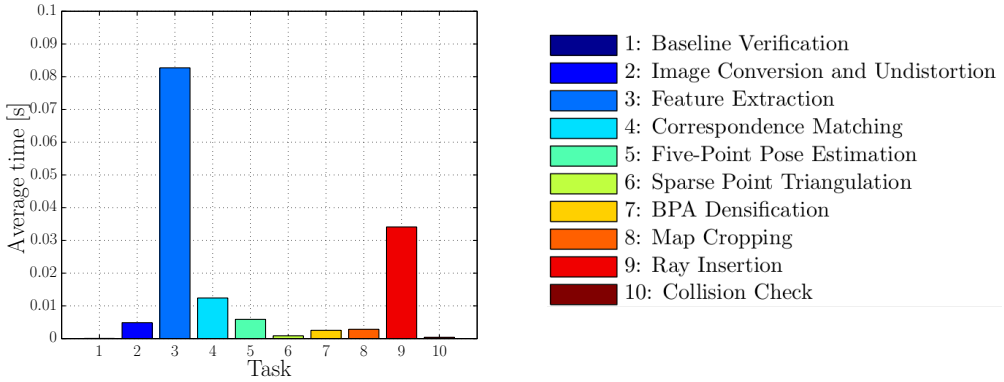


Figure 6.7: Average execution times with enabled GPU-capabilities. SIFT feature extraction and correspondence matching are significantly speeded up. An entire system cycle only takes 0.15 seconds now.

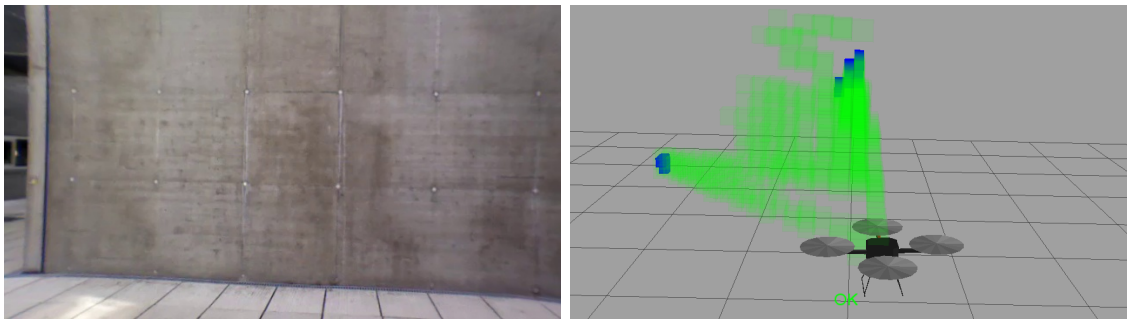
The experiment proves that exploiting the GPU leads to significant speedups. Extraction of SIFT features only took 0.08 seconds on average, which is 6 times faster than on the CPU. Furthermore, correspondence matching took 0.012 seconds compared to 0.025 seconds on the CPU, which is still twice as fast. In this configuration an entire system cycle took 0.15 seconds. Thus, our system is able to handle more than 6 key images per second and can clearly meet real-time requirements.

If no GPU-capable ground station is available, it is still possible to speedup the execution by using a less complex feature descriptor. Replacing SIFT with the more efficient SURF [6] descriptor speeds up the feature extraction task from 0.46 to 0.16 seconds and still yields good reconstruction results. This makes the system feasible to use with ground stations that are not able to perform tasks on the GPU. In the next experiment we determine the response time of the system to rapidly emerging obstacles.

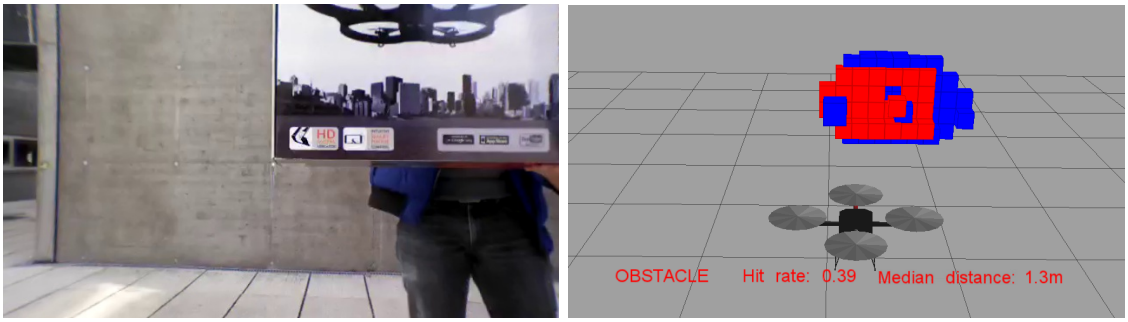
6.4.2 Response Time to Dynamic Obstacles

To complete the speed evaluation we analyze the system's response time to rapidly appearing real-world obstacles. For this purpose we let the UAV fly up and down

constantly in an environment without close obstacles. The constant up-down-motion is necessary to generate key images on a regular basis. At time t_0 we rapidly place an obstacle in front of the up- and down-moving quad-rotor helicopter. Instead using a separate reactive controller we employ a collision check each time after rays have been inserted into the map. It is expected that the collision avoidance system successfully detects the obstacle at time t_1. The response time $\Delta t = t_1 - t_0$ is measured and evaluated in the experiments. Figure 6.8 illustrates the experimental setup.



(a) (b)



(c) (d)

Figure 6.8: (a) First, the quad-rotor constantly moves up and down seeing no close obstacles. (b) During this time the map contains mainly free space (green) and a few occupied cells (blue) far away. (c) At time t_0 an obstacle is placed rapidly in front of the UAV. (d) After the obstacle has been reconstructed and inserted into the occupancy map it is detected at time t_1. Free space is not shown for better visibility here.

Besides the processing time of the system tasks, the response time mainly depends on the rate at which key images are generated. A key image is generated if the horizontal or vertical distance to the previous keyframe exceeds the fixed value of B_{min}. Large values

of B_{min} or slow UAV motion are therefore causing long response times. Consequently, a fast moving UAV or a small B_{min} value result in shorter response times due to faster key image generation. In the performed experiments the UAV moved up and down with a velocity of approximately $0.5m/s$. We recorded 5 instances of the experiment and determined the response times for different values of B_{min}. Precisely, we set the minimum baseline B_{min} to $0.1m$, $0.2m$, and $0.3m$ respectively. Table 6.2 shows the entire parameter setting used throughout these experiments.

Parameter	Value
Localization method	IMU-based
GPU processing	Enabled
Minimum baseline B_{min}	0.1m, 0.2m, and 0.3m respectively
OctoMap resolution	0.1m
BPA radius ρ	0.3m
Maximum ray length	5m
Volume of interest	5m × 5m × 5m
Collision check rectangle	1.0m × 0.5m (width × height)
Critical distance d_{crit}	2m
Hit rate threshold HR_{max}	0.1

Table 6.2: Parameter setting for the response time experiments. Five instances of the experiment have been performed and the response time Δt was assessed.

Response time. We recorded 5 independent instances of the experiment. The recorded data was fed to our collision avoidance system and the response time Δt was determined manually. We assessed the influence of B_{min} on the response time of the system. Our system was able to detect the obstacle successfully in all five experiments. Table 6.3 shows the response times that we measured.

	$B_{min} = 0.3m$	$B_{min} = 0.2m$	$B_{min} = 0.1m$
Experiment #1:	$1.9s$	$1.6s$	**1.1s**
Experiment #2:	$2.4s$	$1.5s$	**1.2s**
Experiment #3:	$1.8s$	$1.7s$	**1.0s**
Experiment #4:	$1.9s$	**1.6s**	$1.7s$
Experiment #5:	$2.6s$	$1.9s$	**1.4s**
Average:	$2.12s$	$1.66s$	**1.28s**

Table 6.3: Response time Δt to rapidly emerging obstacles.

The experiments prove that our system is able to detect dynamic obstacles without significant delay. As expected, the response times are shorter when key images are generated faster due to a small value of B_{min}. However, the quality of reconstructed 3D points decrease when B_{min} is very small as a result of the short baseline between key images. When using $B_{min} = 0.1m$ we observed one *false positive detection* in Experiment #3, which means the system detected an obstacle before it was visible. Furthermore, in Experiment #4 the response was slower than for $B_{min} = 0.2m$. The reason was that despite key images have been generated faster, it required more image pairs until the obstacle was successfully detected.

Another thing to keep in mind when using small baselines is computational effort. If key images are generated faster than the system can process them, real-time requirements are violated. Our speed evaluation showed that key images can be processed in 0.15 seconds when exploiting the GPU. Thus, the system could easily handle even the smallest baseline of 0.1m. However, without GPU-capabilities the system would have been pushed to its limits. We found that a good tradeoff between speed and accuracy is achieved with a minimum baseline of 0.2 meters. In the next experiment we present results of autonomous UAV flights.

6.5 Autonomous Flight Results

In this section we present the results of fully autonomous flights in a cluttered environment. We utilized two inflatable pylons as obstacles and placed them 2 meters in front of the quad-rotor's start point. To provide sufficient texture on the obstacles we covered both pylons with newspaper. As described in Section 5.2.1, the target flight path is straight forward and in case of an obstacle the reactive controller should guide it *sidewards* past the obstacle. To achieve a sufficient baseline between key images, the unmanned aerial vehicle performs a *baseline motion*, i.e. a short up- and down motion, before progressing forwards. We set the minimum baseline parameter B_{min} to $0.2m$ and the remaining parameters according to Table 6.2. The experimental setup is depicted in Figure 6.9.

We performed 6 autonomous flights towards the inflatable pylons and utilized the Vicon motion tracking system to record the trajectories. The reactive controller periodically initiated a baseline motion before checking for obstacles in the occupancy map. Based on

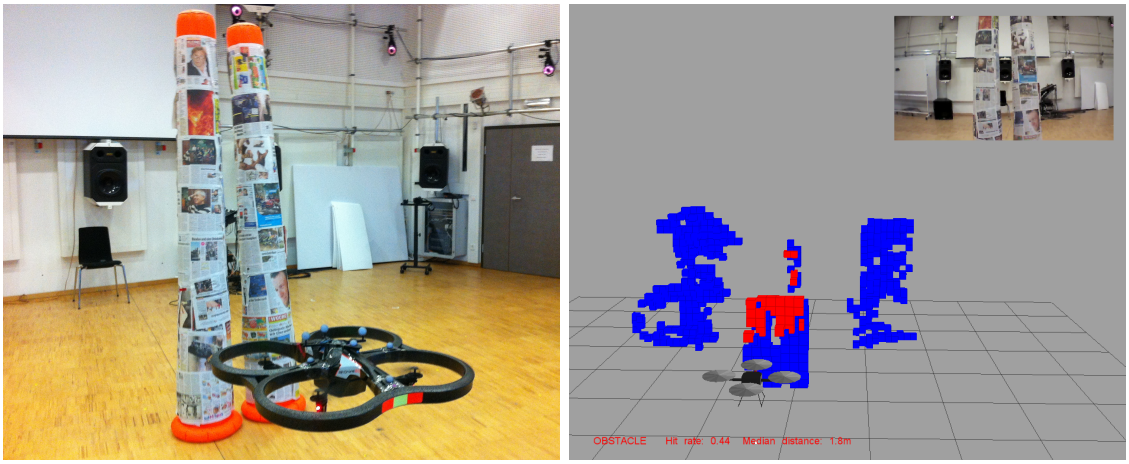


Figure 6.9: Setup of the autonomous flight experiment. (a) Two inflatable pylons were used as obstacles and placed in front of the quad-rotor helicopter. (b) The goal is that the collision avoidance system detects the obstacles during the flight and guides the quad-rotor sidewards past the pylons.

the query results it proceeded either straight forward or evaded to the right or left hand side. The system was able to detect the obstacles and guide the UAV collision-free past them in all 6 experiments. The trajectories of 3 flights are illustrated in Figure 6.10.

Discussion. The experiments prove that our collision avoidance system can detect and avoid obstacles in real-time during a flight. It is evident that the trajectories of individual flights vary significantly, which is a result of the slightly different start positions. Furthermore, it can be seen that the resulting trajectories deviate from the target path, which is straight forward, even after the obstacle has passed. This is mainly due to turbulences from the rotor-airstream, which cause little displacements or rotations of the UAV. If this affects the quad-rotors heading, it progresses towards an offset direction, as shown in the first two rows of Figure 6.10.

A correction of this heading error is difficult for two reasons. First, the IMU-based localization method might not always be aware of such a rotational offset, as illustrated in the top left image in Figure 6.10. Second, the system can only guide the quad-rotor by sending velocity commands in an open-loop manner. Accurate motion maneuvers (e.g. “correct the heading by 5 degree and progress exactly 1m in forward direction”) would clearly require a closed-loop control mechanism. However, despite following a target flight

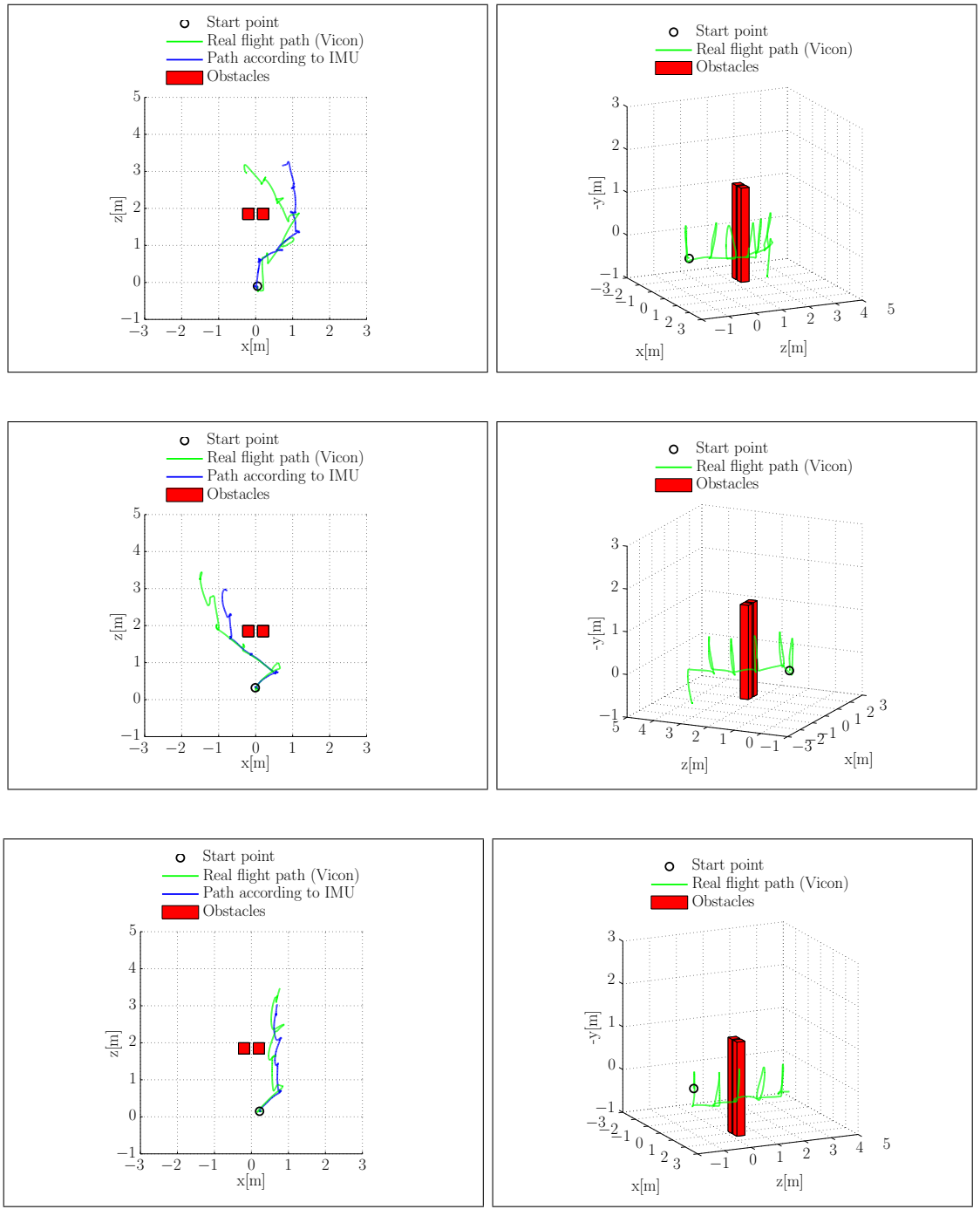


Figure 6.10: Resulting flight paths of 3 autonomous flight experiments. Each row illustrates the resulting path of one autonomous flight. The left column shows a top view where the global position drift of the IMU-based localization method is apparent. The right column shows appropriate side views and the baseline motion can be seen.

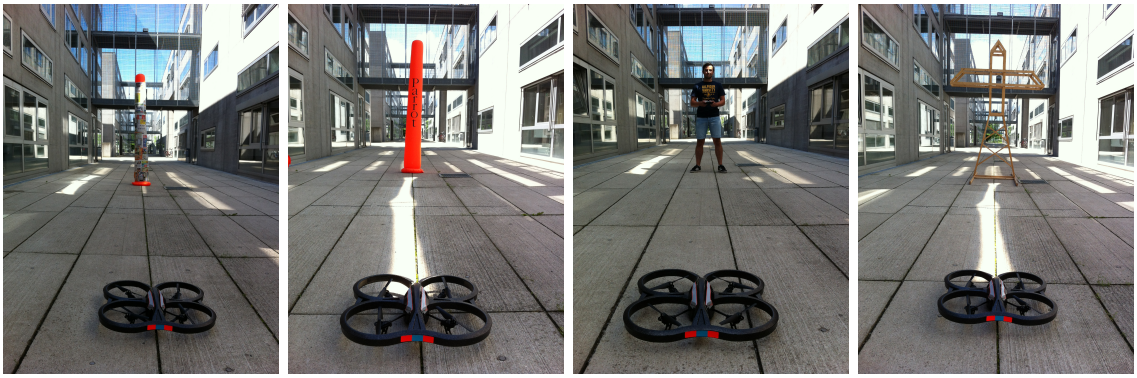
path is problematic, our experiments prove that well-textured obstacles can be detected and avoided reliably. In the subsequent section, we evaluate the performance on different types of obstacles and assess its influence on the detection reliability. To avoid path deviations we employ assisted flights, which means the unmanned aerial vehicle is controlled manually and stops in case of close obstacles.

6.6 Assisted Flight Results

Finally, we investigate the performance of the system in terms of reliable obstacle detection. Therefore, we performed several assisted flights in an outdoor environment and utilized the reactive controller described in Section 5.2.2. The UAV was controlled manually using a gamepad and the reactive controller assists the human operator. As previously described the controller constantly adds a slight up- and down-motion (called automatic baseline motion) to the operator's commands to ensure key image generation. The controller verifies each command from the operator in terms of safety with the occupancy map. If the command has been approved it is forwarded to the UAV. Otherwise, the controller stops the UAV immediately and informs the operator with an acoustic signal.

The goal of these experiments was to assess how good different obstacles can be detected by the system. Four different obstacles were used, which clearly differed in terms of surface texture, as shown in Figure 6.11. We performed 10 independent flights towards each obstacle and assessed whether the system was able to stop the UAV before colliding with it. We evaluate the quality by determining the achieved *success rate*, which gives the fraction of successful experiments among the flights. An experiment was considered successful if the UAV was stopped before touching the obstacle.

Each flight started at a distance of 5 meters from the respective obstacle. The critical distance d_{crit} was set to $2m$ and we demanded that the controller has to stop the UAV within this distance. If it is stopped earlier, the result is interpreted as a *false positive detection* and the experiment is not considered successful. We observed that false positives are more likely in the specific outdoor scene and thus increased the hit rate threshold HR_{max} to 0.15. The minimum baseline parameter B_{min} was set to $0.2m$ again. The forward speed of the UAV was approximately $0.5m/s$ and the altitude deviation due to automatic baseline motion was $\pm 0.2m$. All remaining parameters were set according to Table 6.2.



(a) Well-textured pylon (b) Weakly-textured pylon (c) Person (d) Wiry model

Figure 6.11: The obstacles used in the assisted flight experiments. In our experiments we assessed how often each obstacle is detected among 10 flights.

Results on different obstacles. First, we used one inflatable pylon covered with newspaper as obstacle. The purpose of the newspaper was to provide a well-textured surface on the object promoting good detection results. In fact, the system had no problems to detect this rich-featured object and stop the UAV before a collision occured. The controller was able to stop the UAV successfully in all 10 flights and thus achieved a success rate of **100%**.

Next, we removed the newspaper from the pylon and repeated the entire experiment. The pylon has a homogeneous orange surface with a few black letters on it, as shown in Figure 6.11(b). Compared to the previous setting the amount of surface texture was clearly reduced. Again, we performed 10 flights towards the obstacle and achieved a success rate of **50%**.

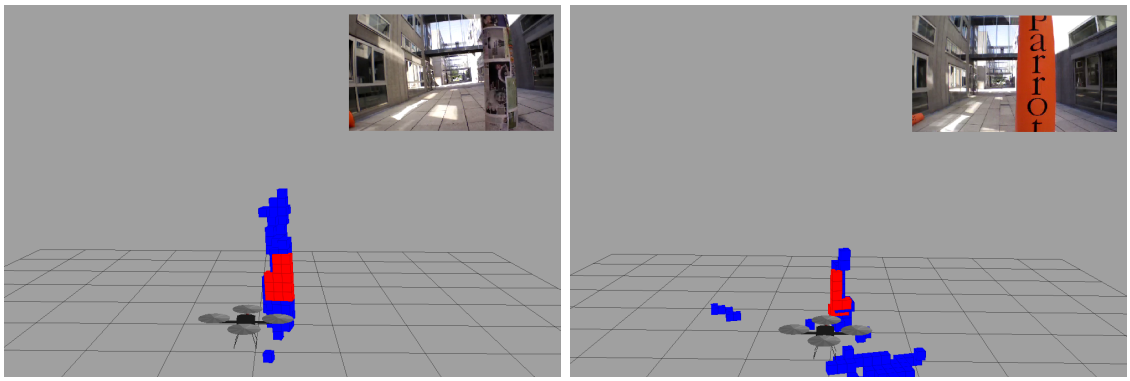
In the third experiment series we assessed the sucess rate when using a human person as obstacle. The system was able to succeed in 8 of 10 flights. In the remaining two flights it stopped the UAV but slightly too late, such that the UAV had already touched the person. However, since we only consider an experiment successful where the UAV was stopped before contact, the achieved success rate is **80%**.

Finally, we used a wiry, wooden model as our last obstacle. Although this object was larger than the previous ones, it was the most challenging obstacle due to its wiry structure and lack of texture. The system could achieve a success rate of **30%**. A summary of the achieved success rates is given in Table 6.4. Moreover, some screenshots from the experiments with different obstacles are shown in Figure 6.12.

Obstacle	Success rate
Well-textured pylon	100%
Weakly-textured pylon	50%
Person	80%
Wiry model	30%

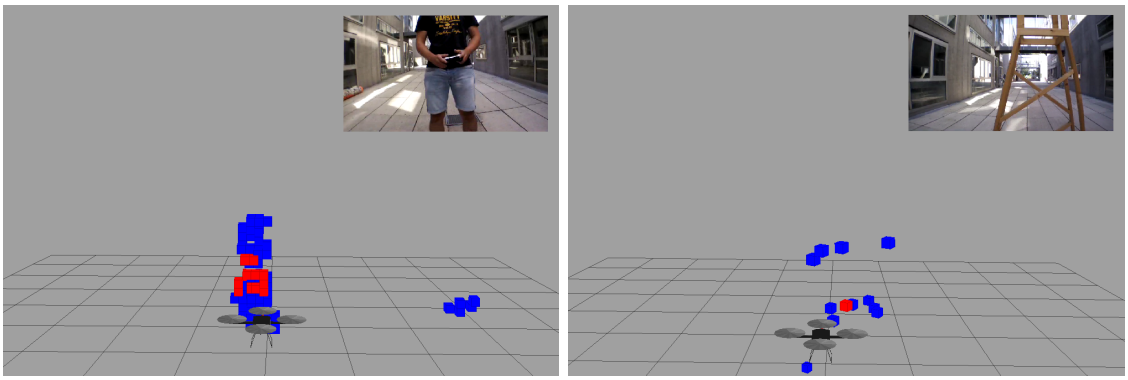
Table 6.4: Success rates of the assisted flight experiments on four different obstacles.

Discussion. The proposed experiments prove that a successful detection is heavily dependent on the provided texture of an obstacle. We demonstrated the ability of our system to detect and avoid well-textured obstacles reliably. Untextured or weakly-textured objects are problematic, which is a result of our approach to reconstruct 3D information from image feature correspondences. However, future work can focus on the incorporation of additional information, such as optical flow [54] or line segments [23]. The speed evaluation in Section 6.4 demonstrated the computational efficiency and proved that real-time requirements are clearly met. Thus, there are still resources to incorporate additional information while adhering real-time capabilities.



(a) Well-textured pylon (success rate: 100%)

(b) Weakly-textured pylon (success rate: 50%)



(c) Person (success rate: 80%)

(d) Wiry model (success rate: 30%)

Figure 6.12: Screenshots from the assisted flight experiments. (a) The pylon covered with newspaper provided rich features and could reliably be detected. (b) The uncovered pylon could be detect when the black letters were clearly visibly in subsequent key images. (c) A person could be detected in most experiments since enough image features were present. (d) The wiry model was problematic since only a few points on the object could be reconstructed.

Chapter 7

Conclusion and Outlook

Contents

In this work we have presented a system for collision avoidance on unmanned aerial vehicles. We use a monocular camera as visual input device and efficiently obtain three-dimensional information from the video stream. The 3D information is integrated into a probabilistic occupancy map and a reactive controller utilizes this map to give appropriate feedback to the UAV. To conclude our work, we give a short summary of our main contributions and present an outlook to future work.

7.1 Conclusion

We have shown that UAV localization based on data from the Inertial Measurement Unit (IMU) is feasible for our system. In contrast to visual SLAM methods, this approach is not dependent on visual landmark tracking and is therefore superior for unconstrained flight paths. Despite our method is affected by global drift, we have assessed that short-range motion can be estimated sufficiently. Since only spatial regions close to the UAV are of interest, the global drift has little negative impact on the quality of the system.

In the computational speed evaluation we have shown that our system can meet real-time requirements. Exploiting a ground station with GPU processing capabilities enables the system to process more than 6 key images per second. Furthermore, we have demonstrated that the system achieves fast response times to rapidly emerging

real-world obstacles. The promising speed of the system allows further extensions without compromising real-time capabilities.

We have shown that the collision avoidance system can be used with diverse UAV tasks by employing customized reactive controllers. Two reactive controller strategies have been implemented to enable safe autonomous forward flights as well as assisted flights. In our experiments we have proven that the system can detect and respond accordingly to different hazardous obstacles.

7.2 Future Work

We have assessed that visual SLAM methods can achieve very accurate global localization as long as enough visual landmarks can be tracked. The overall localization quality could be improved if such a method is combined with our IMU-based technique. This way, the localization drift could be reduced, which allows to build large-scale maps for enriched applications.

In our experiments we have encountered that autonomous waypoint flights are difficult when the UAV is controlled by sending open-loop velocity commands. The reason is that effects as airstream cause considerable deviations from a desired flight path. Thus, autonomous flights could be improved by establishing a closed-loop method for controlling the UAV. Together with an accurate localization this would enable autonomous waypoint flights in large, cluttered environments.

Future work should also aim to create enhanced reactive controller strategies. A lot of additional information could be incorporated, such as the spatial distribution of free, occupied, and unknown cells or map query results on lower occupancy map resolutions. The long-term goal should be to go from the reactive controller scheme towards global path planning. However, accurate localization and a closed-loop UAV control method are prerequisites to achieve this goal.

We have shown that the proposed system is able to detect sufficiently textured obstacles in a reliable way. However, detecting weakly-textured or wiry obstacles is still a challenging problem and requires to gather additional information. Incorporation of optical flow or line elements are only a few possibilities that could improve the detection quality.

Appendix A

Acronyms

List of Acronyms

AR	Augmented Reality
BA	Bundle Adjustment
CPU	Central Processing Unit
DoG	Difference of Gaussian
DSM	Digital Surface Model
EKF	Extended Kalman Filter
FAST	Features from Accelerated Segment Test
FPS	Frames Per Second
FOV	Field Of View
GPS	Global Positioning System
GPU	Graphics Processing Unit
IR	Infrared
IMU	Inertial Measurement Unit
LRF	Laser Range Finder
MAV	Micro Aerial Vehicle
PTAM	Parallel Tracking and Mapping
RANSAC	Random Sample Consensus
RMS	Root Mean Square
ROI	Region Of Interest
SfM	Structure from Motion
SIFT	Scale Invariant Feature Transform

SLAM	Simultanious Localization and Mapping
STOC	Stereo On Chip
SURF	Speeded Up Robust Features
UAV	Unmanned Aerial Vehicle
VSLAM	Visual SLAM
VOI	Volume Of Interest
WiFi	Wireless Fidelity

Bibliography

[1] Amanatides, J. and Woo, A. (1987). A fast voxel traversal algorithm for ray tracing. In *Proceedings of Eurographics*.

[2] Azad, P., Gockel, T., and Dillmann, R. (2011). *Computer Vision: Das Praxisbuch*. Elektor-Verlag, 2nd edition.

[3] Bachrach, A., He, R., and Roy, N. (2009). Autonomous flight in unknown indoor environments. *International Journal of Micro Air Vehicles*, 1(4):217–228.

[4] Bachrach, A., Huang, A. S., Maturana, D., Henry, P., Krainin, M., Fox, D., and Roy, N. (2011a). Visual navigation for micro air vehicles.

[5] Bachrach, A., Prentice, S., He, R., and Roy, N. (2011b). RANGE – Robust autonomous navigation in GPS-denied environments. *Journal of Field Robotics*, 28(5):644–666.

[6] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*.

[7] Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G. (1999). The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359.

[8] Bills, C., Chen, J., and Saxena, A. (2011). Autonomous MAV flight in indoor environments using single image perspective cues. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[9] Boissonnat, J.-D. and Yvinec, M. (1998). *Algorithmic Geometry*. Cambridge University Press, UK.

[10] Bouabdallah, S., Becker, M., Perrot, V., and Siegwart, R. (2007). Toward obstacle avoidance on quadrotors. In *Proceedings of the International Symposium on Dynamic Problems of Mechanics*.

[11] Call, B., Beard, R., Taylor, C., and Barber, B. (2006). Obstacle avoidance for unmanned air vehicles using image feature tracking. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*.

[12] Collins, R. T. (1996). A space-sweep approach to true multi-image matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[13] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(6):1052–1067.

[14] Dryanovski, I., Morris, W., and Xiao, J. (2010). Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[15] Dryanovski, I., Morris, W., and Xiao, J. (2011). An open-source pose estimation system for micro-air vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[16] Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15.

[17] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

[18] Fowers, S. G., Lee, D.-J., Tippetts, B. J., Lillywhite, K. D., Dennis, A. W., and Archibald, J. K. (2007). Vision aided stabilization and the development of a quad-rotor micro UAV. In *Computational Intelligence in Robotics and Automation (CIRA)*.

[19] Geiger, A., Roser, M., and Urtasun, R. (2010). Efficient large-scale stereo matching. In *Asian Conference on Computer Vision (ACCV)*.

[20] Hartley, R. (1997). In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 19(6):580–593.

[21] Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition.

[22] Herbert, M., Caillas, C., Krotkov, E., Kweon, I. S., and Kanade, T. (1989). Terrain mapping for a roving planetary explorer. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[23] Hofer, M., Wendel, A., and Bischof, H. (2013). Line-based 3D reconstruction of wiry objects. In *Computer Vision Winter Workshop (CVWW)*.

[24] Hoppe, C., Klopschitz, M., Rumpler, M., Wendel, A., Kluckner, S., Bischof, H., and Reitmayr, G. (2012a). Online feedback for structure-from-motion image acquisition. In *British Machine Vision Conference (BMVC)*.

[25] Hoppe, C., Wendel, A., Zollmann, S., Pirker, K., Irschara, A., Bischof, H., and Kluckner, S. (2012b). Photogrammetric camera network design for micro aerial vehicles. In *Computer Vision Winter Workshop (CVWW)*.

[26] Hrabar, S. (2008). 3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[27] Irschara, A., Rumpler, M., Meixner, P., Pock, T., and Bischof, H. (2012). Efficient and globally optimal multi view dense matching for aerial images. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*.

[28] Katusic, M. (2012). *Human-Inspired Visual Servoing for Automatic Take-Off, Hovering and Landing of MAVs*. Master Thesis, Graz University of Technology, Austria.

[29] Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[30] Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*.

[31] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *International Symposium on Mixed and Augmented Reality (ISMAR)*.

[32] Lee, J.-O., Lee, K.-H., Park, S.-H., Im, S.-G., and Park, J. (2011). Obstacle avoidance for small UAVs using monocular vision. *Aircraft Engineering and Aerospace Technology*, 83(6):397–406.

[33] Loop, C. and Zhang, Z. (1999). Computing rectifying homographies for stereo vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[34] Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJVC)*, 60(2):91–110.

[35] Meagher, D. (1982). Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147.

[36] Meier, L., Tanskanen, P., Heng, L., Lee, G. H., Fraundorfer, F., and Pollefeys, M. (2012). PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1-2):21–39.

[37] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI National Conference on Artificial Intelligence*.

[38] Nistér, D. (2004). An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(6):756–770.

[39] Pan, Q., Reitmayr, G., and Drummond, T. (2009). ProFORMA: Probabilistic feature-based on-line rapid model acquisition. In *British Machine Vision Conference (BMVC)*.

[40] Pupilli, M. and Calway, A. (2005). Real-time camera tracking using a particle filter. In *British Machine Vision Conference (BMVC)*.

[41] Roberson, R. E. and Schwertassek, R. (1988). *Dynamics of multibody systems*. Springer Verlag.

[42] Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. (2013). Learning monocular reactive UAV control in cluttered natural environments. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[43] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV)*.

[44] Rumpler, M., Irschara, A., and Bischof, H. (2011). Multi-view stereo: Redundancy benefits for 3D reconstruction. In *Proceedings of the Annual Workshop of the Austrian Association for Pattern Recognition*.

[45] Rumpler, M., Wendel, A., and Bischof, H. (2013). Probabilistic range image integration for DSM and true-orthophoto generation. In *Scandinavian Conference on Image Analysis (SCIA)*.

[46] Szeliski, R. (2011). *Computer Vision: Algorithms and Applications*. Springer Verlag.

[47] Triebel, R., Pfaff, P., and Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[48] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (2000). Bundle adjustment – A modern synthesis. In *Vision Algorithms: Theory and Practice*, pages 298–372. Springer Verlag.

[49] Wendel, A., Maurer, M., Graber, G., Pock, T., and Bischof, H. (2012). Dense reconstruction on-the-fly. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[50] Wilhelms, J. and Van Gelder, A. (1992). Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227.

[51] Wu, C. (2007). SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). *http://cs. unc. edu/˜ccwu/siftgpu*.

[52] Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. (2010). OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[53] Yguel, M., Aycard, O., and Laugier, C. (2008). Update policy of dense maps: Efficient algorithms and sparse representation. In *Field and Service Robotics*, pages 23–33. Springer Verlag.

[54] Zach, C., Pock, T., and Bischof, H. (2007). A duality based approach for realtime TV-L1 optical flow. In *Pattern Recognition*, volume 4713, pages 214–223. Springer Verlag.