Stereo Vision for Autonomous Micro Aerial Vehicles

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Konstantin Schauwecker, M.Sc.

aus Tübingen

Tübingen

2014

Tag der mündlichen Qualifikation:	11.07.2014
Dekan:	Prof. Dr. rer. nat. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. rer. nat. Andreas Zell
2. Berichterstatter:	Prof. Dr. sc. nat. Reinhard Klette, FRSNZ

Abstract

Small unmanned and lightweight aircrafts, known as Micro Aerial Vehicles (MAVs), have gained much attention in recent years. In this thesis we approach the problem of enabling such MAVs to fly autonomously without the need for human intervention. The sensor technology that is chosen for this task is stereo vision. As research platform for this work serves a small quadrotor MAV that has been equipped with four cameras in two stereo configurations. We study a broad range of problems that need to be solved for the construction of a stereo vision based autonomous MAV.

The first problem that we examine is stereo matching. We introduce a new sparse stereo matching algorithm that achieves very high processing rates while also delivering accurate results. A key component of this algorithm is a combined consistency and uniqueness check that evaluates a dense disparity range. This new stereo algorithm is used for processing the imagery of both stereo camera pairs that are available on the used MAV platform. For the first camera pair that is facing forward, we process the stereo matching results with a simplified Simultaneous Localization and Mapping (SLAM) algorithm, which tracks the cameras' pose (*i.e.* position and orientation).

A different method is applied to the second stereo camera pair that is facing downwards. Here, the stereo matching results are used for detecting the dominant ground plane. From this plane and a method based on frame-to-frame tracking, we are able to derive another estimate of the MAV's pose. Both pose estimates are then fused and used for controlling the MAV's flight. The ability of this MAV to fly autonomously is demonstrated in several flight experiments and evaluations. We successfully demonstrate autonomous take-off, landing, hovering, 360° yaw rotation, shape flight and error recovery.

Finally, we examine the problem of sensing free and occupied space, which would be needed to facilitate autonomous path planning for our MAV. For this purpose, we extend an existing volumetric occupancy mapping method, such that it provides more robust results when used in conjunction with stereo vision. The performance improvement is mainly achieved by introducing a more complex update mechanism for voxels in this map, which considers the probability that a voxel is currently visible. Furthermore, the expected depth error is modeled and considered during map updates, and the overall run-time performance of the method is improved. The resulting method is fast enough to perform occupancy mapping in real-time, including the necessary dense stereo matching.

Kurzfassung

Kleine, unbemannte und leichte Flugzeuge, bekannt als Micro Aerial Vehicles (MAVs), haben in jüngerer Vergangenheit viel Aufmerksamkeit erfahren. In dieser Dissertation befassen wir uns mit dem autonomen Flug von MAVs, bei welchem diese agieren, ohne dass ein menschliches Eingreifen notwendig ist. Die hierfür in dieser Dissertation gewählte Sensor-Technologie sind Stereo-Kameras. Als Forschungsplattform dient ein Quadrocopter-MAV, welches mit vier Kameras ausgestattet wurde, die zu zwei Stereo-Paaren angeordnet sind. Wir befassen uns mit einer breiten Sammlung von Problemen, die es zur Konstruktion eines stereobasierten autonomen MAVs zu lösen gilt.

Das erste dieser Probleme, das wir untersuchen, ist Stereo-Matching. Wir stellen einen neuartigen Sparse-Stereo-Algorithmus vor, welcher sehr hohe Verabeitungsgeschwindigkeiten erreicht und dennoch akkurate Ergebnisse liefert. Die Schlüsselkomponente dieses Algorithmus ist ein kombinierter Konsistenz- und Einzigartigkeitstest, welcher den gültigen Disparitätsbereich lückenlos prüft. Dieser neuartige Stereo-Algorithmus wird für die Verarbeitung der Bilddaten beider Stereo-Kamerapaare eingesetzt. Die Ergebnisse für das nach vorne schauende Kamerapaar werden dann mit einem vereinfachten SLAM-Algorithmus (Simultaneous Localization and Mapping) verarbeitet, welcher Änderungen der Kamerapose (d.h. Position und Ausrichtung) verfolgt.

Eine andere Methode wird für die Auswertung des zweiten, nach unten gerichteten, Kamerapaares verwendet. In diesem Fall werden die Ergebnisse des Stereo-Matchings für die Detektion der Bodenebene genützt. Mittels dieser Ebene und einem zweiten, auf Bildverfolgung basierten Verfahrens, lässt sich eine weitere Schätzung für die Pose des MAVs ermitteln. Beide Schätzungen werden anschließend fusioniert und zur Steuerung des MAVs verwendet. Die autonomen Flugfähigkeiten werden mittels verschiedener Flugtests und Untersuchungen demonstriert. Gezeigt werden autonomer Start, Landung, Schwebeflug, 360° Drehung, Figurenflug, sowie die selbstständige Fehlerkompensation.

Zum Abschluss untersuchen wir das Problem der Wahrnehmung von Freiräumen und Hindernissen, das zur autonomen Pfadplanung notwendig ist. Hierfür erweitern wir ein existierendes volumetrisches Verfahren für Occupancy Mapping. Das Verfahren wird modifiziert, sodass es robustere Ergebnisse bei der Verarbeitung von Stereodaten liefert. Den Hauptbeitrag leistet hierbei eine neue Methode zu Aktualisierung der Belegtwahrscheinlichkeit eines in der Karte gespeicherten Voxels. Diese Methode berücksichtigt die Wahrscheinlichkeit, dass ein Voxel gerade sichtbar ist. Des Weiteren modellieren wir den zu erwartenden Tiefenfehler und berücksichtigen diesen bei der Kartenaktualisierung. Außerdem verbessern wir die Verarbeitungsgeschwindigkeit dieses Verfahrens, wodurch eine Echtzeitverarbeitung inklusive Stereo-Matching möglich wird.

Acknowledgements

I would like to thank Sebastian Scherer for his work on setting up the MAV platform that served as a base for the autonomous MAV presented in this thesis. He was also of great help in resolving the numerous problems that occurred with the MAV's hard- and software during this work. I would further like to thank him for sharing his PTAM extensions and for his collaboration on adapting PTAM for on-board processing on our MAV. Next, I would like to thank Shaowu Yang for the interesting discussions and his help on several problems that occurred with the employed MAV platform. I also thank him for inviting me to be his co-author in two publications on his monocular vision based autonomous MAV. I thank my supervisor Prof. Andreas Zell for his help and guidance during my research and for allowing me to work on this interesting research topic. Further thanks go to my co-supervisor Prof. Reinhard Klette, who is the person that initially inspired me to work in computer vision. My special thanks go to my girlfriend Nan Rosemary Ke for her support and the big compromises she made while I was working on this thesis. Also, I thank my mother, Annemarie Schauwecker, for her support and care during this time. Finally, I would like to thank my employer STZ-Softwaretechnik, and in particular Prof. Joachim Goll, for enabling me to work part time, which has funded me throughout this thesis.

Konstantin Schauwecker

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Contributions . . . 3
- 1.3 Outline . . . 5

2 Background **7**
- 2.1 Micro Aerial Vehicles . . . 7
  - 2.1.1 Types of MAVs and Principal Axes . . . 8
  - 2.1.2 Quadrotors . . . 9
  - 2.1.3 Research Platform . . . 11
- 2.2 Stereo Vision . . . 14
  - 2.2.1 Stereo Geometry . . . 14
  - 2.2.2 Image Rectification . . . 17
  - 2.2.3 Challenges . . . 20

3 Sparse Stereo Vision **21**
- 3.1 Introduction . . . 21
- 3.2 Related Work . . . 22
  - 3.2.1 Feature Detection . . . 23
  - 3.2.2 Dense Stereo Methods . . . 27
  - 3.2.3 Sparse Stereo Methods . . . 29
- 3.3 Feature Detection . . . 31
  - 3.3.1 Adaptive Threshold . . . 31
  - 3.3.2 Averaged Center . . . 34
- 3.4 Stereo Matching . . . 35
  - 3.4.1 Correlation . . . 35
  - 3.4.2 Dense Consistency and Uniqueness Check . . . 37
  - 3.4.3 Processing of Unrectified Stereo Images . . . 39
- 3.5 Evaluation . . . 42
  - 3.5.1 Feature Detector Performance . . . 43
  - 3.5.2 Combined Feature Detection and Stereo Matching . . . 45
  - 3.5.3 Comparison with Other Stereo Matching Methods . . . 48
  - 3.5.4 Real World Performance Evaluation . . . 50
- 3.6 Summary and Discussion . . . 52

4 Stereo-Based Visual Navigation **55**
4.1 Introduction . . . 55
4.2 Related Work . . . 56
4.2.1 Visual Odometry . . . 56
4.2.2 Visual Simultaneous Localization and Mapping . . . 58
4.2.3 Autonomous Navigation for MAVs . . . 60
4.3 Approach Using One Camera Pair . . . 63
4.3.1 Feature Detection and Stereo Matching . . . 64
4.3.2 Local SLAM . . . 66
4.3.3 Sensor Fusion and Motion Feedback . . . 69
4.3.4 Evaluation . . . 70
4.4 Approach Using Two Camera Pairs . . . 73
4.4.1 Problems of Single-Stereo Solution . . . 74
4.4.2 Performance Improvements . . . 75
4.4.3 Processing Method for the Downward-Facing Cameras . . . 76
4.4.4 Sensor Fusion and Control . . . 81
4.4.5 Drift Correction . . . 82
4.4.6 Recovery . . . 84
4.4.7 Evaluation . . . 85
4.5 Summary and Discussion . . . 96

5 Stereo-Based Obstacle and Environment Perception **99**
5.1 Introduction . . . 99
5.2 Related Work . . . 101
5.2.1 2D Occupancy Maps . . . 101
5.2.2 2.5D Elevation Maps . . . 102
5.2.3 3D Occupancy Maps . . . 103
5.2.4 Occupancy Mapping with Autonomous MAVs . . . 104
5.3 Method . . . 105
5.3.1 Visibility Estimation . . . 107
5.3.2 Occupancy Probability Integration . . . 110
5.3.3 Sensor Depth Error Modeling . . . 112
5.3.4 Performance Considerations and Optimizations . . . 113
5.4 Evaluation . . . 115
5.4.1 Map Quality Analysis . . . 116
5.4.2 Run-Time Performance Evaluation . . . 119
5.4.3 Mapping of MAV Environment . . . 121
5.5 Summary and Discussion . . . 123

6 Summary and Conclusions **125**

Bibliography **129**

Nomenclature

AGAST	Adaptive and Generic Accelerated Segment Test, page 25
BA	Bundle Adjustment, page 57
BM	Block Matching, page 28
BMP	Bad Matches Percentage, page 46
BP	Belief Propagation, page 28
BRIEF	Binary Robust Independent Elementary Features, page 26
DoF	Degrees of Freedom, page 81
DoG	Difference of Gaussians, page 26
DP	Dynamic Programming, page 29
EKF	Extended Kalman Filter, page 58
ELAS	Efficient LArge-scale Stereo, page 105
ESM	Efficient Second order Minimization, page 70
FAST	Features from Accelerated Segment Test, page 23
GC	Graph Cut, page 28
IMU	Inertial Measurement Unit, page 11
LoG	Laplacian of Gaussian, page 30
MAV	Micro Aerial Vehicle, page 7
MRF	Markov Random Field, page 28
NCC	Normalized Cross Correlation, page 35
ORB	Oriented FAST and Rotated BRIEF, page 26
PF	Particle Filter, page 58
PTAM	Parallel Tracking and Mapping, page 59
RANSAC	RANdom SAmple Consensus, page 57
RMS	Root Mean Square, page 33
RMSE	Root Mean Square Error, page 71
ROI	Region Of Interest, page 44
ROS	Robot Operating System, page 13
SAD	Sum of Absolute Differences, page 35
SGM	Semi-Global Matching, page 29
SIFT	Scale Invariant Feature Transform, page 26
SLAM	Simultaneous Localization and Mapping, page 58
SSD	Sum of Squared Differences, page 35
SURF	Speeded-Up Robust Features, page 26
SUSAN	Smallest Univalue Segment Assimilating Nucleus, page 23
UAV	Unmanned Aerial Vehicle, page 7

VO	Visual Odometry, page 56
ZMSAD	Zero-Mean Sum of Absolute Differences, page 35
ZMSSD	Zero-Mean Sum of Squared Differences, page 35

Symbols

$\|S\|$	Cardinality of a set S
$\|\|\mathbf{a}\|\|$	Vector norm of $\mathbf{a}$
$[a,b]$	Interval between a and b, including a and b
(a,b)	Interval between a and b, excluding a and b; point with coordinates a and b
$\{a,b\}$	Set containing a and b
$\{S\|c(S)\}$	All elements of set S that meet condition $c(S)$
$\lfloor a \rfloor$	Largest integer not greater than a (floor function)
$\wedge$	Logical conjunction
$\neg$	Negation
∞	Infinity
$\emptyset$	Empty set
a	Feature detection adaptivity factor
$\mathbf{a}$	Vector
$\mathbf{a}^T$	Vector transpose
$\mathbf{A}$	Matrix
b	Baseline distance; bitstring
c	Matching cost
C	Camera; set of matching costs
C_v	Event that voxel v is visible
d	Disparity value
d_{max}	Maximum disparity
Δ	Difference or distance
$\Delta_h(a,b)$	Bitwise Hamming distance between a and b
ε	Real number greater than zero
$\in$	Set membership
f	Focal length; function
F	Cumulative probability distribution for distribution function f
h	Height
H	Hit event
$\mathbf{H}$	Homography matrix
$i,\ j, m, n$	Natural numbers
I	Image
I_v	Event that voxel v is inside an obstacle

κ, λ	Weighting, scaling or interpolation factors
l	Pyramid level; length
$L(A)$	Logarithm of the odds-ratio (log-odds) for event A
M	Occupancy measurement (H or $\neg H$)
N	Image or volume neighborhood
O	Projection center
O_v	Event that voxel v is occupied
p, q	Probabilities
p, q, r	Points in $\mathbb{R}^2$ or $\mathbb{R}^3$
$\mathrm{P}(A)$	Probability of event A
$\mathrm{P}(A\|B)$	Conditional probability of event A given B
ψ	Bad matches percentage
q	Uniqueness factor
r	Residual
R	Ray of voxels
ρ	Feature detection repeatability
s	Clusteredness measure
S	Set
$S_1 \setminus S_2$	Set S_1 excluding elements in S_2
$\overline{S}$	Set average
σ	Standard deviation
σ^2	Variance
t	Threshold
Θ, Φ, Ψ	Pitch, roll and yaw angle
τ	Contrast measure
u, v	Horizontal and vertical image coordinates
u_L, v_L	Horizontal and vertical left image coordinates
u_R, v_R	Horizontal and vertical right image coordinates
$\tilde{u}$, $\tilde{v}$	Rectified horizontal and vertical image coordinates
v	Voxel
V_v	Event that voxel v is visible
w	Step width
x, y, z	World coordinates in $\mathbb{R}^3$
ξ	Census transformation comparison function

Chapter 1

Introduction

1.1 Motivation

Small unmanned aircrafts, known as Micro Aerial Vehicles (MAVs), have received much research and industry attention in recent years. The probably most evident application of MAVs is military reconnaissance, where MAVs such as the Honeywell RQ-16 T-Hawk (see Honeywell International, Inc., 2014) are already used today. There exists, however, an increasing number of civilian applications. For example, after the 2011 disaster at the Fukushima nuclear power plant in Japan, a military MAV was used to examine the damages of the reactor buildings. This was done while the radiation levels where too high to allow any human workers to enter the disaster area.

Another civilian application of MAVs is the visual inspection of structures that are otherwise hard to access. Examples are bridges, dams, chimneys, or wind turbines. While the manual inspection of such structures is usually difficult and time-consuming, such tasks can be performed faster and cheaper with an MAV that has been equipped with a high-resolution camera. Such an inspection service is *e.g.* offered by the company Fly & Check by Drone (see Fly & Check by Drone, 2014).

What these applications have in common is that they require an operator who controls the MAV. Amazon, however, recently announced the use of autonomous MAVs for the fast delivery of online orders (see Amazon.com, Inc., 2013). This service has been called Amazon Prime Air and is claimed to facilitate delivery within 30 minutes from the time of order. For this being feasible, the delivery address needs to be within a range of 10 miles from an Amazon warehouse. Amazon has claimed that this service might be available as early as 2015.

Following Amazon's announcement of Prime Air, logistics companies such as UPS, FedEx and DHL quickly announced similar plans (see Popper, 2013; Lang, 2013). Unfortunately, no details are known about any of these MAV projects. It is, however, very likely that the current prototypes of these MAVs primarily rely on GPS for navigating themselves to the designated delivery address. This conjecture is supported by the fact that pictures of Amazon's MAV do not show any apparent sensors.

Unfortunately, GPS is not yet accurate enough to allow MAVs to fly autonomously in urban environments. The position estimate provided by GPS can be several meters off

from the true location. In a densely populated area, this means that an MAV delivering a parcel might drop this parcel at the wrong house or at an inaccessible or dangerous location such as a rooftop or on the street. Furthermore, when flying in urban environments, buildings pose a significant problem for GPS receptions. Buildings might shadow satellite signals or reflect the signals, which then travel to the receiver on a path that is longer than the expected line of sight. Thus, in urban environments GPS position estimates might be far off from the true location, or GPS localization might fail altogether.

The reception of erroneous location information, or the lack of it for a considerable time span, can result in fatal consequences for an autonomous MAV. Thus, for an MAV to reach its destination safely and reliably, it cannot depend on GPS alone. Instead, additional sensors are required to allow navigation even in cases when GPS location information is unavailable. Furthermore, an autonomous MAV should be able to sense its environment, such that it can avoid obstacles in situations where it deviates too far from the planned trajectory. But even if the MAV remains close to the intended trajectory, it should be able to sense unexpected obstacles that were not known during trajectory planning. Only if an MAV is able to actively avoid collisions, it can be considered safe for use in densely populated urban environments.

The choice of additional sensors with which an MAV can be equipped is limited by the maximum payload and power consumption constraints imposed by the MAV platform. For wheeled robots, laser scanners are a popular sensor choice, as they offer an accurate 3D perception of the robot's environment, and can be used for robust and accurate localization. However, laser scanners have a considerable weight and power consumption, which makes it critical to employ them on MAVs. This is particularly true for laser scanners that use several beams to obtain measurements from more than one sensing plane.

An alternative to laser scanners are vision-based methods. Compared to laser scanners, cameras are generally much lighter and have a lower power consumption. A single camera can be used for 3D localization, as has *e.g.* been shown by Klein and Murray (2007). While in this case, the current camera position can only be observed with respect to an unknown scaling factor, it is possible to track the true metric position when using stereo vision (*i.e.* two cameras). In case of stereo vision, we also gain information on the 3D position of objects within the field of view. This information can be used to facilitate obstacle avoidance or interaction with known objects in the MAV's environment. Another advantage of cameras over laser scanners is that cameras can be produced at much lower costs. This allows them to be employed even on low-cost MAVs that are intended for the consumer market.

In this thesis we[1] thus focus on stereo-vision based methods that facilitate the construction of autonomous MAVs. In particular, we aim at achieving the following two

[1]Following common practices as formulated by Knuth *et al.* (1989), I use "we" rather than "I" in this thesis for inviting the reader to be part of my presentation. However, I confirm that I, Konstantin Schauwecker, was the sole author of this thesis.

tasks using stereo vision: enabling an MAV to track its current pose and enabling the MAV to map obstacles and other non-traversable space. Solving these two problems would allow us to enhance today's GPS-controlled MAVs. A vision-based pose estimate could be used to bridge periods where GPS is temporally unavailable. Vision-based environment mapping, on the other hand, could allow the MAV to detect and avoid obstacles that appear on the intended flying trajectory.

An autonomous MAV that can fly using only vision-based information could facilitate a range of new applications. For example, due to the dependency on a radio link, the remote controlled MAV that was used for inspecting the Fukushima disaster site was not able to enter the reactor buildings. If an autonomous vision-based MAV would have been available at the time, it could have been used for entering the damaged reactor buildings and provide detailed information on the occurred damage. Similarly, such an MAV could be employed for search and rescue missions in other damaged buildings that are not safe to be entered by humans. But also more commonplace tasks could be solved with the help of vision-based autonomous MAVs. An MAV capable of safe autonomous indoor flight could be used to transport items between different parts of a building. As an example, such MAVs could be employed in hospitals to transport drugs to the places where they are currently needed.

As we can see, there exists a large variety of tasks that could one day be performed by autonomous MAVs. Before this vision comes true, however, much research needs to be done on the technology behind autonomous flights. This thesis hopefully provides a contribution towards this goal.

1.2 Contributions

An autonomous MAV that relies on stereo vision has to employ a range of different vision-based methods and techniques. Hence, this thesis makes contributions to several fields that are all part of the computer vision domain. The first contribution is a new algorithm for detecting image features, which is based on the popular FAST detector from Rosten and Drummond (2006). The features detected by this method are more evenly distributed over the input images than the features provided by a standard FAST algorithm. This new feature detector has proven to be particularly well suited for sparse stereo matching.

The next contribution is a new sparse stereo matching algorithm. While this algorithm has shown to be very efficient, it also provides robust and accurate stereo matching results. This is due to the fact that the algorithm employs a 'dense consistency and uniqueness check', which is an efficient post-processing method for filtering erroneous matches. Furthermore, the proposed algorithm includes an efficient method for processing input images that have not previously been rectified to compensate lens distortions and camera alignment errors. This new stereo matching algorithm has been published alongside the mentioned feature detector at the 2012 IEEE/RSJ International Conference

on Intelligent Robots and Systems (IROS) in Vilamoura, Portugal (Schauwecker *et al.*, 2012a).

Using the mentioned feature detector and sparse stereo matching algorithms, it was possible to construct an autonomous MAV that only relies on stereo vision and inertial measurements. The first prototype of this MAV, which was presented at the 2012 Autonomous Mobile Systems Conference (AMS) in Stuttgart, Germany (Schauwecker *et al.*, 2012b), featured a forward facing camera pair. The MAV uses the matched features from our stereo matching algorithm as input for a simplified Simultaneous Localization and Mapping (SLAM) algorithm based on the method proposed by Scherer *et al.* (2012). This SLAM system provides the MAV with its current pose, which enables it to achieve controlled and stable flight. To the author's knowledge, this was the first demonstration of an MAV that performs stereo matching on-board, and is able to use the gained stereo matching results for visual navigation.

In another revision, this first prototype was extended to include an additional downward-facing camera pair. This MAV was presented at the 2013 International Conference on Unmanned Aircraft Systems (ICUAS) in Atlanta, USA and in the Journal of Intelligent and Robotic Systems (JINT) (Schauwecker and Zell, 2013, 2014a). Features detected in the imagery from this camera pair are again used for stereo matching. The MAV is able to obtain an additional estimate of its current pose, by detecting and tracking the ground plane using the stereo matching results and the camera imagery. This additional pose estimate is then fused with the estimates gained from the forward facing cameras. The resulting MAV has been thoroughly evaluated in several flight experiments, where it demonstrated its autonomous flying capabilities. Compared to the first prototype, this MAV is able to achieve a more precise and robust autonomous flight. To the author's knowledge, this was the first demonstration of an MAV that is able to perform stereo matching on-board and in real-time for two camera pairs.

Finally, the last contribution of this thesis is in the field of occupancy mapping. The knowledge of free and occupied space is necessary for an autonomous MAV in order to facilitate autonomous path planning. In this thesis, the popular OctoMap method for volumetric occupancy mapping (Wurm *et al.*, 2010; Hornung *et al.*, 2013) is extended, in order to improve processing results for noisy measurements as obtained from stereo matching. This is made possible by considering whether or not a given voxel in the map should be visible from the current camera location. Furthermore, we model the expected depth error of the stereo matching results, and respect this error when performing map updates. By applying a code-level optimization, the resulting occupancy mapping system is able to run in real-time on a commodity PC. Although the MAV used in this thesis does not yet provide sufficient processing resources to perform occupancy mapping on-board, a map can be created off-board after the MAV finished an autonomous flight. This method was first presented on the 2014 IEEE International Conference on Robotics and Automation (ICRA) in Hong Kong, China (Schauwecker and Zell, 2014b).

1.3 Outline

The remaining parts of this thesis are structured as follows: Chapter 2 provides an introduction to basic concepts and current technologies that are relevant for this thesis. In particular, this chapter looks at the current state of MAV technology and provides an introduction to stereo vision. Chapter 3 presents the new feature detector and sparse stereo matching algorithm that are proposed in this thesis. Both methods are evaluated in several experiments on different evaluation datasets. Chapter 4 presents two autonomous MAVs that are based on our new feature detector and stereo matching algorithm. The first MAV uses only a forward facing camera pair and serves as a prototype for demonstrating the feasibility of a stereo vision based autonomous MAV. This prototype is then extended with another camera pair that is facing downwards. While only a brief evaluation is performed for the first prototype, a detailed evaluation of the extended revision of this MAV is provided, which includes several flight and offline-processing experiments. Chapter 5 presents the new volumetric occupancy mapping method proposed in this thesis. This method is evaluated on a publicly available dataset and on data recorded by our autonomous MAV. Finally, the work presented in this thesis is summarized and concluded in Chapter 6.

Chapter 2

Background

This chapter is dedicated to fundamentals that might be necessary for the understanding of this thesis. It provides an overview on MAV technology, including the hardware platform that has been used for implementing the methods presented in this thesis. Given that our MAV should achieve autonomous flight using stereo cameras, we also discuss the basic principles and techniques of stereo vision.

2.1 Micro Aerial Vehicles

Micro Aerial Vehicle (MAV), or Micro Air Vehicle, is a general term that refers to small and light unmanned aircraft systems. Unfortunately, there exists no general agreement up to which size an aircraft should be considered to be an MAV. As a general guideline, the International Micro Air Vehicles Conference (see IMAV, 2013) imposed a weight limit of 2 kg and a maximum size of 1 m for all aircrafts participating in its 2013 flight competition. The size of an aircraft was defined as either the aircraft's wingspan or the largest horizontal rotor-to-rotor distance.

There exist, hover, MAVs with dimensions that are far below this size limit. One extreme example is the robotic fly developed by Wood (2008). This small biologically inspired MAV weights only 60 mg and its two wings have a length of only 1.5 cm. While this MAV is externally powered and only able to fly along two guiding wires, it is an impressive demonstration of today's design and manufacturing capabilities.

The transition between MAVs and larger scale Unmanned Aerial Vehicles (UAVs) is gradual. On the larger end of the size scale are military drones that can reach the size of an average jet plane, such as the Global Hawk that is built by Northrop Grumman (see Northrop Grumman Corp., 2014). Due to the lack of an exact definition for MAVs, it is not always possible to find an objective category for a given aircraft. The aircrafts that we are focusing on in this thesis, however, have a size that is well below the IMAV size restrictions, which should allow us to distinctly label them as MAVs.

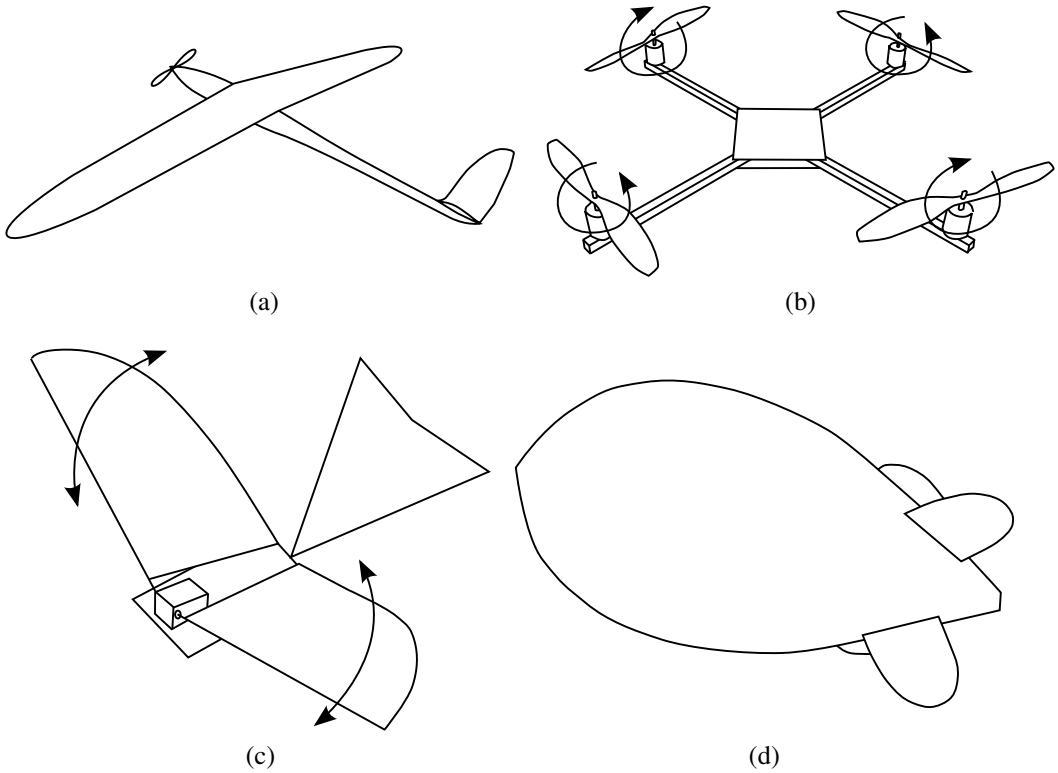


Figure 2.1: Illustration of (a) fixed wing, (b) rotary wing, (c) flapping wing and (d) lighter-than-air MAV.

2.1.1 Types of MAVs and Principal Axes

MAVs can usually be divided into four distinct categories. An example for each has been illustrated in Figures 2.1a–2.1d. The first category are *fixed wing* MAVs, which are model-sized airplanes that usually require a runway for take-off and landing. These MAVs can achieve high air speeds, which allow them to cover long distances. Commonly, such MAVs receive their propulsion from a propeller, but alternative methods such as small-scale turbine engines also exist.

The next category are *rotary wing* MAVs that are characterized by one or more rotors, which each consists of several rotor blades. Typically, these MAVs have an even number of rotors, such that they can easily compensate each other's torque. For example, even though a traditional helicopter design only has one main rotor, the main rotor's torque is compensated by the much smaller tail rotor. Common numbers of rotors used for rotary wing MAVs are two, four, six and eight, but other numbers are also possible. Rotary wing MAVs are usually unable to achieve speeds that are comparable to those of fixed-wing MAVs. However, rotary wing MAVs have the advantage that they can take-off and land vertically and move at low speeds, including hovering. In the next section, we have

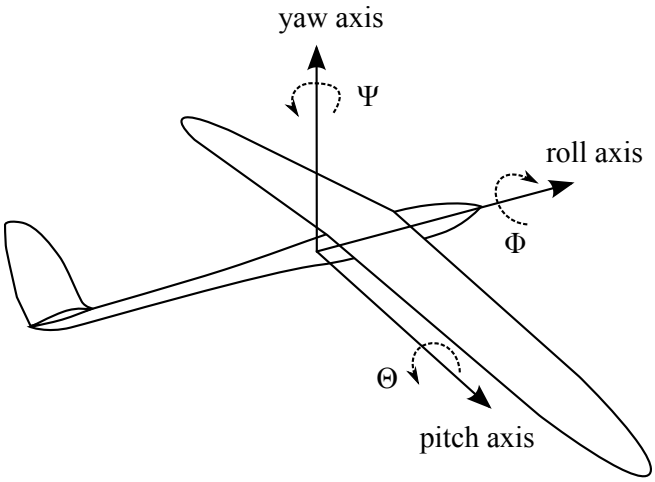


Figure 2.2: Aircraft principal axes and angles of rotation.

a close look at rotary wing MAVs with four rotors, which are known as *quadrotors*.

Flapping wing MAVs achieve propulsion by moving their wings similar to the movements of birds or insects. While the robotic fly of Wood is an extremely small member of this category, flapping wing MAVs can also be of larger scale. One example is the Festo SmartBird (Festo AG & Co. KG, 2011), which has a wingspan of 2 m. Just like rotary wing MAV's, some flapping wing MAVs such as the robotic hummingbird developed by Keennon *et al.* (2012), are able to take-off and land vertically and hover in one place.

Finally, our last MAV category are *lighter-than-air* MAVs or *blimps*. These MAVs receive their vertical thrust form a body that is filled with a lighter-than-air gas. While these MAVs are very energy efficient, they also tend to be large and are only capable of slow movements. Just like fixed-wing MAVs, these MAVs are commonly equipped with propellers to facilitate their propulsion.

With the exception of blimps, all MAV types can rotate in three dimensions. To describe the orientation of an MAV, we use the three principal axes that are shown in Figure 2.2. These axes are called *roll* axis, *pitch* axis and *yaw* axis. Accordingly, the rotation angles around these axes are named pitch angle Θ, roll angle Φ, and yaw angle Ψ. In order to achieve a stable flight, it is particularly important to control the MAV's pitch and roll angles Θ and Φ.

2.1.2 Quadrotors

A *quadrotor* is a rotary wing aircraft that is equipped with four rotors with vertically directed airflow. The first quadrotor in history was built by Étienne Œhmichen in 1922 (see Seddon and Newman, 2011). While this aircraft was only able to fly for a distance of a few hundred meters, it set a new flight record for rotary wing aircrafts at its time. More capable quadrotor aircrafts were constructed at later dates, such as the transport aircraft

Curtiss-Wright X-19 from 1963, which however never made it past the experimental stage (see Ranson, 2002).

Compared to a helicopter, a quadrotor can be built with much simpler rotor mechanics. While a helicopter has to be able to change the angles of its rotor blades in order to achieve a controlled flight, this is not necessary for quadrotors. Instead, fixed rotor blades can be used for all four rotors, which greatly simplifies the rotor construction. Each rotor contributes an individual thrust and torque. Hence, a quadrotor can influence its roll, pitch, yaw and thrust, by adjusting the rotational speed of its four rotors. This, however, requires precise control of the rotational speed of each rotor, which is simple for an electrically powered quadrotor with an electric motor for each rotor. For a quadrotor with a conventionally powered central engine, this is not easily feasible. In this case, adjustable rotor blades can again be employed, which however annihilates the advantage of a simplified rotor design.

The fact that quadrotors can be constructed with fixed rotor blades makes this design particularly interesting for electrically powered MAVs. In this case, the only required actuators are the electric engines for powering each rotor. This circumstance and the fact that quadrotors do not require wings, allow for the construction of quadrotors at very small scales. The probably smallest example is the externally powered Mesicopter built by Kroo and Prinz (2001), which has a rotor diameter of only 1.5 cm. Flight tests of this prototype were, however, limited to a test bed with a constraining arm.

Quadrotors are usually constructed in either a plus- or an X-configuration, as shown in Figure 2.3a and 2.3b. These two configurations differ in the assumed forward direction, which has an influence on the rotor control. The control of a quadrotor with plus configuration is simpler, as only the rotational speed of one rotor pair has to be adjusted in order to influence the quadrotor's roll, pitch or yaw. For a quadrotor with X-configuration, on the other hand, we are required to always control the rotational speed of all four rotors. The advantage of an X-configuration is, however, that no rotor is blocking the field of view when carrying a forward-facing camera.

For simplicity, we focus on the plus-configuration in this chapter. In both configurations, however, a rotor always rotates in the same direction as the rotor opposite to it, and in the opposite direction of the two rotors next to it. This way, the torque of the rotors rotating clockwise is neutralized by the torque of the rotors rotating counter-clockwise, which allows the quadrotor to hover without spinning.

The pitch of a quadrotor with plus configuration can be influenced by adjusting the rotational speed of the left and right rotors in relation to the rotational speed of the front and back rotors. Because of the unequal rotational speed of the clockwise and counter-clockwise rotors in this case, the overall torque is no longer zero. Thus, the quadrotor experiences a torque around its pitch axis, which allows us to control the quadrotor's pitch angle. Similarly, we can influence roll by adjusting the rotational speed of the front and back rotors in relation to the left and right rotors. For influencing the quadrotor's yaw, we lower the rotational speed of either the left or the right rotor, while increasing the rotational speed of the opposite rotor. A detailed description and analysis of rotor

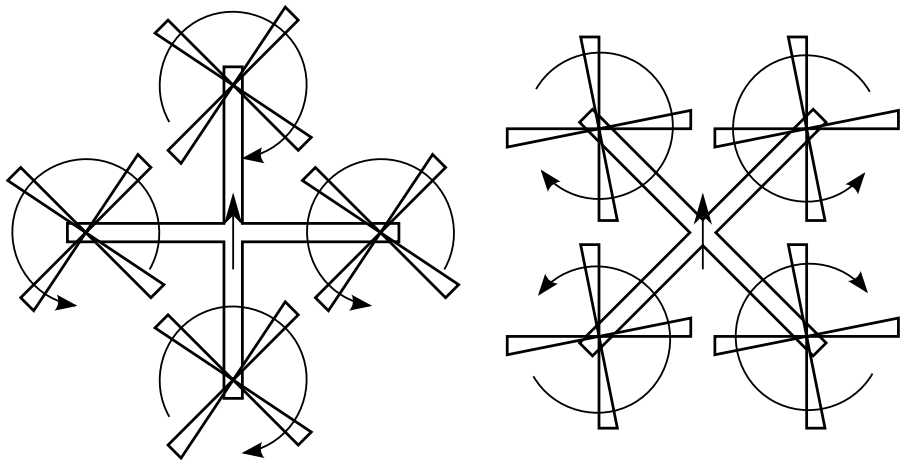


Figure 2.3: Schematics of Quadrotor MAV in (a) plus and (b) X configuration.

control methods for quadrotors has been published by Bouabdallah (2007).

Today, readily built quadrotors are commercially available in various sizes. For example, the Hubsan X4 (see Husban, 2013) has a size of only 6×6 cm, while the md4-1000 from Microdrones (see Microdrones GmbH, 2013) has a rotor-to-rotor distance of about 1 m. The smaller models are particularly well suited for indoor flight. This is not only due to their small dimensions, but also due to the fact that like other rotary wing aircrafts, quadrotors are able to take-off and land vertically and move at low speeds. Furthermore, quadrotors can achieve a high flight stability, which enables their usage within small confined spaces. Thus, it is a logical choice to select a quadrotor MAV as the research platform for this thesis.

2.1.3 Research Platform

A front- and bottom-view of the quadrotor MAV, which was used for the research conducted in this thesis, is shown in Figure 2.4. The design of this MAV is based on the PIXHAWK Cheetah that was developed at the ETH Zürich (see Meier *et al.*, 2011, 2012). The PIXHAWK Cheetah includes a custom-designed microprocessor board that serves as Inertial Measurement Unit (IMU) and low-level flight controller. Furthermore, it includes a custom-made carrier board for a COM-Express single board computer. The hardware designs of both circuit boards, as well as the low-level flight control software, have been made available as open source and can be obtained from the PIXHAWK website (see Meier *et al.*, 2013).

Apart of the two custom circuit boards, the quadrotor used in this thesis consists mainly of standard components that can be purchased from shops for radio controlled model aircrafts. One exception, however, is the quadrotor frame that was custom cut

Figure 2.4: The quadrotor MAV that has been used for the research presented in this thesis, as seen from front and bottom.

from a lightweight carbon sandwich material and resembles the frame of the original PIXHAWK Cheetah design. This frame has a rotor-to-rotor distance of 40 cm. The MAV is powered by four brushless electric engines with a maximum mechanical power of 110 W each. As rotor blades, the MAV uses propellers with a diameter of 10 inch, which is the recommended size for the used motor type.

The quadrotor has been equipped with the powerful COMe-cPC2 single board computer from Kontron (see Kontron AG, 2013). This computer follows the COM-Express Compact form factor, which means that it has a size of only 9.5×9.5 cm. This makes it ideal for the employment on a weight-constrained MAV. Despite these small dimensions, the on-board computer features an Intel Core 2 Duo CPU with 1.8 GHz, which provides sufficient computing resources for on-board image processing. In the used configuration, the computer is fitted with 2 GB main memory and a 64 GB solid state disk, which should facilitate the recording of our extensive sensor data.

On the sensory part, the quadrotor has been equipped with four USB Firefly MV cameras from Point Grey (see Point Grey Research, Inc., 2013). The cameras are arranged in two stereo configurations by using a custom-made camera holder, which was constructed of the same lightweight carbon sandwich material as the quadrotor frame. Two cameras are facing forward with a baseline distance of 11 cm, while the other camera pair faces downwards with a baseline distance of 5 cm. To ensure that the downward-facing cameras have a large field of view even during ground proximity flight, they have been fitted with ultra wide angle lenses with a focal length of only 3 mm. For the forward-facing cameras, lenses with a focal length of 4 mm are used.

Each camera has a gray-scale image sensor and is operated at a resolution of 640×480 pixels. The cameras have an additional GPIO port, through which they can send or receive trigger signals. The left forward-facing camera has been defined as master camera, and all other cameras are connected to its GPIO port. By letting the master camera generate trigger signals, we can ensure that all four cameras trigger simultaneously. This is a property that is particularly important for stereo vision systems.

The cameras are connected through USB to the on-board computer, which is dedicated to all image processing and high-level processing tasks. The computer runs Linux and the Robot Operating System (ROS), which was initially introduced by Quigley *et al.* (2009). ROS is a set of software libraries and tools that assist in the development of robotics applications. In particular, it comprises libraries for message passing of sensor and processing data, and tools for recording and visualizing this message communication. All software discussed in this thesis has been implemented by making extensive use of ROS, which simplified the development process and makes software components interchangeable.

In addition to the main on-board computer, the MAV features the already mentioned microprocessor board from the original PIXHAWK design. This microprocessor board is connected through a serial link with the on-board computer. In addition, it is connected through an I^2C bus with the motor controllers. The available on-board software includes a nested PID controller that consists of separate attitude and position controllers, which

is similar to the method proposed by Mellinger *et al.* (2012). With this software it is possible to control the MAV if its current pose is known. While the MAV's position can be provided from the on-board computer using the available serial link, the attitude is estimated using the inertial sensors that are available on the microprocessor board. For steering the MAV, the on-board computer can transmit a desired target position to the control software, which then attempts to approach it. It is thus possible to implement autonomous flight options for this MAV by letting the on-board computer generate a series of desired target positions.

2.2 Stereo Vision

Another main focus of this thesis is stereo vision. Hence, in this place we provide a brief introduction to the geometric principles of stereo vision and the inherent challenges. Algorithms for stereo matching are not considered in this place, but are covered in Section 3.2 on page 22.

2.2.1 Stereo Geometry

With stereo vision we refer to all cases where the same scene is observed by two cameras at different viewing positions. Hence, each camera observes a different projection of the scene, which allows us to perform inference on the scene's geometry. The obvious example for this mechanism is the human visual system. Our eyes are laterally displaced, which is why we observe a slightly different view of the current scene with each. This allows our brain to infer the depth of the scene in view, which is commonly referred to as stereopsis. Although it has for long been believed that we are only able to sense the scene depth for distances up to few meters, Palmisano *et al.* (2010) recently showed that stereo vision can support our depth perception abilities even for larger distances.

Using two cameras and methods from computer vision, it is possible to mimic the human ability of depth perception through stereo vision. An introduction to this field has *e.g.* been provided by Klette (2014). Depth perception is possible for arbitrary camera configurations, if the cameras share a sufficiently large common field of view. We assume that we have two idealized pinhole-type cameras C_1 and C_2 with projection centers O_1 and O_2, as depicted in Figure 2.5[1]. The distance between both projection centers is the baseline distance b. Both cameras observe the same point p, which is projected as p_1 in the image plane belonging to camera C_1. We are now interested in finding the point p_2, which is the projection of the same point p on the image plane of camera C_2. In literature, this task is known as the *stereo correspondence problem*, and its solution through matching p_1 to possible points in the image plane of C_2 is called *stereo matching*.

[1] For simplicity the image planes have been drawn in front of the projection center, rather than in the physically correct position behind the projection center. The projections observed at both positions are, however, identical.

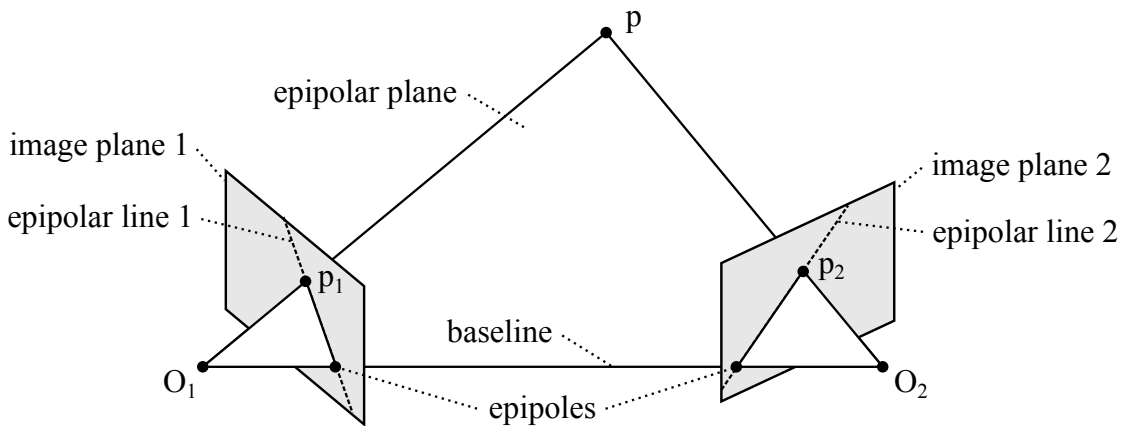


Figure 2.5: Example for epipolar geometry.

We consider a plane that passes through the projection centers O_1 and O_2, and point p_1. This *epipolar plane* intersects with both image planes at the *epipolar lines*. Irrespective of the position of p, the epipolar line for a given camera always passes through the projection of the respective other camera's projection center, the so-called *epipole*. Furthermore, the point p_2 on the image plane of camera C_2 is located on the epipolar line of C_2. This means that we can constrain the search for p_2 to this line, which renders stereo-matching a 1D search problem.

Depth inference can be further simplified if the cameras are aligned in the *standard epipolar geometry*, which is shown in Figure 2.6. In this setting, the optical axes of the left and right cameras C_L and C_R are parallel, and the image planes are coplanar. Furthermore, both cameras must have an identical focal length f. Please note that this geometry does not entirely match the human visual system. Although the optical axes of our eyes are aligned in parallel when observing distant objects, we tend to verge their optical axes when observing near objects.

In the standard stereo geometry, all epipolar lines are aligned horizontally. Furthermore, the epipolar lines corresponding to the same scene point have the same vertical offset v. Thus, in this case stereo matching only requires the comparison of image locations from equal image rows. Due to the high level of ambiguity, however, it is not possible to match two corresponding points by solely looking at individual pixels on the considered epipolar lines. Rather, we need to rely on further image information for this decision. Many algorithms have been proposed for this task, and we discuss some of these methods in the next chapter.

Once we have identified two matching image locations p_L and p_R from camera C_L and C_R, we can infer the location of the corresponding scene point $p = (x, y, z)$. In our case of idealized pinhole-type cameras, the projection of p to the image location $p_L = (u_L, v_L)$ is described through central projection. If we assume that the scene coordinate system matches the coordinate system of camera C_L, then the image location p_L can be determined as follows:

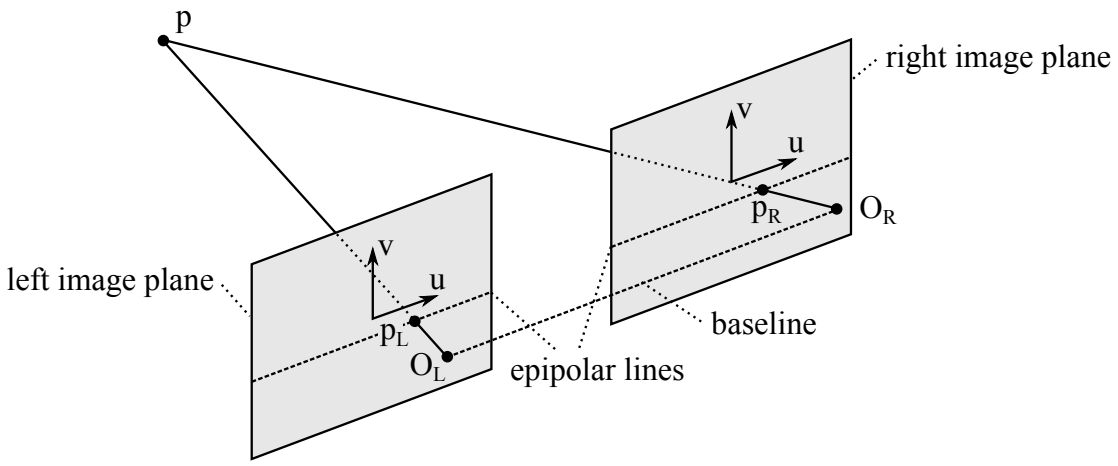


Figure 2.6: Example for standard epipolar geometry.

$$\begin{pmatrix} u_L \\ v_L \end{pmatrix} = \frac{f}{z} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.1}$$

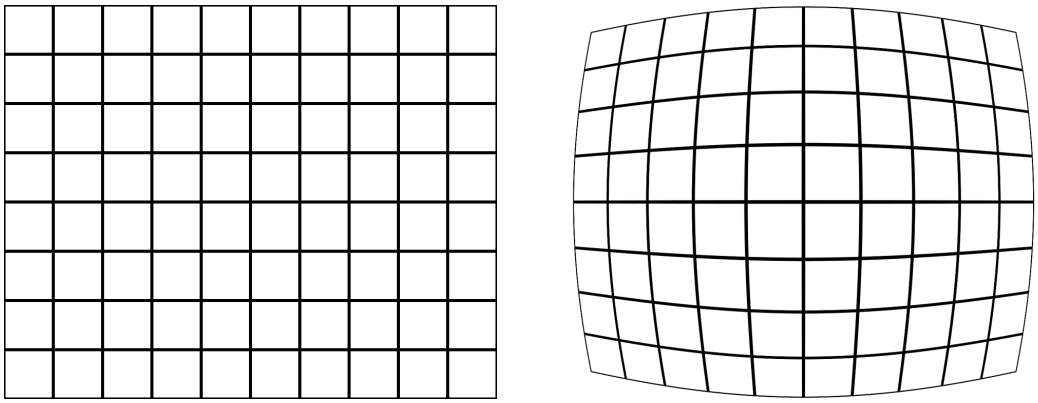
Similarly, we can determine the image location $p_R = (u_R, v_R)$ of the projection of p on the image plane of camera C_R:

$$\begin{pmatrix} u_R \\ v_R \end{pmatrix} = \frac{f}{z} \begin{pmatrix} x-b \\ y \end{pmatrix}. \tag{2.2}$$

Due to the standard epipolar geometry, the vertical coordinates v_L and v_R of the left and right image locations p_L and p_R are identical. Thus, the two image locations only differ by a horizontal displacement of magnitude $u_L - u_R$, which is commonly referred to as *disparity* $d \geq 0$. Small disparities correspond to a large depth z of the observed scene point, with $d = 0$ being at infinity. Similarly, large disparities correspond to a small depth of the given scene point. This allows us to further constrain the search space, by introducing a minimum depth limit z_{min}. If d_{max} is the disparity corresponding to this minimum depth, we can limit our search for corresponding points to $d \in [0, d_{max}]$.

If we have a valid estimate for the disparity d of image point p_L, we are able to reconstruct the location of the corresponding scene point p approximately. For this task, we consider Equations 2.1 and 2.2 as an equation system, and solve it for the scene coordinates of p. Thus, we are able to reconstruct the location of p if we know the baseline distance b and focal length f of the stereo setup:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{b}{d} \begin{pmatrix} u_L \\ v_L \\ f \end{pmatrix}. \tag{2.3}$$



(a) (b)

Figure 2.7: (a) Undistorted image and (b) image after radial distortion.

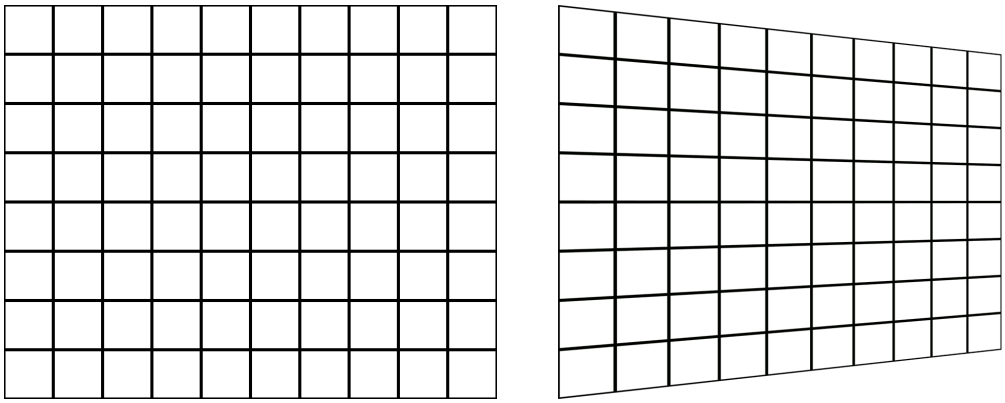
2.2.2 Image Rectification

The definition of standard stereo geometry uses idealized pinhole cameras. Real cameras, however, are not ideal and use lenses, which lead to various distortions. One of the major distortion types found in today's cameras is *radial distortion*. This is caused by the fact that an ideal lens should have a parabolic shape. Due to the difficulty of manufacturing a parabolic lens, however, most commercially available lenses have in fact a spherical shape.

The radial distortion that results from the deviation from the parabolic shape is shown for one example in Figure 2.7. Image regions close to the optical center receive only little distortion, while the image periphery is significantly warped. In the given example, straight lines are bent outwards from the optical center, which is known as *barrel distortion*. Radial distortion can, however, also cause straight lines to be bent inwards towards the optical center, which is known as *pincushion distortion*

Another common optical distortion is *tangential distortion*, which is caused by the image sensor not being aligned exactly perpendicular to the optical axis. An example for this distortion type is shown in Figure 2.8. Here, the image sensor is slightly rotated around its vertical axis, with the left edge of the sensor being closer to the scene than the right edge. Thus, the scene visible in the left image half appears larger than the scene visible in the right image half.

Both, radial and tangential distortion break the standard epipolar geometry. Should such distortions occur, then the epipolar lines are no longer exactly horizontal and collinear. Thus, the common approach to stereo vision includes a preliminary *image rectification* step, during which distortions are corrected. The resulting image after rectification should match the image received from an ideal pinhole camera. To be able to perform such a correction, we first require an accurate model of the image distortions. The distortion model that is most frequently used for this task today, is the one introduced by



(a) (b)

Figure 2.8: (a) Undistorted image and (b) image after tangential distortion.

Brown (1966). Using Brown's distortion model, we are able to calculate the undistorted image location $(\tilde{u}, \tilde{v})$ that corresponds to the image location (u, v) in the distorted image:

$$\begin{aligned}\tilde{u} =&(u-u_c)(1+K_1r^2+K_2r^4+\cdots)+(P_1(r^2+2(u-u_c)^2)+\\ &2P_2(u-u_c)(v-v_c))(1+P_3r^2+P_4r^4+\cdots),\end{aligned} \tag{2.4}$$

$$\begin{aligned}\tilde{v} =&(v-v_c)(1+K_1r^2+K_2r^4+\cdots)+(P_2(r^2+2(v-v_c)^2)+\\ &2P_2(u-u_c)(v-v_c))(1+P_3r^2+P_4r^4+\cdots).\end{aligned} \tag{2.5}$$

Here, (u_c, v_c) is the image location of the projection center, $K_1, K_2, \ldots, K_n$ are radial distortion coefficients and $P_1, P_2, \ldots, P_m$ are the coefficients for the tangential distortion. The quantity r was introduced for simplification and can be computed as follows:

$$r = \sqrt{(u-u_c)^2+(v-v_c)^2}. \tag{2.6}$$

The infinite sum in this distortion model is approximated by limiting the number of distortion coefficients. This is done by setting all higher elements in the infinite series of radial and tangential distortion coefficients to 0. Hence, the infinite sum in Equation 2.5 is reduced to a computable finite sum. Common numbers of coefficients are two tangential distortion coefficients, and between two and six radial distortion coefficients.

The challenge at hand is now to identify the radial and tangential distortion coefficients, which happens by camera calibration. This is commonly done by recording several images of one or more known geometric objects. A flat board with a visible checker pattern is frequently used for this task, but other calibration objects such as the circle and ring patterns used by Datta *et al.* (2009) are also possible. Once these images have been recorded, known features of the calibration objects are extracted. For a checker pattern, these are the corners of the individual checker tiles, while the circle and ring centers are

(a) (b)

Figure 2.9: Example for (a) unrectified and (b) rectified camera image.

used for the patterns from Datta *et al.*.

Because the geometries of the calibration objects are known, we also know the expected relative scene locations of the extracted features. Thus, we are able to use the feature locations for estimating the distortion coefficients. The method proposed by Zhang (2000) is widely used for this task in case of planar calibration patterns. This method employs a closed from solution of the calibration problem, which is improved through a non-linear refinement based on the maximum likelihood criterion.

Further efforts have to be made in case of a stereo camera pair. We want the epipolar lines of both cameras to have an exact horizontal alignment. Furthermore, corresponding epipolar lines from both cameras should have an identical vertical coordinate. Hence, we are required to determine a rotation matrix $\mathbf{R}$ for the relative rotation between both cameras and their relative translation vector $\mathbf{t}$. This allows us to correct the alignment of the two image sensors. We can determine both $\mathbf{R}$ and $\mathbf{t}$ by estimating the poses P_L and P_R of the calibration object as observed by the left and right camera. We are able to determine these poses due to the known geometry of the calibration object. Because there is a geometric relation between the object poses determined for each camera, we are able to infer $\mathbf{R}$ and $\mathbf{t}$.

Existing implementations of the discussed algorithms can be found in the OpenCV library (Itseez, 2013) or the Matlab camera calibration toolbox (Bouguet, 2013). An example for an unrectified camera image with strong radial distortion is shown in Figure 2.9a, and the corresponding rectified image is given in Figure 2.9b. Please note that due to the strong deformation of the input image, the rectified image now includes regions that have not been observed by the camera. No image information is available for these regions, which is why they have been shaded black. These regions need to be ignored when processing the rectified camera image.

2.2.3 Challenges

Stereo rectification leads to a precise alignment of the epipolar lines in the left and right camera image. Hence, the rectified imagery appears to originate from an ideal standard epipolar geometry. This means that during stereo matching, we are only required to compare image locations that have the same vertical coordinate v in both images. The task of matching corresponding points in the left and right image might thus appear simple at first, but it is actually very challenging. There exists a high ambiguity when matching corresponding image locations. Furthermore, matching is aggravated by several disruptive effects, to which we refer to as 'noise'.

Noise can be caused by a range of different sources. For example, like all signal measurements, the intensity of an individual pixel is subject to signal noise. Further noise is added by the fact that the camera performs a quantization of all intensity measurements. These effects are well known from signal processing. Noise in stereo vision can, however, also originate from geometric sources. The left and right camera might observe a slightly different perspective distortion for an object in view, due to their different viewing positions. Another example for geometric noise is occlusion. A background object occluded by a foreground object in one camera might be visible in the other camera.

Another source of noise are surface properties. It is common in stereo vision to assume that all visible surfaces are perfect Lambertian scatterers. This means that they reflect light equally in all possible directions. This assumption is, however, violated if we encounter glossy, reflective or semi-translucent objects. In this case, the same surface point on one such surface might have a very different intensity when observed by the left or right camera.

Finally, the cameras themselves contribute noise as well. The cameras might use slightly different exposure times and the image sensors might have slightly different sensitivities. Furthermore, there are many problems introduced by the camera lenses. We have already discussed lens distortion, which we attempt to correct through image rectification. However, a small residual distortion is likely to remain. Lenses also have a limited depth of field and they might not be focused exactly equally. Furthermore, lenses can be subject to lens flares when encountering bright light sources such as the sun.

Having so many sources of noise makes stereo matching challenging. When matching image locations from the left and right input image based on their pixel intensities, it is likely that the two best matching image locations do not correspond to the same scene point. If the object in view does not exhibit sufficient texture, finding a correct match might not even be possible in case of noise-free observations. It is thus not sufficient to evaluate individual pixels for their stereo correspondence. In order to perform reliable stereo matching, we require robust methods that can operate within the presence of noise and matching ambiguities. We present and evaluate one such method in the next chapter.

Chapter 3

Sparse Stereo Vision

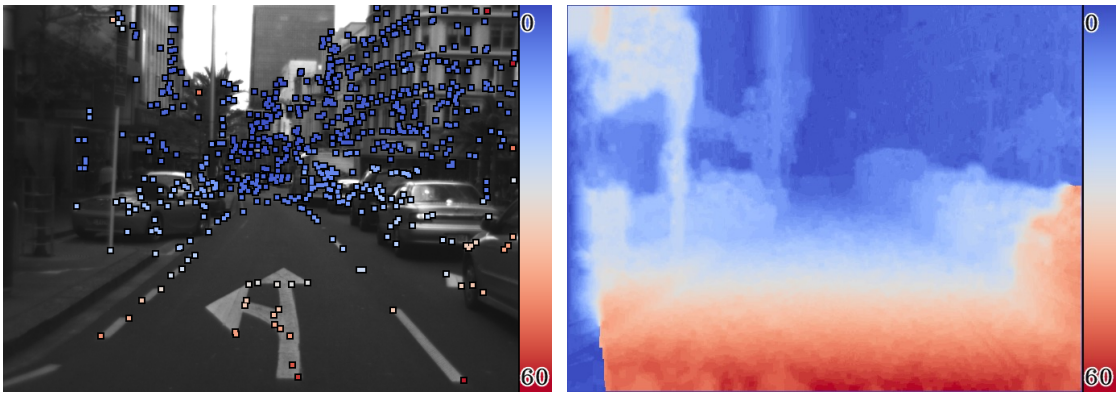
3.1 Introduction

As we have seen in the previous chapter, stereo matching is essentially a 1D search problem. After selecting a reference image (usually the left image), its pixels are matched to pixels from the equivalent epipolar line in the match image. The result is a labeling that links pixels from the reference image to their best matching counterparts in the match image. Due to the many sources of noise and ambiguities, it is hardly sufficient to look at individual pixels for this task. Rather, we need to consider a pixel neighborhood, or even better the entire input images, when choosing the label assignment for a single pixel from the reference image. This greatly increases the computational load, which is why stereo matching generally is computationally a very expensive process.

Our chosen MAV platform unfortunately has firm computational constraints. Furthermore, stereo matching is only one amongst several computationally expensive processes that are required to run on-board the MAV in real-time. Hence, stereo matching is only allowed to consume a fraction of the available on-board processing power. In addition, a quadrotor MAV is an inherently unstable system that requires fast responding controllers in order to maintain stable flight. This means that if we rely on stereo vision for controlling our MAV, then stereo matching has to run at a relatively high processing rate. Our aim is to achieve a rate of at least 30 frames per second with images of size 640×480 pixels that are delivered by our on-board cameras. All these requirements make it imperative that the chosen stereo matching method is computationally very efficient.

One way to greatly speed-up stereo matching is to not process all pixel locations of the input images. While the commonly used *dense* approaches find a disparity label for almost all pixels in the reference image, *sparse* methods only process a small set of salient image features. An example for the results received with a sparse compared to a dense stereo matching method can be found in Figures 3.1a and 3.1b.

The shown sparse example was received with the method that we present in this chapter, which only finds disparity labels for a set of selected corner features. The color that is displayed for these features corresponds to the magnitude of the found disparity, with blue hues representing small and red hues representing large disparity values. The method used for the dense example is the gradient-based belief propagation algorithm



(a) (b)

Figure 3.1: (a) Sparse stereo matching results received with the presented method and (b) dense results received from a belief propagation based algorithm. The color scale corresponds to the disparity in pixels.

that was employed by Schauwecker and Klette (2010) and Schauwecker *et al.* (2011). The results of this algorithm are dense *disparity maps* that assign a disparity label to all pixels in the left input image.

Although sparse methods provide much less information than common dense approaches, this information can be sufficient for a set of applications. In particular, many methods for camera pose tracking map a sparse set of salient image features. Hence, a sparse stereo matching method would integrate well into such systems. For our MAV, this means that a sparse stereo matching method can be used for enabling the MAV to track its current pose, which is an important prerequisite for autonomous flight. In this chapter, we hence focus on designing an accurate and efficient sparse stereo matching algorithm that can be used for this task.

The method described in this chapter was first published in 2012 at the IEEE International Conference on Intelligent Robots and Systems (IROS) (Schauwecker *et al.*, 2012a). This chapter includes important details that are beyond the scope of this previous publication. Most importantly, the processing of unrectified input images that is discussed in Section 3.4.3 on page 39 has not been covered in details before.

3.2 Related Work

Work relevant to the methods discussed in this chapter includes both work on stereo matching and feature detection algorithms. Hence, we have a separate look at the relevant literature in each field. For stereo matching, we also provide an overview of existing dense methods, as they represent the current state of the art in stereo vision research.

3.2.1 Feature Detection

In computer vision, a *feature detector* is an algorithm that selects a set of image points from a given input image. These points are chosen according to detector-specific saliency criteria. A good feature detector is expected to always select the same points when presented with images from the same scene. This should also be the case if the viewing position is changed, the camera is rotated or the illumination conditions are varied. How well a feature detector is able to redetect the same points is measured as *repeatability*, for which different definitions have been proposed (*e.g.* see Schmid *et al.*, 2000; Gauglitz *et al.*, 2011).

Feature detectors are often used in conjunction with *feature descriptors*. These methods aim at providing a robust identification of the detected image features, which facilitates their recognition in case that they are re-observed. In our case, we are mainly interested in feature detection and less in feature description. A discussion of many existing methods in both fields can be found in the extensive survey published by Tuytelaars and Mikolajczyk (2008). Furthermore, a thorough evaluation of several of these methods was published by Gauglitz *et al.* (2011).

Various existing feature detectors extract image corners. Corners serve well as image features as they can be easily identified and their position can generally be located with good accuracy. Furthermore, image corners can still be identified as such if the image is rotated, or the scale or scene illumination are changed. Hence, a reliable corner detector can provide features with high repeatability.

One less recent but still popular method for corner detection is the Harris detector (Harris and Stephens, 1988). An example for the performance of this method can be seen in Figure 3.2b. This method is based on the second momentum or auto-correlation matrix, which describes the gradient distribution in the local neighborhood of a given image point. The eigenvalues of this matrix correspond to the intensity change along two orthogonal axes. Hence, image corners are located at points where both eigenvalues of this matrix are large.

A computationally less expensive method for detecting image corners is the Smallest Univalue Segment Assimilating Nucleus (SUSAN) detector that was proposed by Smith and Brady (1997). Because this method does not rely on local image gradients, it is claimed to be less sensitive to image noise. To evaluate whether a given image point p is at a corner location, this method examines the pixels in a disc-shaped neighborhood with p at its center (the nucleus). The pixels in this neighborhood are then classified as pixels with 'similar' and 'significantly different' intensity values compared to p. Corners are located where the number of 'similar' pixels is minimal.

A more efficient method that is similar to the SUSAN detector is Features from Accelerated Segment Test (FAST), for which an example is shown in Figure 3.2c. Instead of evaluating all pixels on a circular disc, this method only considers the 16 pixels on a Bresenham circle of radius 3, which can be seen in Figure 3.3. The circle pixels are again compared to the central point p and classified as pixels with 'similar' and 'significantly

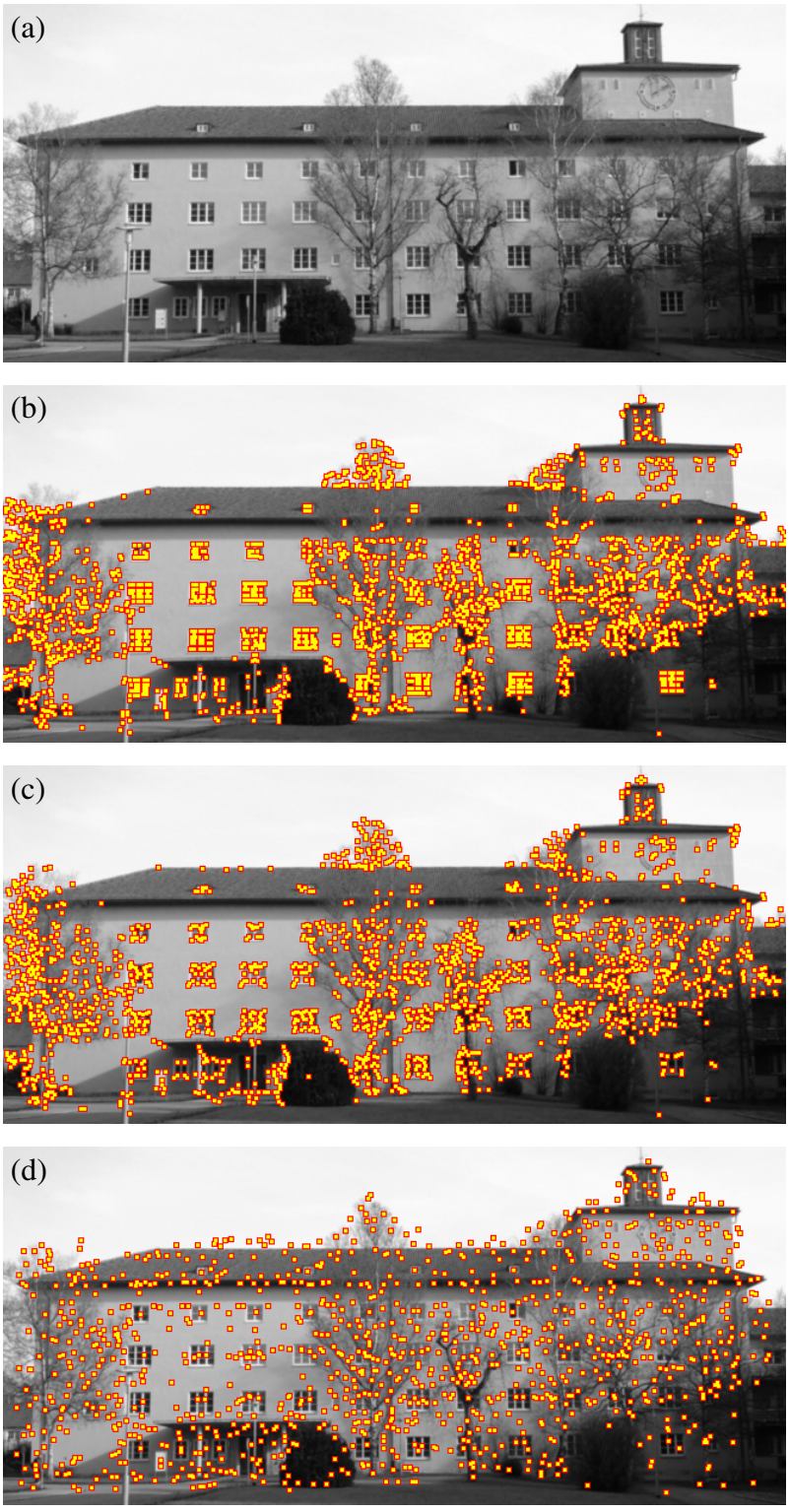


Figure 3.2: (a) Input image and features form (b) Harris detector, (c) FAST and (d) SURF.

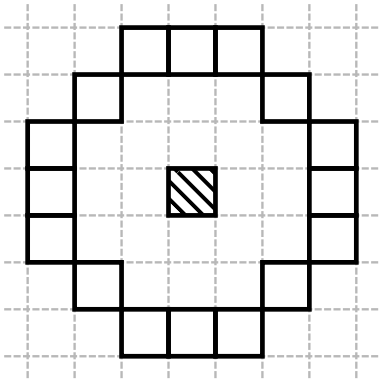


Figure 3.3: Pixels used by FAST for detecting image corners.

different' intensity values, by using a threshold t. If a contiguous arc of length $l \geq n$ is formed by pixels with 'significantly different' intensities, then p is classified as being a corner location. This method was first published under the name FAST by Rosten and Drummond (2005) for $n = 12$. In this case, a fast corner evaluation can easily be implemented by means of nested conditional branches, which each compare the intensities of two pixels. However, it was shown by Rosten and Drummond (2006) and Mair *et al.* (2010) that better results can be achieved if $n = 9$.

Rosten and Drummond thus extended their method to allow for arbitrary values of n and to reduce the total number of comparison operations (Rosten and Drummond, 2006). This algorithm was again named FAST and is the algorithm that is commonly referred to under this name today. The method uses a machine learning algorithm to find a decision tree that allows for the quick rejection of non-corners. The result of this machine learning step is used to generate code that consists of a large number of nested conditional branches. The authors reported that in an evaluation, their method compared each pixel to only 2.26 other pixels on average. An extension of this method that considers a larger pixel neighborhood was published under the name FAST-ER by Rosten *et al.* (2010). This method, however, requires more processing time than the original FAST algorithm, which has impeded its widespread adoption.

FAST is usually combined with non-maxima suppression, in order to receive only one feature per image corner. Because the segment test does not produce a corner score, one has to be computed separately for each identified corner pixels. The method proposed by Rosten and Drummond (2006) is based on brightness differences between the detected arc and the central pixel. The current version of the available source code for FAST, however, applies a binary search to find the highest value of t, for which a pixel is still detected as a corner. This method matches the approach applied by Rosten *et al.* (2010).

An improved adaptation of FAST, coined Adaptive and Generic Accelerated Segment Test (AGAST), was published by Mair *et al.* (2010). The authors simplified the decision tree to not always evaluate a pixel for higher *and* lower intensity, but to only evaluate "one single question per time". Furthermore, the authors avoided the training step that

is required for FAST, by generating two specialized decision trees for homogeneous and structured regions. The algorithm then switches between the two decision trees depending on the previously observed pixel configuration. In a performance evaluation, the authors observed a speed-up of 13% for $n = 9$, when compared to the original FAST algorithm. However, more recent implementations of the FAST detector, such as the ones available in recent versions of OpenCV (see Itseez, 2013) and libCVD (see Rosten, 2013), are no longer based on decision trees. Rather, these methods employ SIMD instructions that are available on current CPUs, to perform a parallel comparison and evaluation of several pixels. Hence, the speed-up achieved by AGAST seems insignificant when compared to current implementations of FAST.

FAST is also at the core of the more recently published Oriented FAST and Rotated BRIEF (ORB) feature detector and descriptor (Rublee *et al.*, 2011). This method applies the FAST detector with a lower-than-necessary threshold. For the found features, the response of the Harris detector is then computed, and only the features with the highest Harris response are kept. A rotation component is then determined for the detected features, which is used to obtain a rotation invariant version of the Binary Robust Independent Elementary Features (BRIEF) descriptor (Calonder *et al.*, 2010). Because of its low computation costs and high quality features, this method has already been adopted in numerous research projects.

An alternative to corner detectors are the so-called *blob detectors*. These methods attempt to find image regions that in some property differ significantly from their surrounding area. One of the most influential methods in this category is the Scale Invariant Feature Transform (SIFT) by Lowe (1999). For this method, two Gaussian convolutions with different values for σ are computed for the input image. The difference between both convolutions, called Difference of Gaussians (DoG), is then used for detecting the feature locations. Features are located at points where the DoG reaches a local maximum or minimum. Lowe further proposed a robust rotation-invariant feature descriptor, which is based on an examination of gradient orientations in a local neighborhood.

A more time-efficient blob detector that was inspired by SIFT, is Speeded-Up Robust Features (SURF) by Bay *et al.* (2006), for which an example is shown in Figure 3.2d. Instead of using a DoG for detecting feature locations, Bay *et al.* rely on the determinant of the Hessian matrix, which is known from the Hessian-Laplace detector (Mikolajczyk and Schmid, 2001). The Hessian matrix is based on convolutions of the input image with second order Gaussian derivatives. Bay *et al.* perform a rough but very fast approximation of these convolutions by using box filters with integral images, as known from Crow (1984) and Viola and Jones (2002). SURF further comprises a robust feature descriptor that is based on the response of Haar wavelets, which are again efficiently computed using integral images.

Both SIFT and SURF exhibit a very high repeatability, as has *e.g.* been shown by Gauglitz *et al.* (2011). However, what Gauglitz *et al.* also have shown is that both methods require significant computation time. Given our high performance requirements,

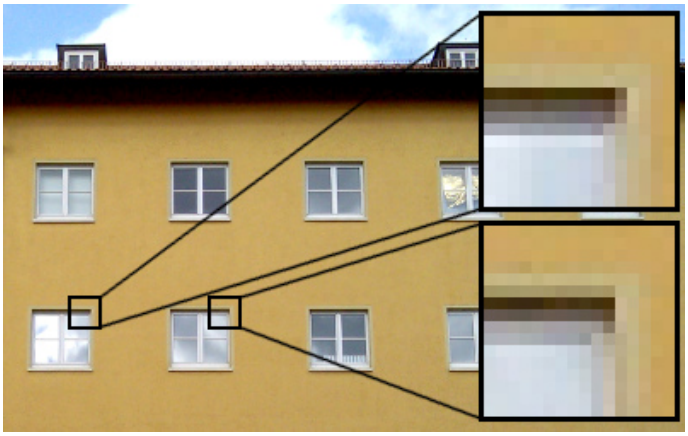


Figure 3.4: Example for ambiguous image regions. Both magnified sections have an almost identical appearance.

these methods are unfortunately not suitable for employment on our MAV. Hence, we instead focus on corner detectors, of which FAST is one of the most efficient methods.

3.2.2 Dense Stereo Methods

Dense methods receive the most interest in today's stereo vision research. Scharstein and Szeliski (2002) categorized dense stereo methods into *local* and *global* methods. For matching a left image location p_L against a right image location p_R, a local method examines the local pixel neighborhoods of p_L and p_R. Hence, only pixels within this local neighborhood can have an impact on the obtained *matching cost*. The problem with such methods is that the set of pixels within the local neighborhood are not always sufficient for computing a reliable matching cost. An example for this circumstance is shown in Figure 3.4, where two magnified subsections of an image have an almost identical appearance.

An alternative to local methods that overcomes this limitation are global methods. When using a global method, all pixels in both images can have an influence on the computed matching costs. Consequently, global methods generally require substantially more processing time than their local counterparts. In effect, these methods also tend to provide significantly more accurate and more robust results.

Both local and global stereo methods generally assume a local smoothness of the scene depth. Scenes observed in the real world usually provide a relatively smooth depth variation with few depth discontinuities. Hence, stereo matching results with strong depth variations generally result from matching errors. By using a *smoothness constraint*, we can penalize such solutions and increase the robustness of a stereo method. For local methods, this constraint is often implicit, by assuming that all pixels within the current local neighborhood are at the same depth. Global methods on the other hand, generally

employ an explicit *smoothness term* that increases the matching cost for correspondences that cause high depth variations.

One of the simplest and fastest local algorithmic approaches to stereo matching is Block Matching (BM). Here, rectangular windows are matched that are centered at the evaluated image locations in the left and right image. Using this method, an image location in the left image is matched against image locations in the right image that are on the same epipolar line and within a preset disparity range. The pair with the lowest matching cost is then chosen as the most likely stereo correspondence.

One example for a very efficient stereo matching implementation that is based on BM has been provided by Humenberger *et al.* (2010). The authors proposed both an efficient CPU- and GPU-based implementation of their method. The CPU implementation achieved a processing rate of 63 frames per second, with test data of resolution 320×240 and only 15 disparity levels. This processing rate is significantly above the rates reported for most of the software stereo methods that were considered by Humenberger *et al.* in their performance evaluation.

Current global stereo methods provide results that are far more accurate than those that can be achieved with BM. These methods usually work by minimizing an explicit cost function that provides a cost for all possible solutions of the stereo correspondence problem (*i.e.* all possible disparity maps). Such a cost function generally consists of a *data term* and the already mentioned smoothness term. While the smoothness term assigns low costs to solutions with smooth depth variations and high costs to solutions with an abruptly varying depth, the data term determines a similarity cost for the possible stereo correspondences. Small costs are assigned to solutions where the found correspondences have a very similar appearance, and high costs are assigned to solutions with very dissimilar correspondences. A global stereo matching algorithm attempts to find a disparity map that minimizes this cost function.

Different methods exist for performing this minimization process. One example are the two Graph Cuts (GC) based stereo matching algorithms proposed by Boykov *et al.* (2001). The algorithms start with an arbitrary disparity labeling, and then change a set of labels in each of their iterations. This happens by either changing a group of pixels that were previously labeled with α to a new label β (called α-β-swap) or by changing a group of pixels with an arbitrary previous label to a new label α (called α-expansion). In each iteration, the algorithms try to find the best α-β-swap or α-expansion, which can be determined using graph cut techniques known from combinatorial optimization. It was shown by Szeliski *et al.* (2007) that α-expansion based algorithms generally perform better than their α-β-swap based counterparts. Unfortunately, however, GC-based algorithms are exceedingly slow on today's hardware, which limits their practical applications.

For the Belief Propagation (BP) based stereo algorithm (Sun *et al.*, 2003), the minimization problem is modeled as a Markov Random Field (MRF). Each node in this MRF corresponds to one pixel in the sought after disparity map. Edges exist between the nodes for adjacent pixels, which allows for the modeling of a smoothness constraint. By itera-

tively passing messages along these edges, the algorithm is able to effectively minimize its cost function. In the evaluation performed by Szeliski *et al.* (2007), this method was able to find solutions that had a lower cost than the available ground truth.

Unfortunately, many iterations are required until the algorithm converges at an accurate solution, and the messages passed between the nodes require much memory. Various methods have thus been published for improving the performance of BP. The hierarchical algorithm published by Felzenszwalb and Huttenlocher (2006) reduces the number of required iterations by introducing a coarse to fine processing scheme that gradually increases the MRF resolution. This approach was extended further by Yang *et al.* (2006), who reduced the number of passed messages by only updating nodes that have not yet converged. The authors further parallelized their algorithm to leverage the performance of a GPU. Another extension of the hierarchical BP algorithm was published by Yang *et al.* (2010), which gradually increases the number of disparity labels. In this case, the authors reported a processing time of 1.5 s for images with resolution of 800×600 pixels and 300 disparity levels.

One algorithm that has become very popular in recent years, and which cannot easily be classified as either local or global, is Semi-Global Matching (SGM) (Hirschmüller, 2005). This method can be seen as an extension of the Dynamic Programming (DP) stereo algorithm that was initially published by Baker and Binford (1981). DP individually optimizes the disparity labeling for each epipolar line, using the dynamic programming paradigm. Because of this independent optimization, errors tend to propagate along the epipolar lines, which results in disparity maps with obvious streaks. SGM solves this problem by performing the optimization using eight scan lines that propagate in different directions. Hence, for each pixel and possible disparity label, the costs from eight different paths are aggregated, which enables this method to find a robust disparity assignment.

One particularly fast implementation of SGM was published by Gehrig and Rabe (2010). This implementation achieved processing rates of 14 frames per second, on test data with an image resolution of 640×320 pixels. Faster processing rates can be achieved by implementing SGM on a GPU or FPGA, as has been demonstrated by Haller and Nedevschi (2010) and Gehrig *et al.* (2009). However, powerful GPUs generally consume much energy, which makes them unsuitable for deployment on an energy-constrained MAV. FPGAs on the other hand, generally require custom designed hardware, which is not easily available. Hence, in our case we limit ourselves to software stereo algorithms.

3.2.3 Sparse Stereo Methods

As we have seen, there exist efficient stereo matching implementations for *e.g.* BM or SGM. For our needs, however, these methods are still too slow, which is why we are interested in sparse stereo methods. In general, the first step of a sparse stereo matching system is the extraction of salient image features. Once the features have been ex-

tracted, similar features from the left and right input image can be matched. Throughout the 1980s, such sparse algorithms have been an active field of research. With the improved performance of dense methods, however, interest in sparse methods decreased and nowadays they only receive very little attention. Much of this early work on sparse stereo matching has been summarized by Dhond and Aggarwal (1989). Many of these methods apply an edge detector to extract edge pixels, which then serve as features for stereo matching.

One method from this era is the edge based stereo method that was published by Medioni and Nevatia (1985). As features serve edge *segments*, which are groups of collinear connected edge points that are extracted with the method proposed by Nevatia and Babu (1980). For the correlation of an edge segment in one image, a parallelogram-shaped local window in the other image is defined in which the candidate segments have to be located. Preference is given to segments that have a similar disparity to their neighbors. This is enforced by a cost function that is applied in multiple matching iterations.

Another example for an early sparse stereo method is the approach published by Eric and Grimson (1985), which was inspired by the human visual system. This method employs a coarse to fine process, by first computing different convolutions of the input image with the Laplacian of Gaussian (LoG) and varying values of σ. The points where these convolutions exhibit a zero-crossing serve as features, and form a set of contour lines. Starting at the convolution with the highest σ, these contours are matched using the direction of the zero-crossing as matching criterion. Matching then continues at the finer levels, while ensuring a consistency with the results from the previous iteration.

One method from this early era that is still popular today is the algorithm proposed by Lucas and Kanade (1981). While today, the Lucas-Kanade algorithm is usually used for optical flow estimation, an application to stereo vision was proposed in its initial publication. The method assumes that the apparent image movement, which in the case of stereo vision is caused by the camera displacement, is constant in a local neighborhood. Furthermore, it is expected that this movement is small or that a sufficiently accurate initial estimate is already known. Lucas and Kanade assume that the so-called *optical flow equation* holds for all pixels within this neighborhood. A least squared error method is then used to find a robust estimate for the solution of the resulting equation system. In the proposed stereo vision application, Lucas and Kanade presented an interactive system that requires a user to select points in an input image. After providing an initial depth estimate, the Lucas-Kanade algorithm is used for receiving an accurate depth measurement.

In more recent times, Vincent and Laganiere (2001) have published an evaluation of sparse stereo matching strategies. In their work, corner features are detected in the left and right input images, and matched using different correlation methods. Stereo matching is not restricted to epipolar lines, as the camera arrangement is unknown. The authors evaluated the impact of different matching constraints on the achieved results. Correspondences that do not satisfy those constraints are rejected as mismatches. The evaluated constraints include methods that ensure uniqueness and symmetry of the found

matches, compare the appearance of the corner features, or ensure a consistency of the found labeling. Many of these constraints have been found to reduce the number of erroneous matches.

One of the very few recently published sparse stereo methods is the BP-based algorithm that was introduced by Sarkis and Diepold (2008). This method, however, does not resemble a classical sparse algorithm. The points used as features are non-uniformly distributed samples that are drawn from the left input image. The algorithm then matches these features to the full disparity range in the right image. Despite a performance improvement when compared to classical BP, this algorithm still exhibits high processing costs. In the evaluation performed by the authors, their implementation required up to 2.1 s for processing an input image with a resolution of 450×375 pixels.

Another recent sparse stereo algorithm was proposed by Witt and Weltin (2012). This method works by extracting edge pixels in both input images. Pixels on those edge segments are then matched with a winner-takes-all strategy. In a next step, the results for each edge segment are refined independently by using a method based on dynamic programming. The authors reported processing times of 60–85 ms for images of size 450×375 pixels or smaller. While this method seems very efficient, we are interested in even faster methods for deployment on our MAV.

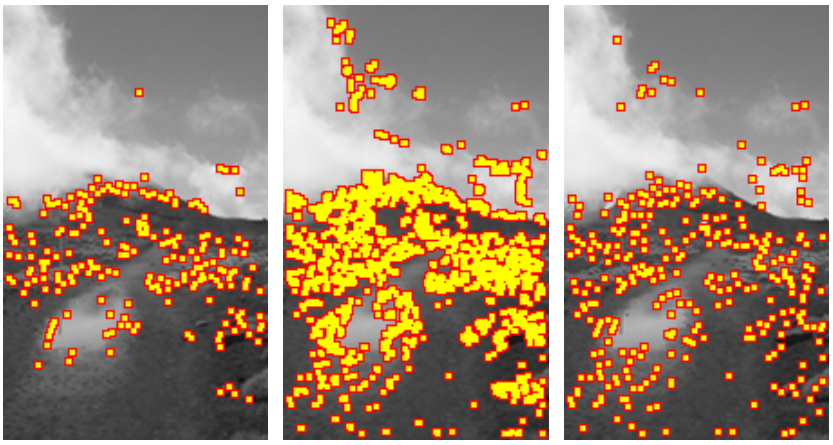
Much progress has been achieved in the computer vision community since the interest in sparse stereo matching declined. In particular, new feature detectors have since been published, which can be used for the construction of new sparse stereo methods. Given our need for highly efficient stereo matching, it seems logical to choose a sparse stereo method. The lack of current sparse stereo algorithms that fulfill our performance requirements, while also delivering accurate matching results, is our motivation to design a novel sparse stereo matching system.

3.3 Feature Detection

Before focusing on stereo matching, it is necessary that we select an adequate feature detection algorithm. As our aim is to design a very high-performance stereo matching system, we chose the previously discussed FAST corner detector. This method was the fastest feature detector in the evaluation performed by Gauglitz *et al.* (2011), and hence appears to be the natural choice for our purposes. However, the results provided by this algorithm are not ideal for a stereo vision system. Hence, we extend this method to deliver features that are more suitable for our needs.

3.3.1 Adaptive Threshold

FAST tends to detect many features in areas with high local image contrast, while detecting only a few features in image areas with low local contrast. This can lead to situations



(a) (b) (c)

Figure 3.5: Examples for feature detection results with (a) FAST, (b) first detection stage of proposed method and (c) full proposed method.

where many features are clustered in a relatively small area of the input image. An example for this behavior can be seen in Figure 3.5a. Here, only a few of the detected features are located in the sky or in the lower quarter of the input image. Such a behavior is undesirable for most computer vision based applications. If the detected features are used for obstacle detection, then this can cause obstacles to be missed if they do not provide sufficient contrast. For visual localization, such a clustered feature distribution can degrade the received localization results due to higher numeric errors, when compared to a more even feature distribution.

To make this effect less severe, we propose to extend the FAST algorithm with an adaptive threshold that depends on the local image contrast. This way, a lower detection threshold can be used in image regions with low local contrast, while a higher threshold can be used for areas with high local contrast. Hence, this approach is expected to counteract the clustered feature distribution.

The main advantage of the FAST detector over other methods is its high detection speed. Consequently, an extension of this method has to ensure that the performance of the original algorithm is not drastically changed. This task is not trivial, since FAST only requires few pixel comparisons to decide whether or not a given image location should be classified as a feature. Applying a local contrast filter to the entire input image before running the FAST algorithm can easily change the feature detection performance by an order of magnitude or more.

To solve this problem, we employ a two-stage process. First, an unmodified FAST detector is run without non-maxima suppression and a low constant threshold t_c. This leads to the detection of a high number of features, as shown for one example in Figure 3.5b with $t_c = 10$. For each detected feature i, we then calculate an adaptive threshold t_i and re-

Algorithm 3.1: Two-stage feature detection with adaptive threshold.

```
/* Detect features with low constant threshold t_c.                    */
F_A := detectFeatures(t_c);
F_B := ∅;
foreach (u, v) ∈ F_A do
    /* Determine adaptive threshold.                                   */
    t_i := getLocalThreshold(u, v);
    /* Test if still a feature with adaptive threshold.                */
    if testFeature(u, v, t_i) then
        /* Add to feature set F_B                                      */
        F_B.append((u, v));
    end
end
/* Perform non-maxima suppression (optional).                          */
suppressNonMaxima(F_B);
```

run feature detection. Only if an image point passes both detection steps, it is considered to be a valid feature. If necessary, non-maxima suppression can then be applied to the features remaining after the second detection step. This approach is formally described in Algorithm 3.1.

As previously mentioned, the adaptive threshold t_i should depend on the local image contrast. Hence, we calculate a contrast measure for each feature location, by using a local pixel neighborhood. One of the most common contrast measures is the Root Mean Square (RMS) contrast, which was first put forward by Peli (1990). This measure is defined as the standard deviation of the pixel intensities, which in our case can be expressed as:

$$\sigma_i = \sqrt{\frac{1}{|N_i|} \sum_{p \in N_i} (I_p - \bar{I})^2}, \tag{3.1}$$

where p is a pixel from the local neighborhood N_i of feature i, I_p is the intensity at p, and $\bar{I}$ is the average intensity of all pixels in N_i.

Given our high performance requirements, we would like to avoid the computation of the square root that is necessary for this contrast measure. We hence perform a simplification of the original formula, by replacing the sum of squared differences with a sum of absolute differences. In this case, the computation of the square root is no longer necessary, which reduces the formula to:

$$\tau_i = \frac{1}{|N_i|} \sum_{p \in N_i} |I_p - \bar{I}|. \tag{3.2}$$

As local neighborhood we choose the same 16 pixels on the circle of radius 3 that

are used by FAST for feature detection. Since we compute our contrast measure for influencing the feature detection results, using this pixel neighborhood seems to be a consequential choice. Finally, we define our adaptive threshold t_i for feature i to be the product of our local contrast measure τ_i and a parameterizable adaptivity factor $a > 0$:

$$t_i = a \cdot \tau_i. \tag{3.3}$$

3.3.2 Averaged Center

For the original FAST detector, the pixel with the highest impact on the detection result is the central pixel that is compared to all pixels on the evaluated circle. Noise in this pixel's intensity can easily impede the detection of obvious features, or cause the detection of false or insignificant features. To reduce the effect of image noise on our detector, we perform a noise reduction for the intensity of the central pixel.

One simple way to reduce image noise is to apply a blurring filter. Such a blurring filter can be implemented as a convolution with a filter kernel (*e.g.* a Gaussian kernel). In its simplest form, a box filter can be used that averages the pixels within a rectangular window. In case of the box filter, this operator can be performed in constant time if we rely on the already mentioned integral images as used by Crow (1984) and Viola and Jones (2002). The computation of an integral image, however, has linear complexity.

Given the high computational efficiency of the original FAST detector, applying a blurring filter to the entire input image can cause a significant increase of the overall processing time. This is even the case if we use simple box filters. Hence, instead of applying a blurring operation to the entire input image, we only filter the pixel locations where a feature has been detected in the first detection stage. For our blurring filter, we select the five pixels in the center of the circle evaluated by FAST, which are highlighted in Figure 3.6. We then use the average intensity of those central pixels for feature detection, which replaces the intensity value of the single central pixel that is otherwise used by FAST.

We combine the use of this *averaged center* with the previously discussed adaptive thresholding. Throughout this thesis, we refer to the resulting algorithm as *extended FAST* or *exFAST*. An example for the performance of this method can be found in Figure 3.5c. As we can see, the detected features are more evenly distributed over the input image when compared to ordinary FAST. This matches the behavior that we intended to achieve with the proposed modifications and addresses the observed shortcomings of the FAST detector. Hence, this new feature detector forms the basis for the proposed sparse stereo matching system.

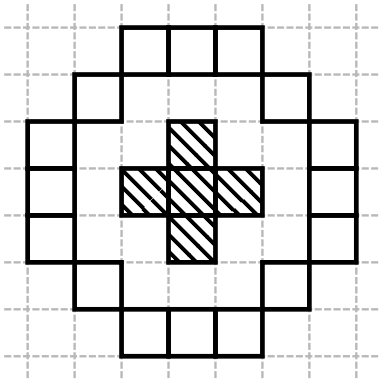


Figure 3.6: Pixels used for feature detection. Hatched pixels in the middle are averaged and compared to the circumcircle.

3.4 Stereo Matching

With the described feature detection method we detect features in the left and right camera image. Unlike for the left image, however, we omit the non-maxima suppression for features detected in the right image. This increases the number of possible matching candidates in the right image, to which features from the left image can be matched. Features in both images are considered to be possible matching candidates if they lie approximately on the same epipolar line (a deviation of 1 pixel is allowed), and if they are within a predefined disparity range. Similar to the previously introduced BM algorithm, features are matched by correlating two rectangular *matching windows* that are centered at the given feature locations. In the implementation targeted for our MAV platform, the matching windows have a size of 5×5 pixels. For correlating two matching windows, we require an appropriate correlation method.

3.4.1 Correlation

Different methods have been proposed in literature for pixel correlation, like the Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), Zero-Mean Sum of Absolute / Squared Differences (ZMSAD / ZMSSD), Normalized Cross Correlation (NCC), and so forth. We choose a method that is based on the census transformation, which is a non-parametric image transformation that was initially proposed by Zabih and Woodfill (1994). For each pixel location in an input image, the census transformation considers a rectangular *census window* (not to be confused with the matching window). The intensity I_i of the central pixel is then compared to the intensity I_j of each remaining pixel, by using the following comparison function:

$$\xi(I_i, I_j) = \begin{cases} 0 & \text{if } I_i \leq I_j \\ 1 & \text{if } I_i > I_j \end{cases} . \tag{3.4}$$

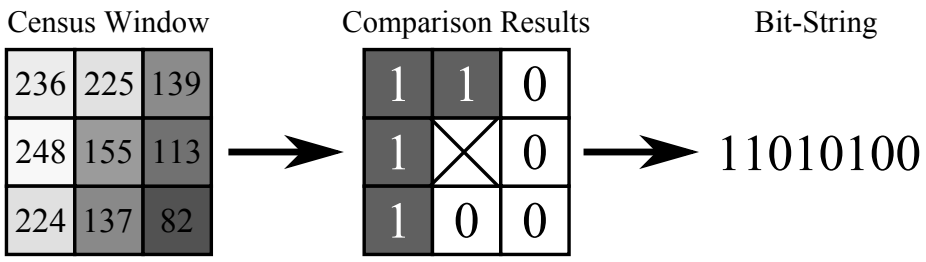


Figure 3.7: Illustration of the census transformation for a window of size 3×3.

The binary comparison results of $\xi(I_i, I_j)$ are then concatenated to form a bit-string. This process is illustrated in Figure 3.7 for a census window of size 3×3 pixels. After processing an input image with the census transformation, we hence receive a new 'image' in which every pixel is represented by one binary bit-string. For our stereo matching system, we again use a window size of 5×5 pixels for the census window. Hence, we require 24 bits to store the comparison results for each pixel location of the original image. To receive a more efficient memory alignment, each bit-string is kept in a 32-bit variable.

The bit strings b_L and b_R for two considered left and right image locations can be correlated by counting the number of unequal bits, which is known as the bitwise Hamming distance $\Delta_h(b_L, b_R)$. For our stereo matching system, we are required to correlate all pixel locations in the left and right matching window. Hence, for each bit-string in the left matching window, we compute the Hamming distance against the bit-string with corresponding coordinates in the right matching window. The aggregated Hamming distance is then our resulting matching cost c:

$$c = \sum_{(u,v) \in N_c} \Delta_h(L_{uv}, R_{uv}), \tag{3.5}$$

where L and R are the left and right bit-string 'images' and N_c is the set of image locations within the census window.

Compared to simpler methods such as SAD or SSD, a census transformation based correlation method can provide a significantly higher matching robustness, as has *e.g.* been shown by Hirschmüller and Scharstein (2008) or Hermann *et al.* (2011). This, however, comes at the price of higher computational requirements. To reduce the performance impact of the census transformation, an optimized version has been implemented that makes use of the SSE instruction set that is provided by current x86 CPUs. This allows for an efficient parallelization of the algorithm and enables us to achieve significant speed-ups when compared to a naïve implementation.

The required bitwise Hamming distance can be implemented by applying the `XOR`-operator to the two evaluated bit-strings. The *population count* of the result, which is the number of set bits, is then equal to the bitwise Hamming distance. For performing this operation efficiently on our MAV platform, we precompute the population count for

Algorithm 3.2: Hamming distance calculation for a and b using lookup tables.

```
/* Perform XOR operation to get non-equal bits.                    */
bits:= a XOR b;
/* Divide 32-bit string into two 16-bit strings.                   */
bits_1 := bitwiseAnd(bits, FFFF_hex);
bits_2 := bitShiftRight(bits, 16);
/* Look-up population counts.                                      */
pop_1 := populationCountTable[bits_1];
pop_2 := populationCountTable[bits_2];
/* Final result is the sum of both population counts.              */
hammingDistance := pop_1 + pop_2;
```

all possible 16-bit permutations, and store the result in a 64 KB lookup table. We then divide our 32-bit strings into two 16-bit strings, and determine the population count for each separately. This process is shown in Algorithm 3.2 for the two bit-strings a and b.

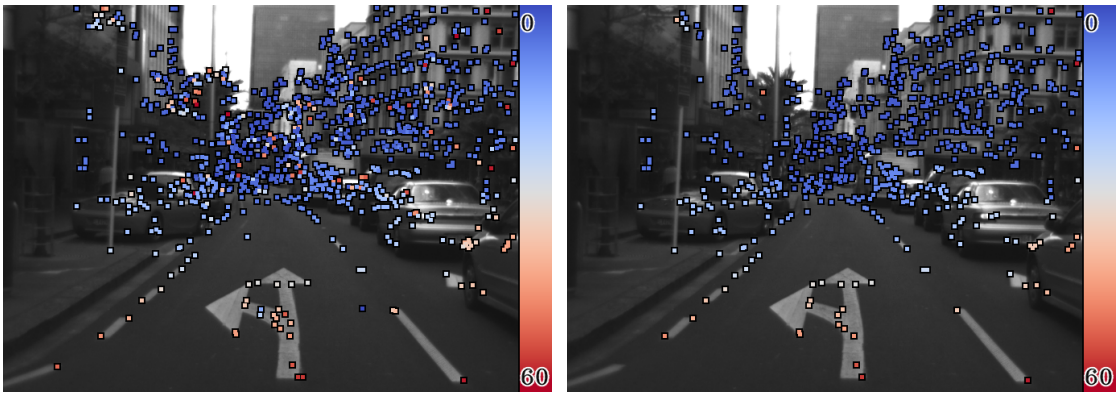
For other platforms, we use an alternative implementation that is based on the `POPCNT` CPU instruction, which was introduced by Intel with the SSE4 instruction set. This instruction provides an efficient way for computing the population count[1]. In particular, we use the 64-bit version of this instruction, which allows us to simultaneously process two census bit-strings. Unfortunately, this forces us to use a matching window with an even width. Hence, for other platforms than our MAV, we use a matching window of size 6×5 pixels.

3.4.2 Dense Consistency and Uniqueness Check

We retain the feature pair that received the lowest matching cost during the previous correlation process. As with most local stereo matching methods, these results contain a significant portion of false matches, which can be seen in Figure 3.8a. In this figure, several features that correspond to the distant background have been assigned an erroneous high disparity. For dense algorithms, the matches are often filtered using a left / right consistency check, as first introduced by Chang *et al.* (1991). This process works by repeating stereo matching in the opposite matching direction, and only retaining matches for which the results are consistent. We, however, perform sparse stereo matching, where a feature from the left image is matched to only few candidates in the right image. Those few matching operations are not sufficient to make the consistency check work effectively.

We thus apply a dense consistency check, despite the fact that our stereo matching method is inherently sparse. Hence, after the sparse matches have been established, we perform a dense matching step from the matched features in the right image to the valid

[1] The `POPCNT` instruction is also available on the Core2-Duo CPU of our MAV platform, but it is slower than the proposed lookup-table based method.



(a) (b)

Figure 3.8: Stereo matching results (a) without and (b) with consistency and uniqueness check. The color scale corresponds to the disparity in pixels.

disparity range in the left image. This means that we evaluate all pixel positions in the left image that are on the epipolar line and within the valid disparity range. Thus, our method differs from an ordinary sparse algorithm in that we examine the entire disparity range when deciding for a valid match.

Furthermore, we discard features with high matching uncertainty by imposing a uniqueness constraint. For a stereo match to be considered unique, the minimum matching cost c_{min} must be smaller than the cost for the next best match times a uniqueness factor $q \in (0,1]$. This relation can be expressed in the following formula, where C is the set of matching costs for all feature pairs and $c^* = c_{min}$ is the cost for the best match:

$$c^* < q \cdot \min\{C \setminus \{c_{\min}\}\}. \tag{3.6}$$

The uniqueness constraint can be combined with our dense consistency check: Rather than verifying during the consistency check that there is no other match with a cost $c < c_{min}$, we instead require c to be not smaller than $q \cdot c_{min}$. This means that we can ensure dense uniqueness at hardly any additional computation cost. However, in this case the uniqueness is ensured in the right-to-left matching direction, and not the left-to-right direction that we use for establishing the initial stereo matches. This however, is only a minor nuisance that should not reduce the quality of our results. How effective the resulting method is in removing erroneous matches can be seen in Figure 3.8b.

To speed-up this combined consistency / uniqueness check, we apply one further modification: Instead of evaluating each pixel location on the examined section of the epipolar line, we instead traverse the epipolar line with a step width $w \geq 1$. This means that we do not evaluate all pixels, but only consider a reduced subset. For example, if we set $w = 2$, then the total number of matching operations reduces to almost half of the original count. Later we show that increasing w only slowly reduces the matching ac-

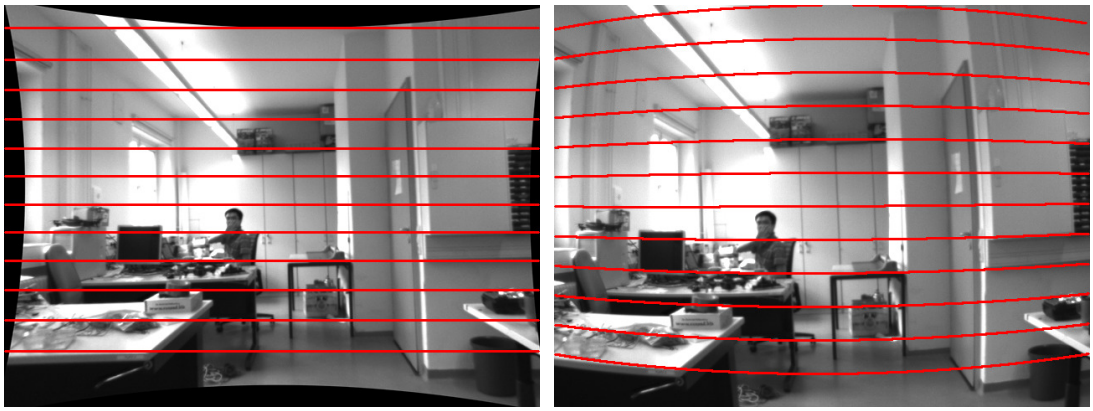


Figure 3.9: (a) Rectified camera image and (b) unrectified camera image with highlighted epipolar lines / curves.

curacy. Hence w provides a trade-off between matching robustness and computational efficiency. The method proposed so far is formally described in Algorithm 3.3 for the case of rectified input images.

3.4.3 Processing of Unrectified Stereo Images

As discussed in Section 2.2.2 on page 17, the first processing step in a stereo matching system is usually image rectification. Compared to common dense stereo matching algorithms, this operation can be performed relatively fast. However, since our MAV is a very computationally constrained platform and we opt for sparse stereo matching, we would like to avoid a preliminary rectification step. For sparse stereo matching, this can easily be done by just rectifying the locations of the matched image features. While this can save much time during the correlation process, this strategy cannot be applied to the previously introduced dense consistency / uniqueness check.

During the consistency / uniqueness check, we are required to traverse the epipolar lines in the left image. Hence, rather than rectifying the left image such that epipolar lines are horizontal, we can instead compute the epipolar lines in the unrectified camera image. Because of the persisting lens distortion, the epipolar lines are depicted as *epipolar curves*, as can be seen for one example in Figure 3.9a and 3.9b. We intend to precompute these epipolar curves in order to allow for a fast traversal when performing the dense consistency check.

To implement this process efficiently, we need to precompute three distinct lookup tables that are each arranged in a two-dimensional matrix with the size of an input image. For the first table $\mathbf{E}_{uv}$, we consider an unrectified left image location (u_{eL}, v_{eL}) and determine its rectified counterpart $(\tilde{u}_{eL}, \tilde{v}_{eL})$. A table entry for $\mathbf{E}_{uv}$ is then determined by

Algorithm 3.3: Sparse stereo matching algorithm.

```
/* Detect features                                                    */
leftFeatures := detectLeftFeatures();
rightFeatures := detectRightFeatures();
bestMatches := ∅;
/* Only perform non-maxima suppression for left image features.      */
suppressNonMaxima(leftFeatures);

/* First match left and right features.                               */
foreach (u_L, v_L) ∈ leftFeatures do
    minCost := ∞;
    minRightFeature := ∅;
    /* Determine right feature matching candidates.                   */
    cadidates := {(u_R, v_R) ∈ rightFeatures | (v_R − v_L)^2 ≤ 1 ∧ (u_R − u_L) ≤ d_max};
    foreach (u_R, v_R) ∈ candidates do
        /* Perform correlation using census transformation.           */
        cost := correlate((u_L, v_L), (u_R, v_R));
        /* Test whether we found a new best match                     */
        if cost < minCost then
            minCost := cost;
            minRightFeature := (u_R, v_R);
        end
    end
    if minRightFeature ≠ ∅ then
        bestMatches.append({(u_L, v_L), (u_R, v_R), minCost});
    end
end

/* Perform consistency / uniqueness check.                           */
foreach match ∈ bestMatches do
    {(u_L, v_L), (u_R, v_R), minCost} := match;
    /* Match densely in opposite direction.                           */
    u := u_R;
    while u < u_R + d_max do
        cost := correlate((u, v_L), (u_R, v_R));
        if cost < minCost / q then
            /* Match is non-unique or non-consistent.  Remove it.     */
            bestMatches.remove(match);
            break;
        end
        /* Increment u by step width w.                               */
        u := u + w;
    end
end
```

using the following equation:

$$\mathbf{E}_{uv} = v_{eL}, \text{ where } u = u_{eL} \text{ and } v = \tilde{v}_{eL}. \tag{3.7}$$

Thus, this table contains the vertical coordinates of the epipolar curves in the left camera image. Next we compute the table $\mathbf{I}_{uv}$:

$$\mathbf{I}_{uv} = \tilde{v}_{eL}, \text{ where } u = u_{eL} \text{ and } v = v_{eL}. \tag{3.8}$$

We use this table together with $\mathbf{E}_{uv}$ to quickly select the best matching epipolar curve for an unrectified left image location.

Finally, we require one more lookup table $\mathbf{Z}_{uv}$. Given a rectified horizontal right image coordinate $\tilde{u}_{zR}$ and an unrectified vertical left image coordinate v_{zL}, we attempt to find the point $z_L = (u_{zL}, v_{zL})$ that meets the following conditions:

- There exists a point $\tilde{z}_R$ in the rectified right image with horizontal coordinate $\tilde{u}_{zR}$.
- $\tilde{z}_R$ lies on the epipolar line corresponding to z_L.
- The stereo disparity between $\tilde{z}_R$ and the rectified image location of z_L is 0.

The coordinate u_{zL} is then stored in $\mathbf{Z}_{uv}$ as follows:

$$\mathbf{Z}_{uv} = u_{zL}, \text{ where } u = \tilde{u}_{zR} \text{ and } v = v_{zL}. \tag{3.9}$$

While the computation of the other tables is straightforward, we need to employ an iterative computation scheme for calculating $\mathbf{Z}_{uv}$. The method that we use for this task is shown in Algorithm 3.4. In this approach, we use $\tilde{u}_{zR}$ as initial approximation for u_{zL}. We then rectify our estimate for z_L and determine the stereo disparity d towards $\tilde{z}_R$. As we want to achieve a disparity of $d = 0$, we refine our estimate for u_{zL} by subtracting d. This process is repeated until either the maximum number of iterations has been reached, or the disparity is smaller than $\varepsilon > 0$.

Once we have computed the three lookup tables, we can perform the dense consistency / uniqueness check as follows: Given the unrectified left and right image locations of a matching feature pair $p_L = (u_{pL}, v_{pL})$ and $p_R = (u_{pR}, v_{pR})$, and the lookup tables $\mathbf{E}_{uv}$ and $\mathbf{I}_{uv}$, we obtain the epipolar curve $e_L(u)$ in the left image that is closest to p_L:

$$e_L(u) = \mathbf{E}_{uv}, \text{ where } v = \mathbf{I}_{u_{pL} v_{pL}}. \tag{3.10}$$

Next, we obtain the coordinates (u_{zL}, v_{zL}) for point z_L that matches the conditions we pointed out above:

$$u_{zL} = \mathbf{Z}_{uv}, \text{ where } u = \tilde{u}_{pR} \text{ and } v = v_{pL}, \tag{3.11}$$

$$v_{zL} = e_L(u_{zL}), \tag{3.12}$$

Algorithm 3.4: Iterative computation scheme for u_{zL}.

```
/* Initial estimate for u_zL is ũ_zR                                  */
u_zL := ũ_zR;
d := ∞;
/* Refine estimate iteratively.                                       */
for i := 1 to maxIterations do
    if |d| < ε then
        /* Approximation is sufficiently accurate.                    */
        break;
    end
    /* Get v_zL using the current estimate for u_zL.                  */
    ṽ_iL := I_uv, where u = u_zL and v = v_zL;
    v_zL := E_uv, where u = u_zL and v = ṽ_iL;
    /* Rectify current estimate.                                      */
    (ũ_zL, ṽ_zL) := rectifyLeftPoint(u_zL, v_zL);
    /* Determine how far we are away from 0-disparity.                */
    d := ũ_zL − ũ_zR;
    /* Refine estimate for u_zL by subtracting difference from
       0-disparity.                                                   */
    u_zL := u_zL − d;
end
```

where $\tilde{u}_{pR}$ is the rectified horizontal coordinate of p_R. The point z_L is our starting point for the dense consistency check. We then traverse the epipolar curve $e_L(u)$ in the range of

$$u_{zL} - d_{max} < u < u_{zL}. \tag{3.13}$$

Because this method only requires a few table lookups, and we only process image locations for which we previously found a stereo correspondence, this method is much faster than performing a full rectification of both input images. A quantitative evaluation of the achieved performance gain is provided in Section 3.5.4 on page 50.

3.5 Evaluation

So far we have introduced the new exFAST feature detector and a novel sparse stereo matching algorithm. In this section, we present a thorough evaluation of both methods. We start with an independent examination of the exFAST detector, and then continue with an evaluation of the combination of exFAST with our stereo matching method. Furthermore, we perform a comparison to other sparse stereo algorithms and different adaptations of our own stereo method that make use of other feature detectors.

3.5.1 Feature Detector Performance

The main contribution of the proposed exFAST detector is an adaptive threshold that is aimed at detecting more points in low-contrast image regions. At the same time, fewer points should be detected in high-contrast image regions, where an unmodified FAST detector tends to detect many features within short distance to one another. Thus, the detected features should be more evenly distributed over the input image.

To evaluate the effect of our adaptive threshold, we hence require a quantitative metric for the distribution of the detected features. We attempt to measure the 'clusteredness' of a feature distribution, for which we divide the input image into a regular grid of 10×10 rectangular cells. For each cell, we determine the fractional amount of points that are within the cell's boundaries. We then use the standard deviation of those fractions as our clusteredness measure s:

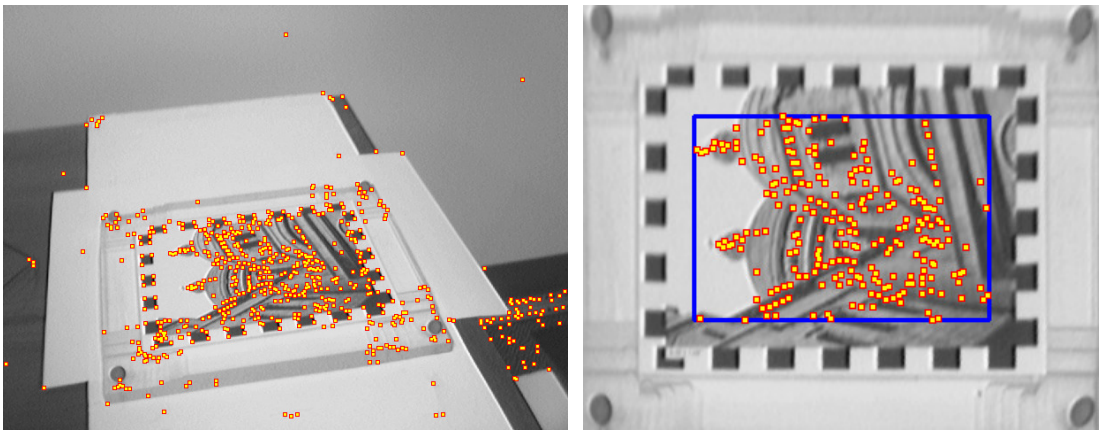
$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{C_i}{|C|} - \frac{1}{n} \right)}, \tag{3.14}$$

where n is the total number of grid cells, C is the set of grid cells and C_i is the set of features within grid cell i. If the features are uniformly distributed, each cell roughly contains the same fraction of features, which causes s to reach a small value. On the other hand, if the features are highly clustered, then few cells contain the bulk of all features and s will be large.

With this metric, a comparison of exFAST against an unmodified FAST detector and the Harris detector was performed. This comparison included another FAST-based algorithm, which only uses the adaptive threshold that was introduced in Section 3.3.1 on page 31. Likewise, a FAST algorithm with only the averaged center from Section 3.3.2 on page 34 was also included. This allows us to individually judge the contributions of each of these two extensions.

The five chosen algorithms were evaluated using the unconstrained motion pattern sequences of the feature detection evaluation data set published by Gauglitz *et al.* (2011). In these sequences, various flat reference pictures are filmed while performing random camera movements. An example frame from one of these sequences is shown in Figure 3.10a. The dataset was repeatedly processed with each algorithm, while varying the parameterizations of the algorithms. This was done in order to receive solutions with different feature counts. For the Harris detector, FAST and FAST with averaged center, the detection threshold t was varied, while the adaptivity factor a was varied for exFAST and FAST with adaptive threshold. For the latter two, the constant threshold $t_c = 10$ was chosen, which appears to provide a good trade-off between detection speed and clusteredness.

In Figure 3.11a, the average of our clusteredness metric is plotted against the average feature count for each examined algorithm. These results show that compared to the Harris detector and ordinary FAST, exFAST provides significantly less clustered feature distributions in most cases. However, with decreasing adaptivity factor a and an increas-



(a) (b)

Figure 3.10: (a) Frame of evaluation data set with features detected by exFAST and (b) generated top-view of reference picture with selected features.

ing feature count, the difference between exFAST and FAST reduces and eventually becomes 0. The reason for this behavior is that we employ an ordinary FAST detector with threshold t_c as the first detection stage. Hence, when reducing a, the results of exFAST become more and more similar to those of FAST with threshold t_c. When $a = 0$, both detectors provide identical results. Figure 3.11a further shows that the reduced clusteredness can be credited to the adaptive threshold, as FAST with adaptive threshold performs almost identical to exFAST. Although FAST with averaged center also provides a reduction in clusteredness, this effect seems insignificant for the combined approach.

Furthermore, the repeatability of the examined feature detectors has been evaluated. Accurate ground truth information is available for the camera movements in the used evaluation sequences. Hence, we are able to warp the projection of the recorded reference picture into a top-down view, as shown in Figure 3.10b for the previous example frame. This top-down view serves as reference view for our repeatability analysis. By projecting two features from different frames into this reference view, we can determine if those features correspond to the same point in the reference picture. For this task, we only consider features within a defined Region Of Interest (ROI), which has been highlighted in blue in the given example.

With the projected feature coordinates, we are able to calculate a repeatability metric ρ for two feature sets that originate from different frames. We use the method proposed by Gauglitz *et al.* (2011), which, in simplified notation, can be expressed as follows:

$$\rho = \frac{|\{(p_a \in S_i, p_b \in S_j) \,|\, \Delta_r(p_a, p_b) < \varepsilon\}|}{|S_i|}, \tag{3.15}$$

where S_i and S_j are the sets of features that were detected in frames i and j. The function

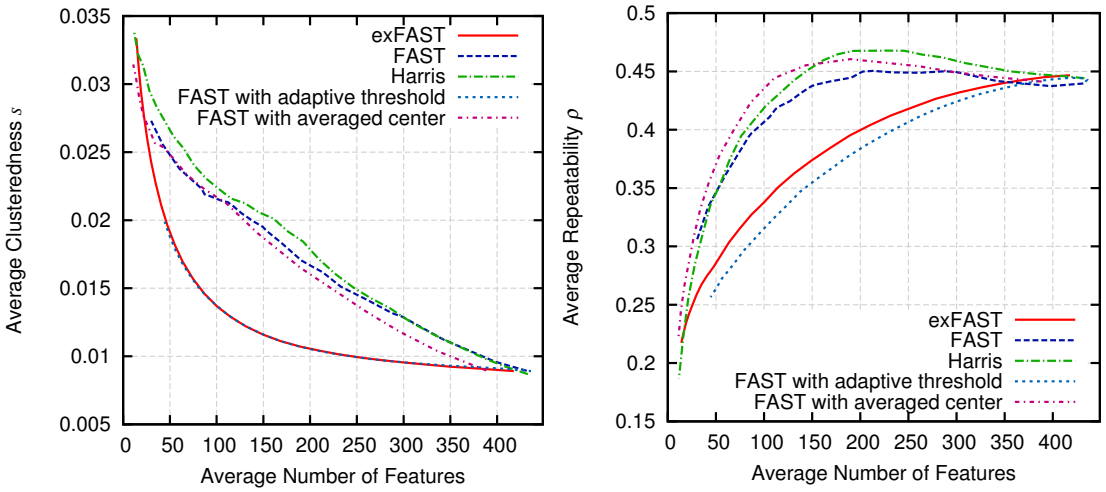


Figure 3.11: (a) Feature clusteredness and (b) repeatability evaluation.

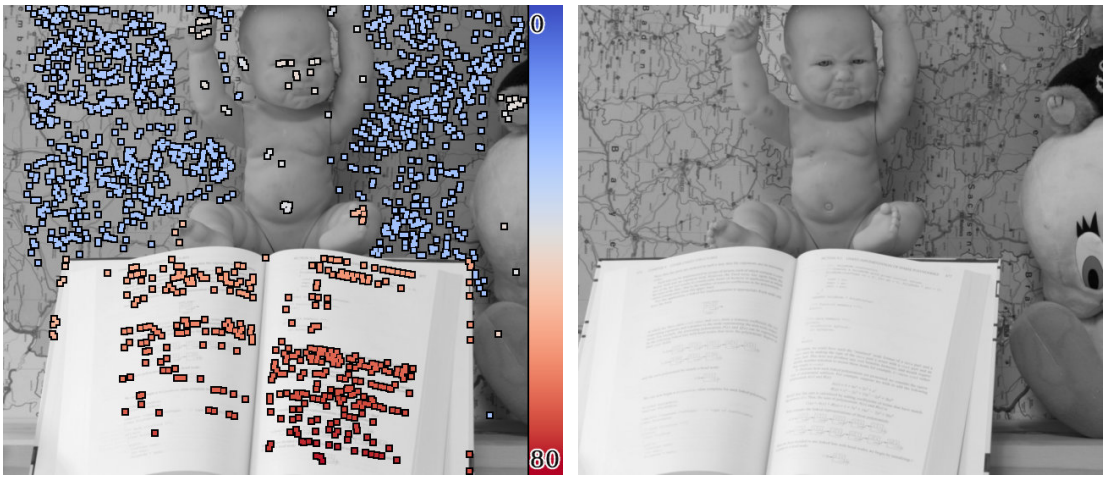
Δ_r provides the distance between two points when projected into the reference view. The threshold $\varepsilon > 0$ determines the maximum distance in the reference image up to which two features are considered to correspond to the same point. In accordance to Gauglitz *et al.* (2011), we set ε to 2 pixels.

This repeatability metric has been computed for each pair of consecutive frames from the used evaluation dataset. The average repeatability for all such frame pairs and all sequences are shown in Figure 3.11b for the five considered feature detectors with varying parameterizations. This diagram reveals that the adaptive threshold causes a significant repeatability reduction. The averaged center, on the other hand, yields a slight repeatability increase when used individually or in conjunction with the averaged center, which matches our assumption from Section 3.3.2 on page 34. In fact, FAST with averaged center ensures a higher repeatability than the Harris detector for large thresholds.

The lower repeatability of exFAST when compared to plain FAST or the Harris detector suggests that exFAST is the 'worst performing' feature detector. However, we in our case are interested in stereo matching, which is a task that differs significantly from the common use of feature detection algorithms. We will see in the following section that despite the lower repeatability, exFAST provides a better performance for this task.

3.5.2 Combined Feature Detection and Stereo Matching

For evaluating the stereo matching performance, we use the 2006 Middlebury College dataset (Scharstein and Pal, 2007). The 2001 and 2003 datasets from the same institution are frequently used for evaluating dense stereo algorithms. However, compared to these earlier datasets, the 2006 dataset is more challenging. This dataset contains more stereo



(a) (b)

Figure 3.12: (a) Left input image with highlighted stereo matches and (b) right input image of used evaluation dataset

pairs, encompasses a larger disparity range and features stereo pairs with untextured or repetitively textured image regions. As a comparison to previous results of dense algorithms is dispensable, we chose the 2006 dataset for our evaluation.

We use the semi-resolution version of this dataset, which has image resolutions that are closer to the VGA resolution of the cameras on-board our MAV. For processing the comprised stereo pairs, the maximum disparity d_{max} is set to 115 pixels. An example for the performance of our stereo matching system on a stereo pair from this dataset is given in Figure 3.12.

As criterion for evaluating the stereo matching accuracy, the Bad Matches Percentage (BMP) ψ is determined, which we define as follows:

$$\psi = \frac{\left|\{p \in M \,|\, (D_p - G_p)^2 \leq \varepsilon\}\right|}{|M|}, \tag{3.16}$$

where M is the set of successfully matched features, and the sets D and G represent the obtained and ground truth stereo disparities. The threshold ε is set to 1, which is in accordance with the threshold commonly used for evaluating dense algorithms (see Scharstein and Szeliski, 2002; Szeliski *et al.*, 2007). However, the resolution of the ground truth for the used dataset is at only 0.5 pixels, which means that the BMP we determine might be overestimated.

We use the optimized stereo matching implementation with matching windows of size 6×5 pixels in combination with three different feature detectors. The chosen methods are our exFAST detector, a plain FAST detector and the Harris detector. For each feature detector, three different parameterizations were selected, such that all algorithms detect

Table 3.1: Parameter sets for the examined feature detectors during stereo matching.

Method	**Parameters**		
exFAST adaptivity (a)	0.5	1.0	1.5
FAST threshold (t)	12	15	20
Harris threshold	$2 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$1.5 \cdot 10^{-5}$

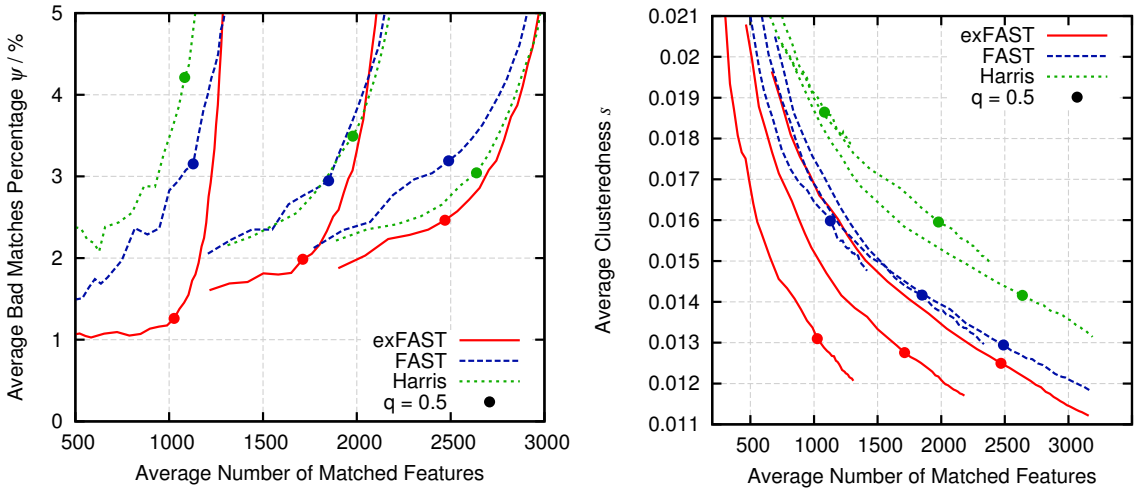


Figure 3.13: (a) Stereo matching accuracy and (b) clusteredness of stereo matching results for different feature detectors and parameterizations.

similar numbers of features. The chosen parameters are listed in Table 3.1.

Our stereo matching system was run with each feature detector while varying the uniqueness factor q. For each test run, the average BMP of all stereo pairs in the evaluation data set was determined. In Figure 3.13a, the average BMP for each feature detector is plotted against the number of received matches. These results show that our proposed stereo matching method provides a significantly higher accuracy when used in conjunction with exFAST, as opposed to FAST or the Harris detector.

With increasing the feature count, the results received for the examined feature detectors converge. The likely explanation for this behavior is that the feature detector becomes unimportant for situations with high feature counts. In this case, we are close to dense stereo matching. Hence, the result is mostly influenced by the used matching strategy, and less by the choice of features. In the parameter range that was examined for our evaluation, however, the BMP received with exFAST remained notably below the BMP for FAST or the Harris detector.

For the features that were successfully matched during stereo matching, the feature distribution was also examined. The resulting clusteredness s for all feature detectors and parameterizations are shown in Figure 3.13b. This figure has been plotted against

the average number of features, while varying the uniqueness factor q. According to this figure, exFAST causes the least clustered distribution of successfully matched feature pairs, while the results for the Harris detector show the highest clustering. The low clusteredness of the features that were detected by exFAST matches our findings from the previous section. Furthermore, Figure 3.13a and 3.13b reveal that the uniqueness factor q provides a trade-off between the number of successfully matched features, BMP and clusteredness.

Even though the proposed exFAST detector was shown to have a lower repeatability when compared to standard FAST or the Harris detector, the received stereo matching results are of a significantly higher quality. This observation appears to be contradictory. A possible explanation for this behavior might be as follows: If the feature distribution tends to be clustered, we receive regions with high feature-detection probability. Since the feature distribution depends on local intensity variations in the input image, we can expect that images of the same scene exhibit similar feature distributions. Hence, when performing a repeatability analysis for two frames by mapping all features into a common reference view, there is a high probability that a feature from a dense feature area in one frame will have a close neighbor from the other frame. We thus conclude that the repeatability measure is biased towards clustered feature distributions.

For stereo matching, however, we expect to see the opposite effect. From an area with high feature density, we receive many features that originate from the same image region. Hence, their local pixel neighborhood is also likely to exhibit a similar appearance. For accurate stereo matching, however, we prefer features with a unique appearance, which we are more likely to receive if the features originate from different sections of an input image. Thus, we expect stereo matching to be biased towards unclustered feature distributions.

As mentioned in Section 3.4 on page 35, it is possible to perform the combined consistency and uniqueness check with larger step-widths w. As a large step width significantly reduces the number of required matching operations, we receive a higher run-time performance in this case. The effect of a varying step-width w on the received BMP is shown in Figure 3.14. For this experiment, exFAST was used as feature detector, with different values for the adaptivity factor a. Furthermore, the uniqueness factor $q = 0.5$ was chosen, which provides good results on the evaluation dataset. As expected, the BMP increases gradually with increasing w. It is thus possible to adjust w in order to receive a trade-off between matching accuracy and the achieved run-time performance.

3.5.3 Comparison with Other Stereo Matching Methods

In Figure 3.15a, the BMP obtained with our algorithm is compared to the results for three alternative methods. Our algorithm has been labeled *Dense Consistency* in this figure. The three alternative methods include a plain sparse stereo algorithm (*Sparse*), which simply matches features found in the left input image to the features in the right image that are close to the same epipolar line. Furthermore, the algorithm *Dense Right*

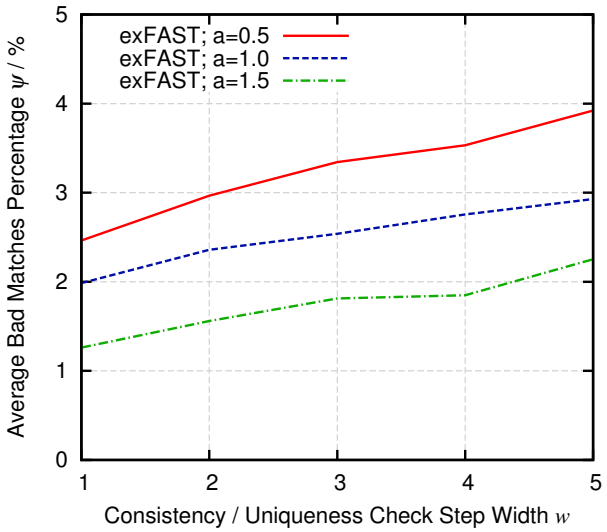


Figure 3.14: Consistency check step width vs. accuracy.

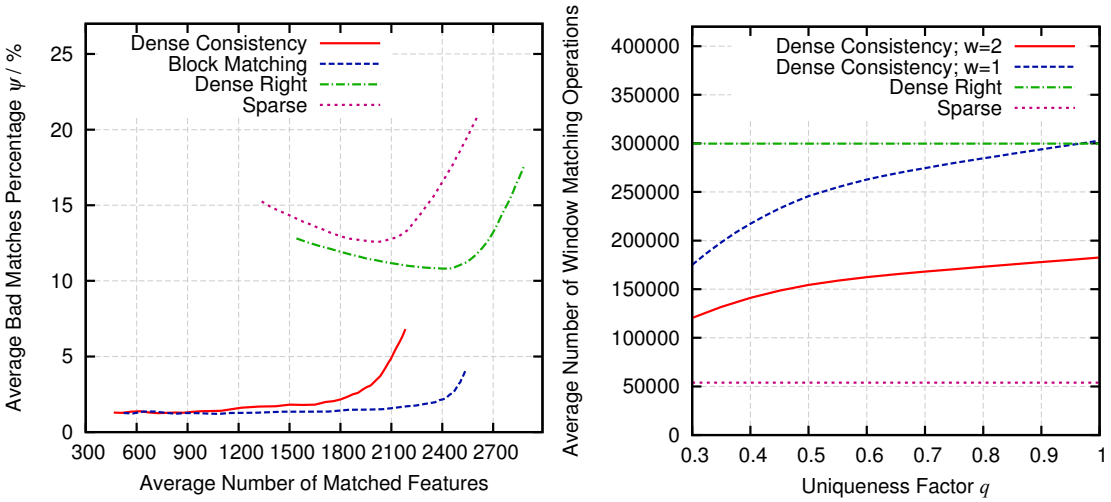
is included that densely matches all features from the left image to the valid disparity range in the right image. Finally, the last method is a dense block matching algorithm (*Block Matching*). For this method, we only evaluate the pixel locations where a feature has been detected.

As feature detector for all stereo matching algorithms we use our exFAST detector with adaptivity factor $a = 1.0$, which is the parameterization that we use for the rest of this thesis. Furthermore, all algorithms use the same correlation method that we presented in Section 3.4.1 on page 35. All algorithms apply our consistency and uniqueness check with varying q. Except for our proposed method *Dense Consistency*, only the cost values calculated during stereo matching are considered for this step.

The given results show that *Dense Consistency* and *Block Matching* greatly outperform *Sparse* and *Dense Right*. For *Block Matching*, the received BMP is lower than for *Dense Consistency*. However, for lower feature counts, which are caused by a lower uniqueness factor q, this difference becomes marginal.

The algorithm *Sparse* shows the worst performance, which was expected as this algorithm performs the least matching operations. As a surprise, *Dense Right* also shows a rather poor performance despite the fact that this method generally requires more matching operations than *Dense Consistency*. The key difference between both algorithms is that *Dense Consistency* examines the entire disparity range relevant for the combined consistency and uniqueness check. *Dense Right*, on the other hand, only considers the image locations for which a cost has previously been calculated during the left-to-right stereo matching. Hence, we can conclude that a dense processing is more relevant during the consistency and uniqueness check, rather than for the initial matching stage.

For judging the achievable run-time performance of each algorithm, we compare the average number of matching operations that each method requires for processing the



(a) (b)

Figure 3.15: (a) Accuracy of different stereo-analysis methods and (b) matching operations by stereo-analysis method.

stereo pairs in the evaluation dataset. The results are displayed in Figure 3.15b for the algorithms *Dense Consistency* with step-widths $w = 1$ and $w = 2$, and for *Sparse* and *Dense Right*. The results for *Block Matching* have been omitted in this diagram, as this method requires $3.8 \cdot 10^7$ matching operations, which is far more than for any other algorithm.

For *Dense Consistency*, the number of matching operations depends on the uniqueness factor q, as low values for q allow for an early rejection of wrong matches. For the other algorithms, the number of matching operations remains constant. Our results show that increasing the step-width w to 2 almost halves the number of matching operations that are required by our proposed *Dense Consistency* algorithm. Hence, by increasing w, we can significantly reduce processing time for stereo matching.

3.5.4 Real World Performance Evaluation

To judge the performance of our stereo vision system in real-world situations, one further evaluation was performed on an unrectified stereo sequence. This time, our stereo matching implementation that is targeted for our MAV platform, which has a matching window size of 5×5 pixels, was also considered. For this evaluation, the Queen Street sequence from the EISATS dataset number 9 was used, which has been published by Hermann, S. and Morales, S. and Klette, R. (2011). This sequence was recorded outdoors with a car-mounted stereo camera with a resolution of 640×480 pixels. An example for the performance of our method with one stereo frame from this stereo sequence was previously shown in Figure 3.1a on page 22.

Table 3.2: Processing rates for images of size 640×480 pixels, on different hardware platforms.

Architecture	One Core	Two Cores
Regular PC (Intel i5 dual core, 3.3 GHz)	175 fps	214 fps
MAV (Intel Core 2 Duo, 1.8 GHz)	71 fps	89 fps

The parameters used for processing this sequence are: adaptivity $a = 1.0$, uniqueness factor $q = 0.7$, consistency check step width $w = 2$, and maximum disparity $d_{max} = 70$. Compared to the previously used dataset, which was recorded in a controlled lighting environment, this real-world sequence is considerably more challenging. This is why a higher value for the uniqueness factor q was chosen, which should lead to a better suppression of erroneous matches.

The sequence comprises a total of 400 stereo pairs and our stereo method was able to successfully match an average number of 633 features with the standard and 605 features with the embedded implementation for our MAV platform. The standard implementation was run on a computer with an Intel i5 dual core CPU with 3.3 GHz, and the embedded version was run on the actual MAV hardware. Two versions of each implementation have been examined: one sequential version and a parallel version that utilizes two CPU cores by means of parallel programming techniques.

Table 3.2 shows the results that were obtained on both platforms. When using both cores on the regular PC, we achieve an average processing rate of 214 frames per second. But even when run on just one core of our MAV platform, the average processing rate is still far above the desired processing rate of 30 Hz. This should leave sufficient processing resources for high-level vision tasks that we are required to run on-board our MAV in addition to stereo matching.

The single threaded test run on the regular PC was repeated with a varying adaptivity factor a. The received processing times are shown in Figure 3.16, where the point for $a = 1.0$ has been highlighted. This figure also includes the processing times of a version that performed full image rectification. For rectification, the method contained in the OpenCV library (see Itseez, 2013) has been used. This method is particularly efficient, as it precomputes a rectification transformation and stores it in a rectification map. With this map, OpenCV then performs a fast transformation of the input image, by making use of the SSE instruction set.

In the provided figure we can see that our implementation, which does not perform full image rectification, has significantly lower processing times. For the examined parameter range, the achieved performance improvement varies between 2.9 ms and 3.1 ms. Given the low total processing time of our stereo matching method, which according to Figure 3.16 can be as low as 3.4 ms, the time required for image rectification can have a high impact on the overall processing rate.

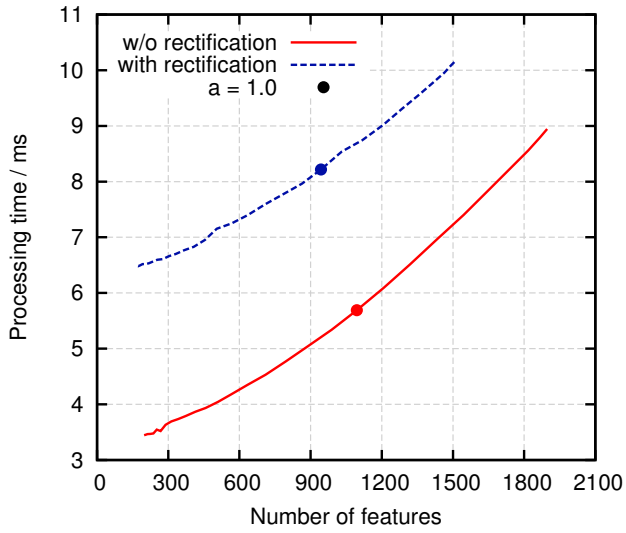


Figure 3.16: Stereo matching performance evaluation.

3.6 Summary and Discussion

In this chapter we introduced an efficient sparse stereo matching system, which we specifically developed for application on-board our MAV. Since the on-board processing resources that are available on our MAV platform are particularly scarce, the most important design criterion for our stereo matching system was high computational efficiency. Most current work on stereo vision is primarily focused on dense stereo methods, which usually only achieve low processing rates when run on a CPU. This is why we opted for a substantially faster sparse stereo matching approach.

Our aim is to utilize stereo matching to achieve accurate vision-based navigation for our MAV. Given that many visual navigation methods only process a sparse set of salient image features, performing sparse instead of dense stereo matching does not seem to be a limitation in our case. Rather, a high processing rate is important, which is why slow dense stereo matching algorithms are not suitable for our MAV.

Thus, we have aimed at creating a sparse stereo matching method that provides accurate results, but still remains computationally efficient. We achieved this goal by using several techniques. First, we presented a new feature detector based on the FAST algorithm. As shown in our evaluation, features detected with this new method exhibit an evidently less clustered distribution, when compared to ordinary FAST or the Harris detector. Although our algorithm performs worse than FAST or the Harris detector for common repeatability measures, its performance was clearly superior in a combined feature detection and stereo matching system.

The performance of our algorithm might also be superior for other feature-based vision tasks. What our evaluation shows is that repeatability is not necessarily a sufficient measure for quantifying the performance of a feature detector. Rather, the uniqueness of the

detected features should also be taken into consideration, and a detector's performance for specific applications should be examined.

The second contribution of this chapter is a new stereo matching algorithm. Even though this algorithm performs sparse stereo matching, we apply a dense consistency and uniqueness check, which successfully eliminates most erroneous matches. We are able to perform this dense consistency and uniqueness check without prior image rectification. We achieve this by using a method based on lookup tables, which traverses the epipolar curves in an unrectified input image. In a performance evaluation, we have shown that this approach saves valuable computation time when compared to full image rectification.

We further evaluated the matching accuracy of the resulting stereo matching system and have shown that it produces accurate results with only few erroneous matches. Compared to several other algorithms that also provide a sparse set of stereo correspondences, our method produces significantly fewer false matches, and it can compete in accuracy with a dense block matching algorithm. Furthermore, we have shown that our stereo matching system is fast enough for real-time stereo matching on a CPU, while leaving sufficient processing resources for higher-level vision tasks. This is even the case when stereo matching is run on our MAV platform. This highly efficient stereo matching system is the first step in the construction of our autonomous MAV.

Chapter 4

Stereo-Based Visual Navigation

4.1 Introduction

To enable our MAV to perform autonomous flight maneuvers, we need to equip it with a method for autonomous navigation. We define autonomous navigation as the ability of an MAV to fly from one location to another, without the need for human intervention. For this to be possible, the MAV has to be aware of its current pose throughout the flight. If the MAV is able to track its pose, it can sense its positional and rotational movements. This allows us to use a control algorithm to stabilize the MAV's flight, which is particularly important for an inherently unstable quadrotor. With the current MAV pose, we can then determine the MAV's relative position towards the dedicated target location. We are hence able to steer the MAV towards this target location and thus achieve autonomous navigation.

If we do not rely on an external system for pose inference, the MAV has to ensure pose estimates by solely using its on-board sensors. This task is usually much simplified if the environment of the MAV is known in advance, in which case the MAV can be equipped with an appropriate map before taking off. It can then use this map and its on-board sensors for the identification of known landmarks. Using the observed relative position of these landmarks towards the MAV, it is possible to infer the current MAV pose.

If we want to operate our MAV in unknown environments, this task is more difficult. Specific solutions for the pose inference problem have been proposed in literature for a range of sensor types, such as laser scanners, monocular cameras or RGBD cameras. Since our MAV is equipped with four cameras in two stereo configurations, we are mainly interested in methods that rely on stereo vision. We can use the efficient stereo vision system that we presented in the previous chapter to develop a stereo vision based pose estimation method for our MAV.

In this chapter, we present two complete MAV systems that are able to navigate autonomously. The first system, which was initially presented at the 2012 Autonomous Mobile Systems Conference (AMS), uses only the forward-facing camera pair of our MAV platform (Schauwecker *et al.*, 2012b). The imagery of these cameras is used for running a stereo SLAM system that has been simplified for meeting the necessary performance requirements. To the author's knowledge, this was the first demonstration of

an MAV that relies on stereo vision for autonomous navigation, and does not depend on visual markers or an otherwise known environment.

The second system that we present in this chapter extends the former one by also employing the downward-facing camera pair. The stereo matching results received from this camera pair are used for detecting and tracking the visible ground, for which a planar model is assumed. This allows us to determine a redundant pose estimate, which in this case is relative to the assumed ground plane. This method was initially published at the 2013 International Conference on Unmanned Aircraft Systems (ICUAS) (Schauwecker and Zell, 2013), while an extended version of this work was published in the Journal of Intelligent & Robotic Systems (JINT) (Schauwecker and Zell, 2014a). To the author's knowledge, this was the first demonstration of an MAV that is able to perform stereo matching for two stereo camera pairs on-board and in real-time.

4.2 Related Work

As already mentioned, the key challenge in autonomous navigation is the estimation of the current MAV pose. Hence, in this section we have a close look at existing methods for pose inference. Since our MAV is equipped with two camera pairs for stereo vision, we are particularly interested in stereo vision based methods. Furthermore, we have a look at existing autonomous MAVs that rely on stereo vision or other sensor types for environment perception.

4.2.1 Visual Odometry

A simple approach for estimating the pose of a moving camera is *Visual Odometry* (VO). In the case of VO, the current camera pose is incrementally tracked from one camera frame to another. A two-part introduction to VO, with a survey of existing methods and techniques in this field, has recently been published by Scaramuzza and Fraundorfer (Scaramuzza and Fraundorfer, 2011; Fraundorfer and Scaramuzza, 2012).

There exist both monocular and stereo vision based approaches for VO. In fact, one of the first VO methods, which was published by Moravec (1980), already relied on stereo vision. In this work, Moravec implemented a VO system for the navigation of a mobile robot, which was equipped with a linearly translatable camera. While the robot is standing still, the camera records several images at different camera translations. The robot then moves a short distance and records another set of images. VO is then applied to obtain an estimate for the traveled distance and direction. Feature points from different camera translations at one robot position are used for sparse stereo matching. The resulting sets of 3D points from both robot positions are then aligned using a weighted least squares method. As a result, an alignment transformation is received, which provides the pose update from the first to the second robot position.

It is no coincidence that the early VO system from Moravec uses stereo vision rather than monocular vision. As previously mentioned, a monocular camera allows us to fully observe camera rotations, but camera translations are only observable with respect to an unknown scaling factor. Only if an estimate for this scaling factor can be obtained by other means, it is possible to recover the camera translation. This circumstance makes the design of monocular VO systems challenging. Nevertheless, several robust monocular VO implementations have been proposed in more recent times. Popular examples are the VO systems published by Corke *et al.* (2004), Nistér *et al.* (2006) or Scaramuzza and Siegwart (2008). Given our focus on stereo vision based methods, we omit a more detailed discussion of these approaches.

Despite the early groundbreaking work by Moravec and others, the term Visual Odometry was coined much later by Nistér *et al.* (2004), who according to Scaramuzza and Fraundorfer (2011) presented the first real-time and long-run VO system. In fact, Nistér *et al.* proposed two VO implementations, of which one is based on monocular and one on stereo vision. The stereo method first detects corner features in the left and right camera images, which are then correlated using a robust NCC-based matching scheme. With the same matching method, features are tracked independently over several camera frames. Using a robust estimator that is based on RANdom SAmple Consensus (RANSAC) (Fischler and Bolles, 1981) and the three-point algorithm proposed by Haralick *et al.* (1994), an estimate for the camera movements is obtained. Different optimizations and refinement strategies are then applied in subsequent steps.

A more recent stereo vision based VO system has been proposed by Kitt *et al.* (2010). The first step in this method is again the extraction of corner features, which are used for stereo matching. The subsequent motion estimation is based on the trifocal tensor (Hartley and Zisserman, 2003), which are three 3×3 matrices that encapsulate the projective geometric relationships among three camera views. Using the trifocal tensor, a RANSAC based estimator is applied to extract a robust motion estimate from two consecutive stereo frames. In a final step, a temporal integration of the extracted motion information is performed.

Another recent stereo vision based VO method, which has gained much popularity, was published by Konolige *et al.* (2011). Unlike in the previously discussed work, this method does not rely on corner features, but uses a more stable feature detector by Agrawal *et al.* (2008). Features are only extracted from the left input image, and then densely matched to the right image. The corresponding feature-pairs are used to obtain a motion estimate from two consecutive stereo frames, by using a robust estimator based on RANSAC and the three-point algorithm. The system preserves the most recent pose estimates and tracked features, which are then optimized using Bundle Adjustment (BA) (Triggs *et al.*, 1999; Engels *et al.*, 2006). This optimization step leads to a significant improvement in pose estimation accuracy.

4.2.2 Visual Simultaneous Localization and Mapping

The key problem of VO is that camera motions are only tracked from one frame to another. Not only does this favor error accumulation, but it also limits the tracking robustness. If tracking ever fails for one camera frame, all subsequent pose estimates will be erroneous. Hence, nowadays VO has mostly been superseded by more accurate and more robust map-based methods.

If we possess a map that contains the 3D-locations of salient scene features, then the camera movements can be tracked by identifying those features in the current camera image. Such a map can be created on the fly, by populating it with features that the camera currently observes. However, for adding new features to the map, we already require knowledge of the current camera pose. Hence, the problem of creating the map and finding the camera pose are strongly interconnected.

The solution is to simultaneously create the map and determine the camera pose, which was coined Simultaneous Localization and Mapping (SLAM) by Durrant-Whyte *et al.* (1996). In this case, an initial map is created by using features from an initial camera frame and by assuming a default camera pose. When the camera is moved, localization is performed against this map and the resulting pose is used to expand the map with newly observed features.

An introduction to SLAM with a review of many less-recent methods and techniques can be found in the two-parts tutorial published by Durrant-Whyte and Bailey (Durrant-Whyte and Bailey, 2006; Bailey and Durrant-Whyte, 2006). Unlike for VO, the reliance on stereo vision was less common in early SLAM systems. For a considerate time span, filter based methods used to be the state of the art in SLAM research. These methods are based on statistical filters such as Extended Kalman Filters (EKF) (see Ribeiro, 2004) or Particle Filters (PF) (Gordon *et al.*, 1993).

An example for a successful EKF-based SLAM method is the approach published by Davison (2003), which was intended for use in small indoor spaces. In this method, the EKF has a state vector that comprises the current camera pose and all mapped feature locations. The state prediction step that is required by the EKF is performed using a motion model for camera movements. Image corners serve as features that are extracted with the algorithm proposed by Shi and Tomasi (1994), and matched using normalized SSD correlation. Because the 3D location of a feature point cannot be inferred from a single camera image, features have to be observed from two different positions before they can be added to the map. The metric scale of the map is initialized with a known object that has to be visible to the camera.

Compared to EKF-based methods, methods based on PF are generally more robust against measurement errors. An example for a method that relies on PF and stereo vision was published by Sim *et al.* (2005). In particular, Sim *et al.* employ a Rao-Blackwellized Particle Filter (Doucet *et al.*, 2000), which provides improved performance in handling large state vectors. Like in the previous EKF approach, the filter state consists of the camera pose and the 3D locations of existing map features. Features are detected and

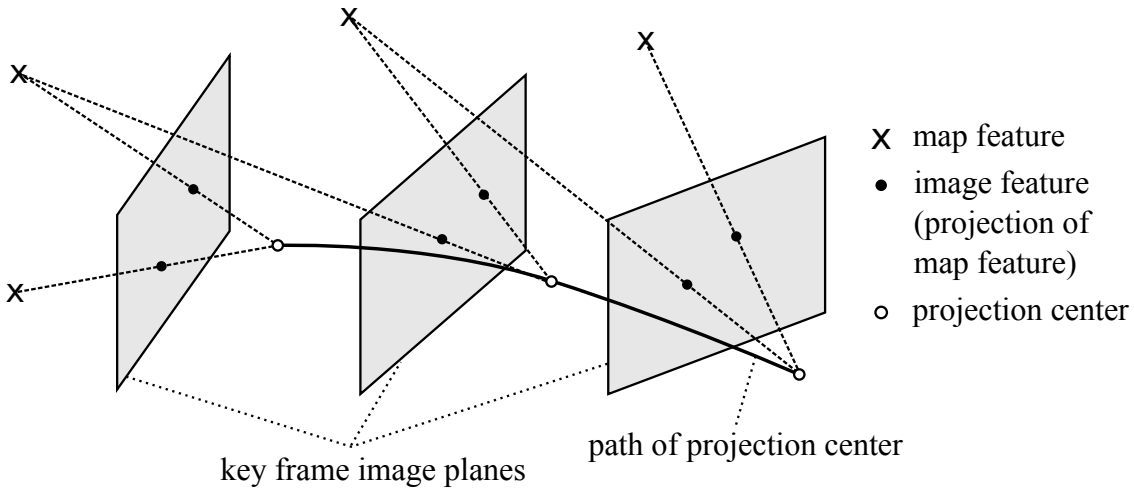


Figure 4.1: Visualization of keyframe based visual SLAM.

identified using the SIFT feature detector and descriptor, which we discussed in Section 3.2.1 on page 23. For motion prediction, a VO algorithm is used that again relies on SIFT descriptors.

Most modern SLAM systems no longer employ statistical filters. Rather, the previously mentioned BA is the predominant optimization technique today. It has been shown by Strasdat *et al.* (2010) that apart from some exceptional cases, methods based on BA generally provide “the most accuracy per unit of computation time". One of the most influential methods in this area is Parallel Tracking and Mapping (PTAM), as proposed by Klein and Murray (2007). The innovation of this monocular SLAM system was a parallelization of the SLAM problem. PTAM uses a *tracking* thread for localizing the current camera pose, while a *mapping* thread performs map expansion. Because map expansion does not have to be performed at video frame rate, it is possible to apply an expensive map optimization based on BA.

The map created by PTAM comprises *keyframes* and 3D feature locations, as illustrated in Figure 4.1. For each key frame, an image pyramid is generated and FAST corner features (see Section 3.2.1) are extracted. Whenever the tracking thread decides that the map has to be expanded, a new keyframe is passed to the mapping thread. The 3D location of an observed image feature is then obtained through triangulation with a previous keyframe from a different viewing position. A horizontal displacement of the camera is required to initialize the map with an initial set of 3D features. The magnitude of this initial displacement also serves for initializing the map scale.

PTAM has gained much popularity in the research community due to its high efficiency and its robust pose estimation results. In fact, the authors even managed to run an optimized version of PTAM on a camera phone (Klein and Murray, 2009). A large part of PTAM’s popularity can also be credited to the fact that the authors have made the source code of their implementation available online (see Klein and Murray, 2010).

This allows this method to be easily adapted by others, as has *e.g.* been done by Scherer *et al.* (2012). In this extension, additional depth information is used to obtain a map with a correct metric scale. The depth information originates from an RGB-D camera, but could in theory also be obtained through stereo vision. A depth estimate is extracted for each detected feature and considered in the BA-based map optimization.

An example for a method that is inherently based on stereo vision is the SLAM system developed by Mei *et al.* (2009). Using the SSD gradient descent algorithm proposed by Mei *et al.* (2008), the system computes an initial estimate for the pose update, by matching the current against the previous camera frame. With this initial pose estimate, potentially visible map features are projected into the current camera image and matched against extracted FAST corner features. A RANSAC estimator that is based on the three-point algorithm is then applied to compute a pose update. Stereo matching is performed only for new features that are added to the map, which saves much computation time.

Unlike PTAM, the system form Mei *et al.* uses a relative map representation. This means that the position of a map feature is stored relative to a corresponding keyframe, and the pose of a keyframe is stored relative to a neighboring keyframe, forming a *pose graph*. Such a relative representation is advantageous if the system detects that it has re-visited a previously mapped location, which is known as *loop closure*. In this case, an optimization of the existing map can be performed that compensates accumulated drift errors. In a relative map, this can be achieved by simply adding a new edge to the pose graph, while an absolute map would require an update of all keyframes and corresponding map features on the detected loop.

Another stereo vision based SLAM method, which also performs loop closure and uses a relative map representation, has been published by Strasdat *et al.* (2011). This method applies a dense stereo algorithm to the captured camera imagery. Feature points are extracted using the FAST detector and matched using BRIEF descriptors. The novelty of this method is that it uses two active windows for map optimization. The inner window consists of a set of recently mapped camera poses and map points, which are optimized using BA. A larger outer window only contains camera poses and is used for pose graph optimization. The smaller window for BA was chosen because the complexity of BA grows cubically with the number of poses. Hence, performing only pose graph optimization in the larger window keeps the computational requirements within a feasible range.

4.2.3 Autonomous Navigation for MAVs

Despite the generally large weight and high power consumption of laser scanners, there exists a significant amount of previous work on using small single-beam laser scanners for autonomous MAVs. Shen *et al.* (2011) presented one such MAV, which is able to navigate autonomously in indoor environments with multiple floors. This was made possible by using a laser SLAM method that is based on pose graph optimization. Using a monocular camera, the MAV is also able to perform loop closure detection. This MAV

was later extended to also perform autonomous path planning, which facilitates the fully autonomous exploration of unknown indoor environments (Shen *et al.*, 2012).

Another outstanding autonomous MAV that relies on laser scanners, is the fixed wing aircraft presented by Bry *et al.* (2012). This MAV is able to perform aggressive autonomous flight maneuvers at high speeds. Using an on-board laser scanner, the MAV is able to localize itself within a preexisting map. Bry *et al.* demonstrated that their MAV is able to fly in a large indoor environment while avoiding previously mapped obstacles.

Compared to laser scanners, cameras have the advantage that they can be built very lightweight and power efficient, which makes visual motion estimation a compelling alternative. As we have already discussed, however, the MAV position can only be observed with respect to an unknown scaling factor, if a single camera is used as only source of information. This is why many MAVs featuring monocular vision are only operable in specific environments.

One such example is the quadrotor MAV presented by Tournier *et al.* (2006), which relies on visual markers. As markers serve several large Moiré patterns as proposed by Feron and Paduano (2004). Using a downward-facing camera, the MAV is able to hover autonomously above an arrangement of four such patterns. Like in most earlier work on autonomous MAVs, however, image processing is not performed on-board. Instead, a ground computer is used that remotely controls the MAV.

An approach that only relies on on-board processing was published by Wenzel and Zell (2009). This MAV employs a downward-facing infrared camera that was extracted from a Wii remote controller. The camera contains an integrated circuit for tracking several infrared blobs, which allows the usage of infrared LEDs as markers. This enables the MAV to hover in a defined pose above an infrared LED pattern.

An example for an MAV with more advanced on-board image processing has been provided by Yang *et al.* (2012, 2013a). The authors presented a quadrotor MAV that is able to track a landing pad with a downward-facing on-board camera. Because the geometry of this pad is known, it is possible to infer the MAV's relative pose from the observed perspective projection. With this information, Yang *et al.* were able to demonstrate autonomous take-off, hovering and landing.

Instead of avoiding the scaling factor problem by only flying in known environments, one can alternatively derive an estimate for this factor. Such an approach has been presented by Yang *et al.* (2013b, 2014), which is based on the mentioned landing pad tracking method. In this approach, the known landing pad is required only during take-off, to initialize a PTAM-based monocular SLAM system with the correct metric scale. Once the initialization is complete, the MAV can fly a predefined trajectory and search for a specific landing site.

An approach with a continuous estimation of the scaling factor has been published by Engel *et al.* (2012). Here, the scale for a PTAM-based monocular SLAM system with a forward-facing camera is estimated by using additional measurements from an ultrasound altimeter. For this task, the authors introduced a new closed-form maximum likelihood method for integrating the measurements from both sensors. The resulting

MAV, however, does not perform any on-board processing but relies on a ground computer.

Another popular method for scale estimation is by means of measurements from an IMU (*i.e.* accelerometer and gyroscope) or an air pressure sensor, which are commonly available on many MAV platforms. Achtelik *et al.* (2011) presented one such MAV with a downward-facing camera, which performs a continuous scale estimation from accelerometer and pressure sensor readings. This is achieved by using a specifically designed EKF, which provides scale estimates that are used for a PTAM based visual SLAM method. The MAV demonstrated its ability to hover autonomously in indoor and outdoor environments. Similar systems that also rely on PTAM have been developed by Weiss *et al.* (2011) and Scaramuzza *et al.* (2013). Here, the authors use readings from an IMU, which are again processed by an EKF to receive a metric scale estimate. In both cases, the MAVs are able to follow a predefined trajectory.

If stereo vision is used instead of monocular vision, then the dependency on an unknown scaling factor vanishes, as we receive a full 3D-position for all matched points. Unfortunately however, stereo matching is generally very computationally demanding. Most less-recent work has thus focused on off-board stereo processing. For example, ground mounted stereo cameras that are focused on an MAV were used by Achtelik *et al.* (2009) and Pebrianti *et al.* (2010). A more advanced system with a forward-facing on-board stereo camera was demonstrated by Carrillo *et al.* (2012). Here, the camera images are transmitted wirelessly to a ground computer at a relatively low frame rate. The computer runs a stereo VO system and uses the obtained motion information for remote controlling the MAV.

Only very recently it has been possible to equip MAVs with sufficient processing resources to perform stereo matching on-board. The MAV presented by Heng *et al.* (2011) features a forward-facing stereo camera and runs a dense block matching algorithm with a resolution of 320×240 pixels. This MAV was later extended by Meier *et al.* (2012) to use a larger image resolution of 640×480 pixels. In both cases, however, the stereo matching results are only used for obstacle avoidance, by creating a 3D occupancy map. For navigation, the MAV still depends on visual markers. This limitation was resolved by Fraundorfer *et al.* (2012), who equipping the MAV with the integrated optical flow camera and ultrasound altimeter developed by Honegger *et al.* (2013). This allows the MAV to perform autonomous exploration tasks in indoor and outdoor environments. However, according to the numbers given for the final revision of this MAV, stereo processing only runs at a relatively low frame rate of just 5 Hz.

An example for an MAV with a downward-facing stereo camera and on-board stereo processing is the MAV developed by Tomic *et al.* (2012). The authors use a dense correlation based stereo algorithm which runs at a very low frame rate of just 3 Hz. The MAV's pose is tracked with VO and the resulting data is fused with further odometry data gained from an on-board laser scanner and readings from an IMU. Drift errors are compensated by recognizing known landmarks.

Another interesting MAV is the lighter-than-air MAV presented by Harmat *et al.*

(2012), which is equipped with three fisheye cameras of which two are arranged in a stereo configuration. However, no stereo matching is performed, but rather the imagery of each camera is tracked independently using PTAM. After tracking, the data from all cameras is fused using a pose alignment step. For this MAV, all processing is performed off-board and no autonomous control has been demonstrated.

There are two further MAVs that show similarity with the methods presented in this chapter, but have been published at a later date. The first one is the quadrotor MAV from Shen *et al.* (2013) that is equipped with a forward-facing stereo camera pair, of which one camera is fitted with a fisheye lens. The fisheye camera is used for a simplified monocular SLAM system that is limited to a local map and which operates at 25 Hz. The second camera is operated at a rate of only 1 Hz, and is used for a sparse stereo matching algorithm. By inserting the 3D points received from stereo matching into the SLAM map, the system is able to operate at a correct metric scale.

The second MAV was presented by Nieuwenhuisen *et al.* (2013), and is equipped with two stereo camera pairs with fisheye lenses. Both camera pairs are inclined towards the ground, with one pair facing forward and one pair facing backwards. In addition, the MAV is equipped with a rotating laser scanner, several ultrasound sensors, an optical flow camera with an integrated altimeter, a GPS receiver and an air pressure sensor. Features from the stereo cameras are used for a VO method, which is unfortunately not described by the authors. Furthermore, no detailed description is given on how the measurements from VO are fused with measurements from the other sensors. Apart from simulated flight results, the authors demonstrated successful obstacle avoidance.

Most of the discussed stereo vision based autonomous MAVs that perform on-board image processing employ dense stereo methods. Because of the computational demands of dense stereo algorithms, however, this means that those MAVs are only able to process their stereo recordings at a very low frame rate. Even the sparse stereo based MAV by Shen *et al.* (2013) only provides stereo processing results at a rate of just 1 Hz. Hence, these MAVs require further means for pose inference in order to meet the timing requirements for autonomous control. We on the other hand intend to use stereo vision as the primary source for pose inference. This requires a very fast stereo matching system, which we already discussed in the previous chapter.

4.3 Approach Using One Camera Pair

Before looking into ways of exploiting all four cameras on-board of our MAV, we investigate a solution that makes use of only one camera pair. We extend this solution later to incorporate all available on-board cameras. For this simplified approach we select the forward-facing camera pair, with which we are able to observe a wide section of the scene ahead. Compared to the downward-facing camera pair, the forward-facing cameras are advantageous during take-off and landing. In this case, the close ground proximity would prevent the imagery of the downward-facing cameras to be used for

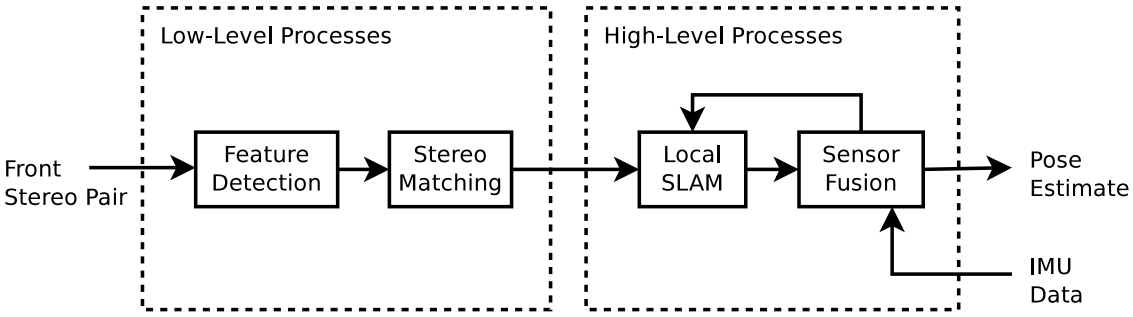


Figure 4.2: System design of the processing method for single-stereo solution.

stereo matching.

The system design of the proposed *single-stereo* solution is shown in Figure 4.2. The necessary processing steps have been categorized into low-level and high-level processes. In our case, low-level process refers to all operations that primarily process image data and have no knowledge about the three-dimensional environment structure or the MAV location. High-level processes, on the other hand, primarily operate on three-dimensional environment features or process the six-dimensional MAV pose. All relevant processing steps are described in detail in the following sections.

4.3.1 Feature Detection and Stereo Matching

Our stereo matching system is based on the efficient feature detector and stereo matching method that we discussed in Chapter 3. As previously suggested, we use a step-width of $w = 2$ for the combined consistency and uniqueness check, which yields a significant processing speed-up. The fact that this stereo method only provides a sparse set of stereo matches is not a limitation in our case, as we use the results for a feature-based localization method that is based on PTAM. For this method, however, we need to perform feature detection for a scale space image pyramid.

As processing time is extremely crucial for our MAV, we create this scale space pyramid without re-running feature detection on each pyramid level. Instead, in each pyramid level we only evaluate those pixel locations for which a feature was detected on the preceding level, as illustrated in Figure 4.3 and described in Algorithm 4.1. This strategy might result in the merging of close-by features from a preceding pyramid level into a single feature on the subsequent pyramid levels.

A feature detected on any pyramid level can be traced back to at least one feature from the primary level. Hence, we only store features from the primary pyramid level and retain the maximum pyramid level l up to which the feature has been detected. This method also has the advantage that the feature locations at higher pyramid levels are still measured with the full image resolution.

The number of features that are detected is crucial for the system performance. In case of too many features, it is impossible to meet our performance requirements. If, on the

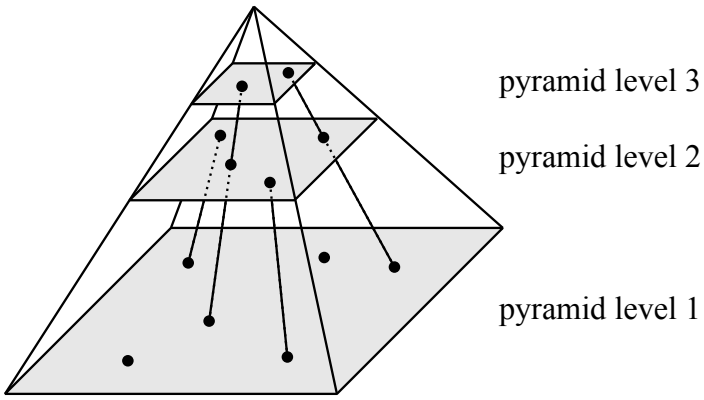


Figure 4.3: Illustration of feature detection in multiple levels of a scale space image pyramid.

Algorithm 4.1: Feature detection using a scale space image pyramid.

```
/* Declare variable for storing detected features.                    */
F := array[pyramidLevels];
/* Detect features for pyramid level 1.                               */
I := inputImage;
F[1] := detectFeatures(I);

/* Detect features in remaining pyramid levels.                       */
for i:=2 to pyramidLevels do
    /* Scale down image for next pyramid level.                       */
    I := scaleDown(I);

    /* Repeat detection for each feature of previous level.           */
    foreach (u_prev, v_prev) ∈ F[i−1] do
        /* Get feature location in current pyramid level.             */
        u_curr := round(pyramidScalingFactor · u_prev);
        v_curr := round(pyramidScalingFactor · v_prev);

        if testFeature(u_curr, v_curr) ∧(u_curr, v_curr) ∉ F[i] then
            /* A new feature was found that does not yet exist in the
               current pyramid level.                                  */
            F[i].append( (u_curr, v_curr) );
        end
    end
end
```

other hand, too few features are detected, then the system will only achieve a degraded accuracy. Although the adaptive threshold of our feature detector reduces this problem, we still receive large numbers of features when encountering highly textured scenes. We solve this predicament by carefully choosing the feature detector adaptivity factor a, and by applying an additional feature reduction step if too many features are detected. The feature reduction only reduces features that originate from the left camera image. Otherwise, different features might be eliminated in both input images, which would degrade the stereo matching performance. We aim at keeping the number of detected left image features below $n_{max} = 1000$.

The primary advantage of our feature detector over other methods is its less clustered feature distribution. Hence, we want to preserve this property when performing the feature reduction. We thus aim at reducing the feature count such that the original feature distribution is retained. For this task, we first determine the feature percentage p that we want to preserve:

$$p = \frac{n_{max}}{n}, \tag{4.1}$$

where n is the total number of detected features.

We then divide the input image into a regular grid of 5×4 rectangular cells. For each cell i, we determine the set S_i of features within the cell's boundaries. We then reduce the number of features in this cell to:

$$m_i = \lfloor \hat{m}_i \rfloor, \text{ with } \hat{m}_i = p \cdot |C_i| + r_{i-1}. \tag{4.2}$$

Here, C is the set of grid cells and r_i is the residual of the rounding operation that is involved in the calculation of m_i. This residual can be determined as follows:

$$r_i = \hat{m}_i - m_i, \text{ with } r_0 = 0. \tag{4.3}$$

By carrying over the residual r_{i-1} from the previous cell $i-1$, we ensure that rounding errors do not lead to too many or too few eliminated features.

When reducing the features of a given cell i, we give preference to features with a large maximum pyramid level l. In case of identical l, preference is given to features with higher feature detection scores. We prefer the pyramid level l over the feature score, as features that are detected on multiple pyramid levels are advantageous for the subsequent processing steps. The final feature reduction algorithm is described in Algorithm 4.2.

4.3.2 Local SLAM

For visual pose estimation, we employ the previously mentioned extension of PTAM that was published by Scherer *et al.* (2012). Whenever PTAM adds a new feature to its map, it requires an initial estimate for the feature's depth. In the original version of PTAM, this depth estimate is obtained by triangulation with the feature's location in

Algorithm 4.2: Grid-based feature reduction.

```
/* Sort features by grid cells.                                    */
cells := array[numVerticalCells][numHorizontalCells];
foreach f ∈ inputFeatures do
    cells[ceil(v/cellHeight)][ceil(u/cellWidth)].append(f);
end

/* Determine percentage of features p to keep.  See Eq. 4.1.       */
p := n_max / |inputFeatures|;
r := 0;
outputFeatures := ∅;

foreach c ∈ cells do
    /* Get number of features to keep in c.  See Eq.  4.2 - 4.3.   */
    m̂ := p · |c| + r;
    m := ⌊m̂⌋;
    r := m̂ − m;

    /* Sort features by preference and copy the first m features.  */
    sortFeaturesByLevelAndScore(c);
    for i := 1 to m do
        outputFeatures.append(c[i]);
    end
end
```

another keyframe from a different viewing position. Scherer *et al.* use an RGB-D camera for their SLAM system, which already provides means for accurate depth perception. Hence, in their system, depth measurements from the RGB-D camera are used as initial depth estimate when adding new features to the map.

Unfortunately, the used RGB-D camera has a limited range and problems with direct or indirect sunlight, depth discontinuities and reflective or highly absorptive surfaces. This means that depth measurements are generally not available for the full camera image. Hence, Scherer *et al.* continue to use the triangulation-based approach for features for which a depth measurement is not available. Furthermore, Scherer *et al.* extended the BA-based map optimization of PTAM, by minimizing the 3D instead of the 2D reprojection error.

We use the approach from Scherer *et al.* for processing the left camera image from our forward-facing camera pair. We bypass PTAM's own feature detection and instead process the successfully matched features, which we receive from stereo matching. Features that have not been matched successfully are not processed any further. This is motivated by the fact that our stereo vision system does not suffer from the same coverage problem as the RGB-D camera, and mismatched features from stereo matching are likely to be of poor quality. Hence, unlike in the approach from Scherer *et al.*, we possess depth

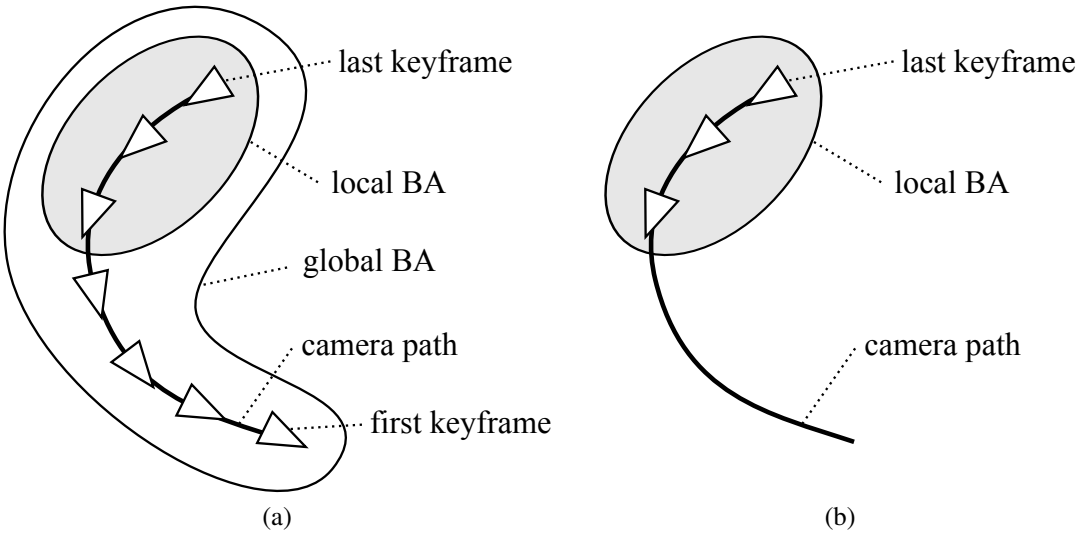


Figure 4.4: Illustration of (a) local and global BA performed by PTAM and (c) local-only BA of our local SLAM method.

measurements for all processed image features.

A general limitation of PTAM is that this method only works well for cases where the camera motion remains within a small volume. Initially, PTAM was proposed for augmented reality applications where the camera is constrained to a small workspace. In the case of large camera movements, PTAM maps a high number of keyframes and features, which quickly degrades the overall run-time performance. Particularly crucial for the performance of PTAM is the BA-based map optimization, which scales poorly with the map size.

As shown in Figure 4.4a, PTAM performs a fast *local* BA optimization of the most recent n keyframe positions and corresponding feature locations. A more time consuming *global* optimization for the entire map is performed at less frequent intervals. If the map grows too large, this global optimization quickly becomes the major performance bottleneck. Due to the missing loop closure detection, an excessive map growth might even happen without large camera displacements. Accumulated errors within the estimated camera pose can prevent the re-detection of previously mapped features. In this case, PTAM keeps creating new keyframes, despite the camera remaining within a constrained volume.

This is a severe limitation, as a quadrotor MAV is capable to quickly cover long distances. At the same time, an autonomous MAV requires frequent and fast updates of its pose estimate, in order to maintain stable flight. To overcome this predicament, we simplify PTAM by avoiding the global BA-based map optimization, as illustrated in Figure 4.4b. Instead, we solely retain PTAM's local optimization, which only requires the most recently mapped n keyframes. At the same time, we erase all keyframes and their

corresponding features that are no longer considered for local map optimization. We thus arrive at a map with a constant size. We prefer the constant run-time performance of this method in favor of a globally optimized map and pose.

As a drawback, the resulting method no longer performs full SLAM, as no global map is preserved. Rather, our approach can be seen as a compromise between SLAM and VO. If the camera movements are constrained to a relatively small volume, our method provides results that are identical to a SLAM system. This is due to the fact that no new keyframes have to be added for as long as the camera remains within the proximity of the previously mapped keyframe locations. If the camera is gradually moved towards one direction, however, the system behavior is similar to VO. In this case, new keyframes are continuously added to the map while old keyframes are dropped. Localization then only happens with respect to a small set of just mapped keyframes. To avoid confusion, we call this method *local SLAM* for the rest of this thesis.

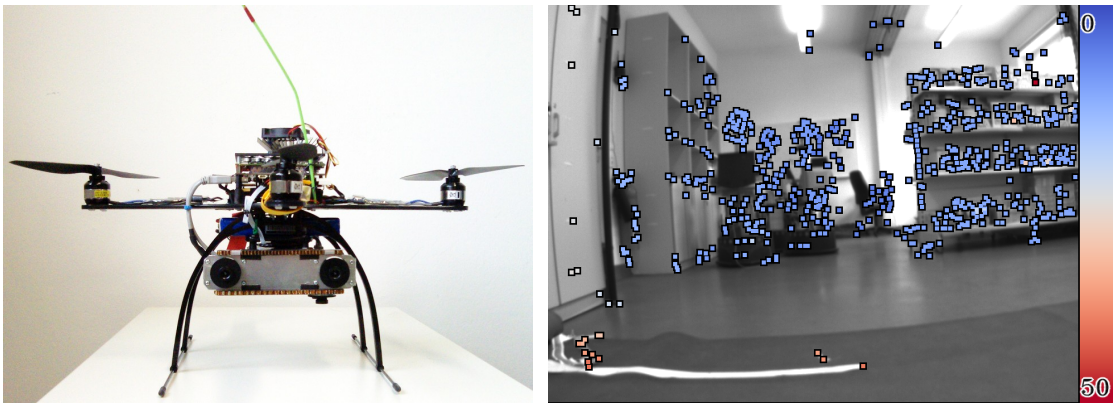
Finally, we perform one further modification of PTAM, which is the replacement of the employed camera model. In its original version, PTAM uses the arctangent-based camera model that was published by Devernay and Faugeras (2001). We replace this model with the more widely used Brown's distortion model, which we discussed in Section 2.2.2 on page 17. This is the same model that we use for calibrating our stereo system, which simplifies the camera calibration process. For the BA-based map optimization, however, PTAM requires the camera model's inverse. Unfortunately, Browns's distortion model is not invertible. We solve this problem by numerically pre-computing a transformation table for the inverse camera model. This transformation table is populated by sampling the original camera model at subpixel accuracy.

4.3.3 Sensor Fusion and Motion Feedback

We fuse the pose estimate from local SLAM with measurements received from the IMU, for which we employ an EKF. A suitable EKF implementation has been published by Klose (2011). This filter uses the measurement data from the IMU to perform the Kalman prediction step. To every pose estimate delivered by our local SLAM system, the filter then applies the Kalman correction. Similar methods have long been used for fusing GPS location data with inertial measurements (see Gross *et al.*, 2012).

The fused pose is passed on to the microprocessor board, which runs the low-level control software that we previously discussed in Section 2.1.3 on page 11. As shown in Figure 4.2 on page 64, the filtered pose is also fed-back to our local SLAM method. This feedback was introduced in order to improve PTAM's motion prediction, which is necessary in order to obtain an estimate for the expected position of map features in the next camera frame. Because the search for matching image features is constrained to a local neighborhood of the estimated feature locations, an accurate motion prediction is important.

By default, PTAM uses a motion model that assumes a linear motion along the camera's optical axis with a decaying velocity. At the same time, the motion model predicts



(a) (b)

Figure 4.5: (a) Prototype MAV used for flight experiments and (b) example for on-board stereo matching results during autonomous flight.

the camera rotation by aligning a small sub-sampled version of the current camera frame to the previous frame. This happens by applying the Efficient Second order Minimization (ESM) algorithm that was proposed by Benhimane and Malis (2004). In this case, the ESM algorithm approximates the in-plane rotation and translation that are necessary for aligning both camera frames. These 2D transformations are then converted into a corresponding 3D camera rotation using an iterative algorithm.

We keep this image alignment based rotation estimation for our new motion model, as this method has proven to deliver robust results. However, we alter the linear motion estimation to rely on the more accurate pose estimate that is fed-back from the EKF. Hence, instead of predicting the camera movement from PTAM's two previous pose estimates, we instead use the two previous estimates received from our EKF.

4.3.4 Evaluation

With the presented system, a preliminary experiment was performed in which the MAV was programmed to hover at a low altitude in an indoor environment. Hovering was achieved by feeding a constant position to the position control algorithm, which runs on the low-level microprocessor board. In this experiment, take-off and landing was performed manually using a remote control. The MAV that was used for this experiment was an early prototype of the MAV platform that we presented in Section 2.1.3 on page 11, which is displayed in Figure 4.5a.

Several autonomous hovering flights were performed successfully, of which we examine one flight in this section. For all evaluations, we only consider the time span where the MAV was hovering autonomously. The total time between take-off and landing in the considered test run was 45.7 s. We neglect the first 8.6 s and last 3.1 s for take-off and

Table 4.1: CPU usage during autonomous hovering flight.

Process	**CPU Usage**
Stereo Matching	37.4%
Local SLAM	32.5%
Image Acquisition	6.7%
Data Recording	5.8%
Sensor Fusion	1.1%
Other	2.5%
Total	**85.6%**

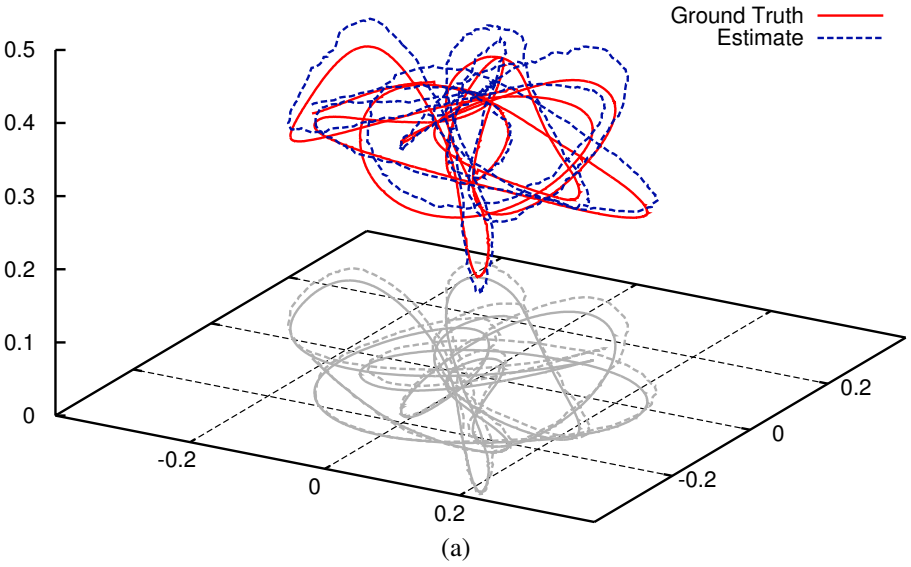
landing, which leaves us a flight time of 33.9 s to analyze.

For evaluation purposes, all sensor data and the outcome of the on-board pose estimation were recorded during the test flight. An example for the recordings of the left on-board camera with overlaid stereo matching results can be seen in Figure 4.5b. Furthermore, ground truth motion information was recorded using an Optitrack tracking system, which relies on a set of highly reflective markers that are attached to the MAV. To make the ground truth and our on-board pose estimates comparable, the trajectories for both have been aligned. For this task, an iterative error minimization was performed for each position coordinate and the yaw rotation, for the first 2 s after the start of autonomous hovering. By aligning only the beginning of both trajectories, we ensure that drift errors are not ignored in our evaluation. The aligned trajectories are shown in Figure 4.6a in a perspective view, and in Figure 4.6b in a top-down view.

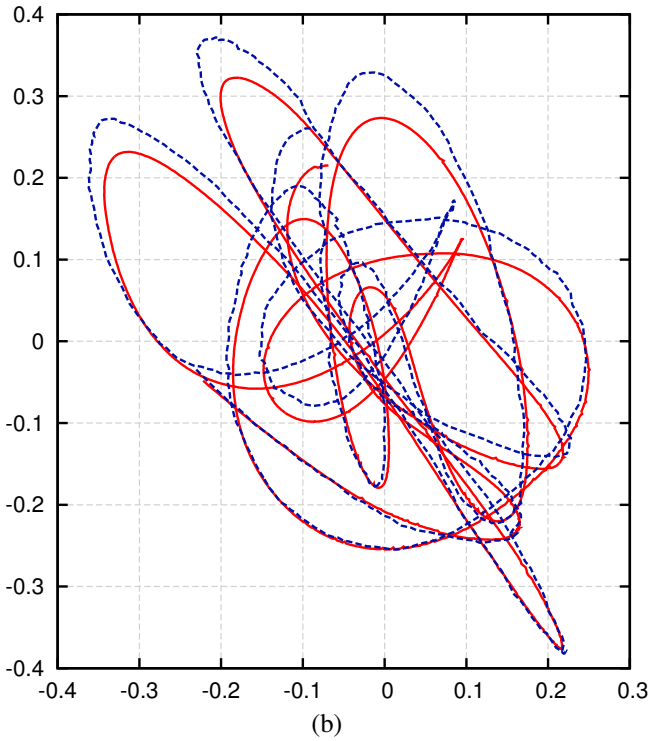
In total, the cameras recorded 1019 frames each and our processing pipeline was able to generate 29.3 pose estimates per second on average. This is very close to the camera frame rate of 30 Hz, which indicates that only very few frames have been dropped. The average number of detected features in the left camera image was 999.5, of which 64% were successfully matched by our stereo matching method. Furthermore, CPU load statistics were recorded, which are listed in Table 4.1. The table reveals that if data recording had been omitted, then the CPU load would have been below 80%.

To analyze the performance of our autonomous MAV, we examine how well it can keep its location during autonomous hovering. As reference hovering position, we consider the average position of the MAV during the evaluated time span. We compare each position sample of the recorded ground truth against this reference position and calculate the position error. The average position error that we receive with this method is 0.26 m, and the Root Mean Square Error (RMSE) is 0.32 m. Although these error margins leave room for improvements, the autonomous flight should already be accurate enough for a set of indoor applications that do not require very precise position control.

We can expect that the used PID position controller is responsible for a fair share of the observed position error. Hence, the more interesting question at hand is the accuracy of the on-board pose estimation. We can measure this error by calculating the Euclidean



(a)



(b)

Figure 4.6: (a) Perspective and (b) top-view of the ground truth motion information and on-board motion estimates. The scale of both diagrams is in meters.

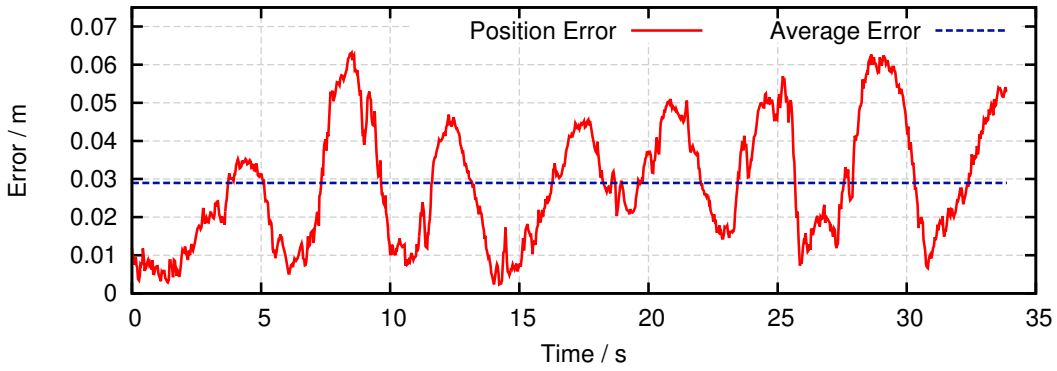


Figure 4.7: Deviation of on-board position estimate from ground truth trajectory.

distance between the on-board position estimates and the aligned ground truth trajectory, which was done for Figure 4.7. This figure reveals that even though we have eliminated the global optimization that used to be present in PTAM, the error stays bounded throughout the hovering experiment. In fact, the average error is only 2.89 cm, and the RMSE has a value of 3.31 cm, which is an order of magnitude less than the previously observed position error.

The discussed experiment provides a first impression on the performance of our MAV system. Although these results seem promising, more work needs to be done for improving the accuracy of the autonomous flight. In addition, more advanced autonomous flight maneuvers need to be implemented, including autonomous take-off and landing. In the following sections, the presented MAV design is developed further to address these demands. A more thorough evaluation of our autonomous MAV is provided in Section 4.4.7 on page 85, after our final system design is introduced.

4.4 Approach Using Two Camera Pairs

Having a forward-facing camera pair ensures a large field of view during low-altitude flights. Furthermore, it can facilitate the detection of obstacles that lie ahead in flying direction. Unfortunately, however, forward-facing cameras do not perform well when encountering fast yaw rotations. In this case, the cameras register fast image movements and might be subject to motion blur, which both impede current visual navigation methods. An alternative are downward-facing cameras, which in the case of stereo vision also allow an observation of the ground distance and the relative orientation of the MAV towards the dominant ground plane. While a downward-facing camera is better at observing fast yaw rotations, it exhibits similar problems when performing ground proximity flights. Furthermore, in case of a downward-facing stereo camera, stereo matching is only possible once the MAV has reached a minimum altitude.

Those dissimilar strengths of downward- and forward-facing cameras lead us to believe that they complement each other when used in a combined setting. Hence, in this

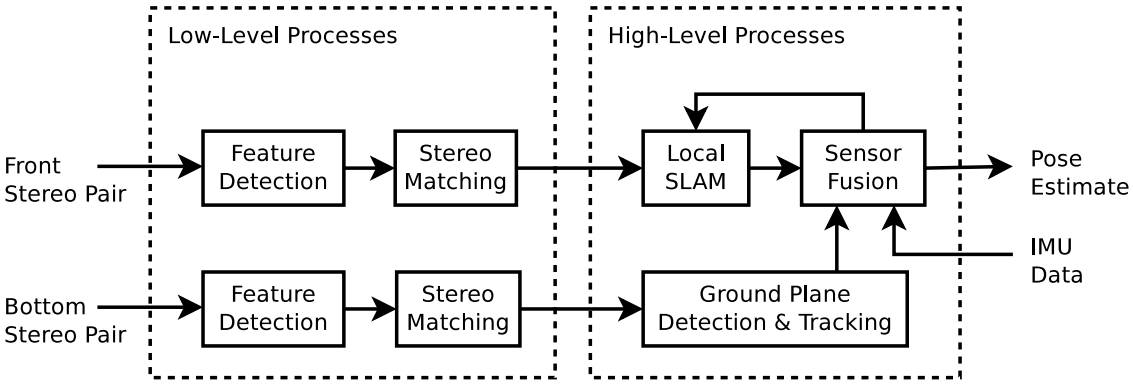


Figure 4.8: System design of the processing method for both stereo camera pairs.

section we present a solution for simultaneously processing the imagery of all four cameras on-board of our MAV platform. However, as we have just seen, the CPU load of our MAV is already at a critical level when just using the forward-facing camera pair for autonomous navigation. Hence, it is not possible to run a second instance of our local SLAM system in parallel for processing the imagery of the downward-facing cameras.

We achieve our goal by first optimizing the run-time performance of our already discussed single-stereo solution, and by introducing a new efficient processing method for the downward-facing camera pair. This method allows us to analyze the imagery of all four cameras on-board our MAV in real-time. The system design of this *double-stereo* solution is shown in Figure 4.8. Again, we have categorized the necessary processing steps into low-level and high-level tasks. In the following, we discuss the details of this system design and the motivations that have lead to the important design decisions.

4.4.1 Problems of Single-Stereo Solution

One key issue with the previously discussed single-stereo solution is the potential drift of the estimated position and orientation. If the orientation estimated by local SLAM is used for controlling the quadrotor MAV, any errors in the estimated pitch and roll angles will have a disrupting impact on the flight stability. In the previous experiments, the MAV was controlled with the original PIXHAWK flight controller, which only relies on IMU measurements for determining the current attitude. It would be preferable to instead use the more accurate vision estimated orientation for this task. However, this would make the handling of orientation errors even more important.

Not only orientation drift can be problematic, but also drift errors for the estimated position are an issue. If the MAV is programmed to fly on a preset trajectory, a position deviation would cause the MAV to leave the desired track, which could lead to potentially dangerous situations. But even if the MAV performs on-board path planning in consideration of the perceived environment, position drift can still cause troubles. For example, the MAV presented by Fraundorfer *et al.* (2012) performs autonomous on-board path

planning, but this happens only in two dimensions at a fixed flying altitude. If such an MAV would not have other means for perceiving the current altitude, it would not be able to react on position drifts in the vertical direction.

Our local SLAM method has further difficulties with yaw rotations. This was more severe for the original PTAM version, which just processes imagery of a monocular camera. Because no triangulation can be performed for the features that are observed for a rotation-only movement, PTAM is not able to obtain reliable depth measurements in this case. Our local SLAM method obtains its depth information from stereo vision, which should make yaw rotations less problematic. However, fast yaw rotations still lead to large image movements. This can cause the majority of the visible map features to quickly move out of sight, which will result in tracking failure.

Finally, if tracking ever fails, recovery can only occur if the camera still depicts a scene that has been well observed by at least one existing keyframe. Since the MAV is likely to be on an onward flying trajectory, we cannot expect that this is the case. But even if the camera hasn't moved much since the previous keyframe, recovery might still fail, which can lead to a random new position. Thus, recovery needs to be improved if we want to achieve robust autonomous flight.

The problems we have described so far can be solved or at least be reduced, if we employ the downward-facing camera pair in addition to the already used forward-facing cameras. How exactly this can be achieved is discussed in the following sections.

4.4.2 Performance Improvements

We keep our local SLAM system for processing the imagery of the forward-facing cameras. To be able to process the data of all four cameras in real time, we hence need to improve the run-time performance of this method. The simplest way to improve the processing performance is to employ a less demanding parameterization. We therefore set the maximum number of allowed features to $n_{max} = 800$, which is less than the previous limit of 1000. However, this only provides us with a marginal speed-up.

To receive higher speed-ups, several code-level optimizations have been performed. Most importantly, a performance problem was identified in PTAM's original BA-based map optimization. This problem occurs if the map optimization finishes before a new key frame has been added. In this case, the map optimization is re-run on the already optimized map, which causes the mapping thread to always have a high CPU load. For the original PTAM system this is only a minor problem, as the global map optimization quickly becomes the major performance bottleneck when the map grows sufficiently large. Furthermore, PTAM only has two computationally expensive threads, and is recommended to be run on a computer with at least two CPU cores. Hence, if the mapping thread causes a high CPU load in one core, the tracking thread that is run on the other CPU core should not be affected.

We, on the other hand, only perform local map optimization, which reduces the performance impact of the map optimization task. Furthermore, we are running more than

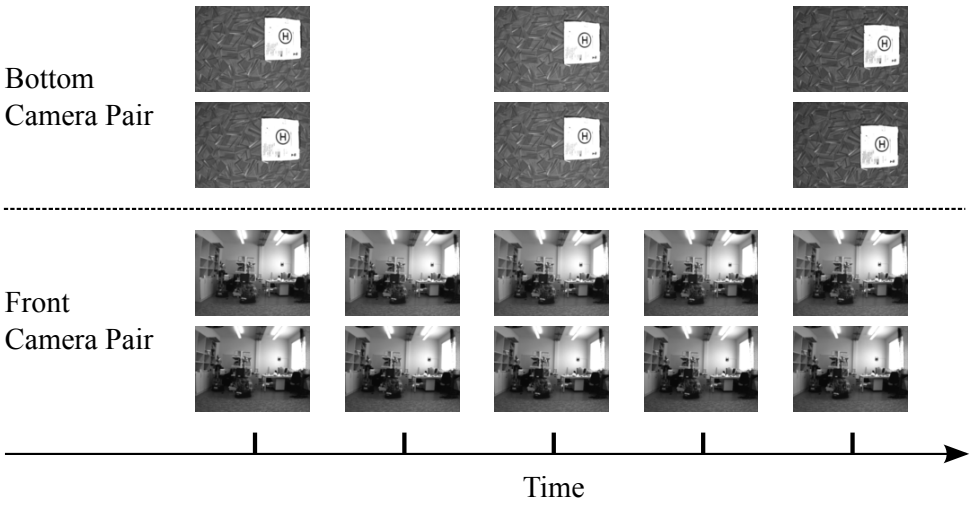


Figure 4.9: Camera synchronization using different frame rates for the forward- and downward-facing camera pairs.

two computationally expensive threads on a dual core CPU, which makes it important that every thread only consumes as much CPU time as absolutely necessary. Hence, by enforcing that the map optimization is not executed more frequently than new keyframes are added, we can greatly improve the processing performance.

Further optimizations were performed in order to improve the overall run-time performance. In particular, image acquisition, stereo matching and local SLAM have been integrated into a single process, by using the Nodelet concept that is offered by ROS. This allows us to pass image data between these system components without the need for memory copying or message serialization. Further smaller code-level optimizations were performed in the various system components.

4.4.3 Processing Method for the Downward-Facing Cameras

The processing method for the data recorded by the downward-facing cameras differs fundamentally from the method that we use for the forward-facing ones. At the beginning, however, there is again stereo matching, for which we employ our efficient sparse stereo matching method. This time however, we set the maximum feature count to 300, as we require fewer features in this case. Furthermore, the downward-facing cameras are operated at only 15 Hz, which is half of the frame rate that we use for the forward-facing cameras. This means that special care has to be taken for camera synchronization.

In our case, the downward-facing cameras are synchronized to every other frame of the forward-facing ones, as shown in Figure 4.9. Hence, every other frame we receive an image from all four cameras, while only imagery from the forward-facing cameras is received for the frames in between. Operating the downward-facing cameras at half the frame rate significantly reduces the computation costs that we require for image pro-

cessing. We are able to use this lower frame rate, as our processing method for the downward-facing cameras is less sensitive to the image rate than our local SLAM system.

Dominant Ground Plane Detection

We can design an optimized processing method for the downward-facing cameras, if we assume that the ground is flat and level. This is usually a valid assumption for man-made indoor environments with even floors. Unfortunately, this assumption does not hold for natural outdoor environments. Unless our MAV encounters very rough or steep terrain, however, a flat and level ground assumption can be a sufficiently accurate approximation.

According to this assumption, all 3D points received from stereo matching are expected to lie in the same geometric plane. If we know the equation for this *ground plane* in the form of

$$ax + by + cz + d = 0, \tag{4.4}$$

then we can extract the height h, pitch angle Θ and roll angle Φ with the following equations[1]:

$$h = \frac{-d}{b}, \tag{4.5}$$

$$\Theta = \tan^{-1}\left(\frac{-c}{b}\right), \tag{4.6}$$

$$\Phi = \tan^{-1}\left(\frac{a}{b}\right). \tag{4.7}$$

Unlike the pose estimates of our local SLAM system, these measurements are absolute. Hence, they are not prone to drift or erroneous offsets, which is why we expect those measurements to increase the overall accuracy in a combined system.

We obtain an estimate for the equation of the dominant ground plane by using a RANSAC-based plane estimator, for which we use the implementation provided in the Point Cloud Library (PCL) (see Rusu and Cousins, 2011). Before running this RANSAC algorithm, however, we need to decide for an outlier threshold t_o. This threshold indicates the maximum distance towards the plane model that is allowed for a point, which is still classified as being an inlier. Since the depth measurement error of a stereo vision system increases quadratically with the measured depth (see Point Grey Research, Inc., 2012), a constant threshold does not seem to be a viable solution.

Instead, the parameterization of t_o should depend on the actual camera height h. Thus, we precompute a robust initial height estimate $\hat{h}$, before performing the actual RANSAC plane fitting. As estimator we use the median depth of the 3D points obtained from stereo

[1] These equations differ from the initial publication of the presented autonomous MAV (Schauwecker and Zell, 2013, 2014a). This is due to the fact that the downward-facing cameras used to be rotated by 90° in a previous camera mount.

matching. Using this initial estimate, we determine the outlier threshold as follows:

$$t_o = \hat{h}^2 \cdot t_r, \tag{4.8}$$

where t_r is a configurable threshold that is relative to the square of the estimated camera height $\hat{h}^2$.

Next, we need to estimate the variances for our measurements of h, Θ and Φ, for which we use a sampling-based approach. For the height variance σ_h^2, we calculate the distance between the plane model and each point that was selected as inlier by the RANSAC method. The variance of the sample mean is then our estimate for σ_h^2:

$$\sigma_h^2 = \frac{1}{|S_i| \cdot (|S_i| - 1)} \sum_{p \in S_i} \left(\Delta_g(p) - \overline{\Delta}_g\right)^2, \tag{4.9}$$

where S_i is the set of selected inlier points, $\Delta_g(p)$ is a function that provides the distance between a point p and the ground plane model, and $\overline{\Delta}_g$ is the average ground plane distance of all inlier points.

For the angular variances σ_Θ^2 and σ_Φ^2, we first group the inlier points into sets of three. These triplets are chosen stochastically such that their points have a large distance to one another. With those triplets we then calculate samples for Θ and Φ, and use the variance of the sample mean as our estimate for σ_Θ^2 and σ_Φ^2:

$$\sigma_\Theta^2 = \frac{1}{|S_t| \cdot (|S_t| - 1)} \sum_{(p,q,r) \in S_t} \left(\alpha_\Theta(p,q,r) - \overline{\Theta}\right)^2, \tag{4.10}$$

$$\sigma_\Phi^2 = \frac{1}{|S_t| \cdot (|S_t| - 1)} \sum_{(p,q,r) \in S_t} \left(\alpha_\Phi(p,q,r) - \overline{\Phi}\right)^2, \tag{4.11}$$

where S_t is the set of point triplets, and the functions $\alpha_\Theta(p,q,r)$ and $\alpha_\Phi(p,q,r)$ determine the pitch and roll angles of the plane that passes through the points p, q and r (analogous to Equations 4.6 and 4.7). The variables $\overline{\Theta}$ and $\overline{\Phi}$ represent the mean pitch and roll angles of the sample set.

Finally, we apply a simple outlier rejection that is based on the previously detected plane model. A plane is classified as outlier if its height, roll or pitch differ from the previous plane model by more than a preset threshold. An example for the performance of the final method on-board of our MAV can be seen in Figure 4.10. Here, the plane has been projected back into the unrectified camera image, which leads to the visible radial distortion. The red points in this figure indicate features that were selected as inliers by the RANSAC plane estimator, while yellow points were classified as outliers.

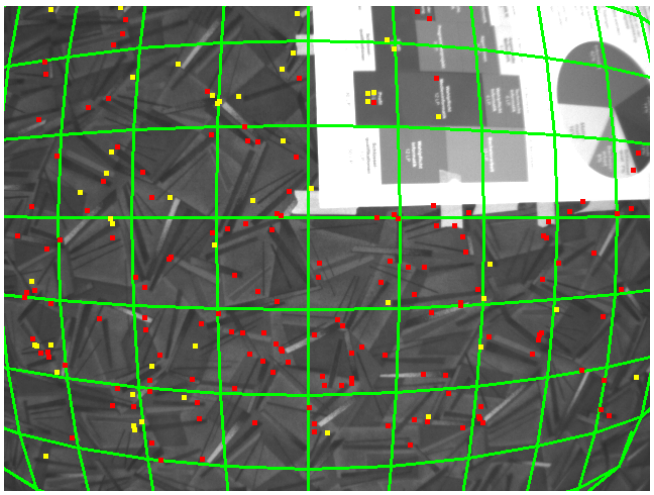


Figure 4.10: Example for a detected ground plane that is projected into the unrectified camera image.

Frame-to-Frame Tracking

While the detected ground plane provides us with measurements for height, roll and pitch, we do not gain information on horizontal translation and yaw rotation. For measuring these quantities, we need to employ a different method. In our case, we use an approach based on frame-to-frame tracking. Because we assume a flat ground, horizontal displacements and yaw rotations should result in an affine image transformation, which consists of a 2d-translation and an in-plane rotation. We hence attempt to find the transformation that aligns a previously captured frame to the current camera frame.

For this task we chose the previously mentioned ESM-algorithm from Benhimane and Malis (2004), which uses a homography (see Hartley and Zisserman, 2003) for image alignment. This happens by iteratively applying transformations to an initial homography, until the algorithm converges at a steady solution. In our case, we limit ourselves to a homography that only consists of translations and in-plane rotations, which means that we neglect perspective effects. Even though ESM is an efficient method, finding the transformation between two full-resolution camera frames is very time consuming. Hence, we perform this step with two very low-resolution sub-sampled frames. In fact, we use a resolution of just 80×60 pixels. This number might seem to be small, but because ESM works well at the sub-pixel level, we still receive sufficiently accurate results.

As mentioned in Section 2.1.3 on page 11, we use lenses with relatively small focal lengths for the downward-facing cameras. This provides us with a large field of view, but also causes strong radial distortions that disrupt the frame-to-frame tracking. Hence, we first perform image rectification, which can be combined with the required sub-sampling into one single image transformation. This transformation is much faster than an indi-

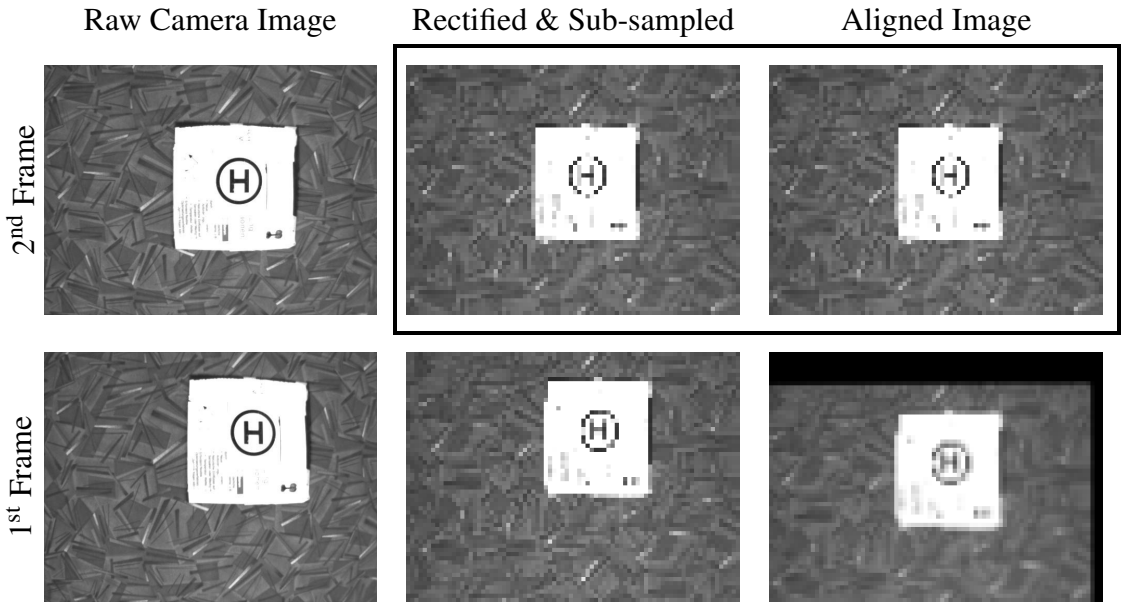


Figure 4.11: Illustration of image processing steps for frame-to-frame tracking.

vidual rectification at full image resolution with subsequent image sub-sampling, and it also avoids unnecessary blur.

The effect of the combined rectification and sub-sampling transformation can be seen in the second column of Figure 4.11. In the third column of this figure we see the result of the ESM-based image alignment. Here, the rectified lower image has been aligned to the rectified upper image. Between both images, the camera performed an upward-right movement. The image alignment thus caused a bottom-left shift of the second input image, which introduced a border with unknown pixel intensities at the upper and right edge.

We can extract our desired measurements from the homography found by ESM. The fourth column of the homography matrix represents the translation vector in homogeneous coordinates $\begin{pmatrix}\Delta_u & \Delta_v & w\end{pmatrix}^T$. Using the height h that we received from the detected ground plane and the camera's known focal length f, we can convert the translation vector from pixel to world coordinates:

$$\begin{pmatrix}\Delta_x \\ \Delta_y\end{pmatrix} = \frac{h}{wf}\begin{pmatrix}\Delta_u \\ \Delta_v\end{pmatrix}. \tag{4.12}$$

The yaw rotation $\Delta\Psi$ cannot be extracted as easily without applying a homography decomposition. Thus, we apply the simple approach of transforming a distant point (u_p, v_p) with the found homography $\mathbf{H}$. We then measure the angle towards the point's initial location:

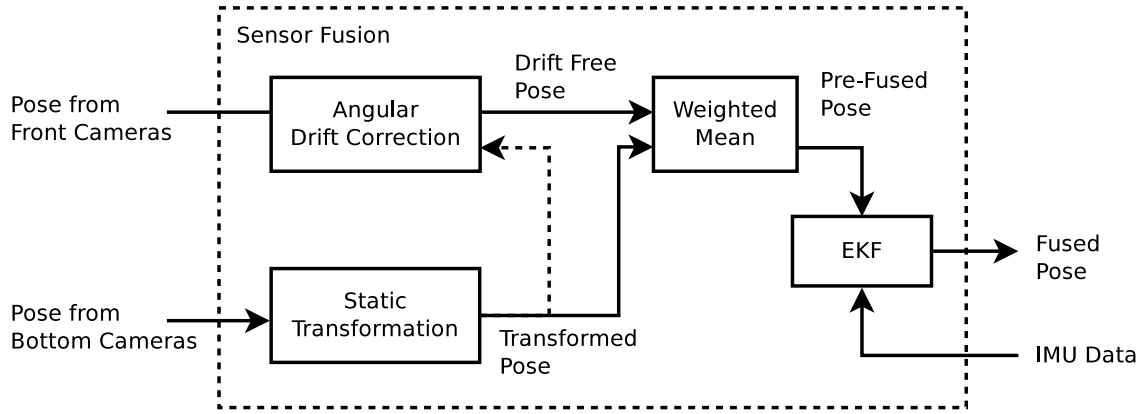


Figure 4.12: Schematics of sensor fusion with forward- and downward-facing cameras.

$$\Delta\Psi = \text{atan2}\left(\frac{v_q}{w_q}, \frac{u_q}{w_q}\right), \text{ with } \begin{pmatrix} u_q \\ v_q \\ w_q \end{pmatrix} = \mathbf{H} \cdot \begin{pmatrix} u_p \\ v_p \\ 1 \end{pmatrix}, \tag{4.13}$$

where $\text{atan2}(u, v)$ is the two-argument version of the arctangent function that respects the quadrant of the computed angle. This function is commonly found in many programming languages.

The variances of Δ_x, Δ_y and $\Delta\Psi$ are assumed to be constant and configured manually. With the measurements for translation and yaw rotation and the measurements extracted from the ground plane equation, we obtain a full six Degrees of Freedom (DoF) estimate for the current MAV pose.

4.4.4 Sensor Fusion and Control

With our local SLAM method and the described plane detection and frame-to-frame tracking, we receive two independent estimates for the current MAV pose. These two estimates need to be unified into one single pose estimate, which happens during sensor fusion. For this task, we can complement the sensor fusion that we previously discussed in Section 4.3.3 on page 69. The schematics of our extended sensor fusion are shown in Figure 4.12. This figure is a breakdown of the 'Sensor Fusion' block that was shown in Figure 4.8 on page 74. For now, we ignore the block labeled 'Angular Drift Correction', which is discussed in detail later on.

Due to camera synchronization, pose measurements are from exactly the same point of time, if received for both camera pairs. We use the pose of the forward-facing cameras as reference pose for sensor fusion. Hence, we transform the pose of the downward-facing cameras, such that it matches the expected pose of the forward-facing cameras for the assumed MAV position and orientation. We do this by applying a static transformation that we determine manually. Because we do not attempt to track the same features in both

camera pairs, a very accurate calibration of this static transformation is not necessary.

We fuse the pose estimates from both camera pairs before applying the EKF. This happens by independently calculating the weighted mean for each element of the pose vector, while using the inverse variance as weights. For the x coordinate, the weighted mean $\overline{x}$ and new variance $\overline{\sigma}_x^2$ can be computed as follows:

$$\overline{x} = \frac{\frac{x_1}{\sigma_{x1}^2} + \frac{x_2}{\sigma_{x2}^2}}{\frac{1}{\sigma_{x1}^2} + \frac{1}{\sigma_{x1}^2}}, \tag{4.14}$$

$$\overline{\sigma}_x^2 = \frac{1}{\frac{1}{\sigma_{x1}^2} + \frac{1}{\sigma_{x1}^2}}, \tag{4.15}$$

where x_1 and x_2 are the coordinates from the forward- and downward-facing camera pair, and σ_{x1}^2 and σ_{x2}^2 are the corresponding variances. If the measurement error for both pose estimates is normally distributed, then the weighted mean should provide us with a maximum-likelihood estimate for the current pose (see Hartung *et al.*, 2011).

The reason why we fuse both poses before applying the EKF is that we want to avoid giving preference to either of them. If both pose estimates would be processed individually by our EKF, then that pose which is processed last would have a more significant influence on the filter outcome. This is usually the pose obtained with the forward-facing cameras, as it requires more time to be computed.

If only one pose is available due to the bottom cameras skipping one frame, or because one method fails to deliver a reliable pose estimate, then the weighted mean is avoided and the single pose estimate is processed by the EKF as is. In any case, the fused pose is passed on to the low-level flight controller. Unlike in the single-stereo solution from Section 4.3, the flight controller has been modified in order to use the roll and pitch angle from our pose estimate as an estimate for the MAV's attitude.

The original PIXHAWK flight controller derives its attitude estimates solely from IMU measurements. Hence, whenever the MAV experiences significant accelerations, this estimate becomes inaccurate. Our vision-based attitude estimation provides a higher accuracy, but unfortunately also has a higher latency. We overcome the latency problem by still deriving the roll and pitch rate from low-latency IMU measurements. Hence, the MAV is still able to promptly sense attitude changes and react appropriately.

4.4.5 Drift Correction

Preliminary experiments with the method presented so far have shown that there are still problems with flight stability. This can mostly be credited to accumulated errors that lead to unwanted drift. Two such error sources have been identified and can be compensated by using additional processing steps.

Map Drift

One major source of error is the map generated by our local SLAM system. Once a keyframe has been added to the map, its position is only altered by the BA-based map optimization, which generally only performs small corrective changes. This means that if a keyframe has been created with an incorrect or inaccurate pose, this error might not be corrected before the keyframe is discarded. Hence, all pose estimates that are obtained by matching against this keyframe will also be inaccurate. Consequently, this also affects the pose of the subsequently added keyframes. Errors hence tend to propagate in the local SLAM map, which is why the map is subject to drift.

The downward-facing cameras deliver us absolute measurements for height, roll and pitch. With those absolute measurements, we should be able to perform at least a partial correction of the local SLAM map. The fused position, which contains contributions from those absolute measurements, is already fed back to the local SLAM system (see Figure 4.8 on page 74). However, so far the fused pose is only used for motion prediction, which does not have an influence on the existing map.

It is thus necessary to correct the pose of existing keyframes. We perform this correction by applying a global transformation to the entire map. This transformation is chosen such that it compensates the difference between the last pose estimated by the local SLAM system and the final pose estimate after sensor fusion. If $\mathbf{T}_s$ is the transformation matrix for the pose estimated by our local SLAM system, and $\mathbf{T}_f$ is the transformation matrix after sensor fusion, then the matrix product ${\mathbf{T}_f}^{-1} \cdot \mathbf{T}_s$ represents the transformation that we required to map $\mathbf{T}_s$ to $\mathbf{T}_f$. We hence define our corrective transformation $\mathbf{T}_c$ as follows:

$$\mathbf{T}_c = \lambda({\mathbf{T}_f}^{-1} \cdot \mathbf{T}_s). \tag{4.16}$$

Here, the transformation is scaled with the weighting factor λ. This weight is set to be a small value (we use a value of 0.05), such that only small corrective changes are performed. Drift errors should thus be gradually compensated over several frames. Furthermore, we force the horizontal displacement of the corrective transformation to be 0. Because there is no sensor that delivers absolute measurements of the horizontal position, we prefer to keep the position estimated by local SLAM instead.

Angular Drift

Although the previously described drift correction works well for correcting the height of a keyframe, its performance is generally poor for roll and pitch errors. This can be explained if we have a look at the variances that are used during sensor fusion. While the height measurements received from the downward-facing cameras are more accurate than the measurements received from local SLAM, the variances for the measured roll and pitch angles are several orders of magnitude larger. This means that roll and pitch measurements from the downward-facing cameras are mostly ignored during sensor fusion.

Unlike local SLAM, however, the downward-facing cameras provide an absolute measurement, which is why we do not want do disregard this information. We solve this problem by introducing an additional processing step during sensor fusion, which was labeled 'Angular Drift Correction' in Figure 4.12. In this step, we try to estimate the angular drift of the local SLAM pose and correct it before sensor fusion starts. Because the angular measurements from the downward-facing cameras are considerably noisy, we employ an additional Kalman filter for this task. This Kalman filter tracks the difference between the orientation estimate gained from local SLAM and the estimate from our downward-facing cameras. We represent the orientation as quaternions, which matches the representation used in the entire sensor fusion pipeline.

If we are able to correct the angular drift, then the pose received after sensor fusion should contain the correct orientation. We know that the fused pose is fed back to local SLAM, where it is used to correct the map drift with respect to the weight λ. Hence, we can expect that the angular drift is reduced in the next frame. This knowledge can be incorporated into the model of our Kalman filter. We assume that the arithmetic difference between the two orientation quaternions Δ_q reduces to $\Delta_q \cdot (1-\lambda)$ from one frame to another. If we ignore all other influences on the orientation drift, then we arrive at the following state transition matrix for our Kalman filter:

$$\mathbf{F_k} = \begin{pmatrix} 1-\lambda & 0 & 0 & 0 \\ 0 & 1-\lambda & 0 & 0 \\ 0 & 0 & 1-\lambda & 0 \\ 0 & 0 & 0 & 1-\lambda \end{pmatrix}. \tag{4.17}$$

The filtered quaternion difference is then added to the orientation quaternion from local SLAM, which effectively removes orientation drift. However, we further adapt the final pose estimate by restoring the yaw rotation to its uncorrected value, as there are no absolute measurements for the yaw angle.

4.4.6 Recovery

The last remaining problem that needs to be solved is recovery of the local SLAM system in case of tracking failure. As mentioned previously, the recovery approach employed by PTAM does not work well for our application. Hence, we use a different technique that makes use of the redundant information available from both camera pairs. Even when the local SLAM method fails, we still receive a full 6-DoF pose estimate from the downward-facing cameras. Thus, the pose of the MAV is still known but with a degraded accuracy. Nevertheless, we should be able to maintain control of the MAV until local SLAM has recovered.

In case of tracking failures, we thus force the local SLAM system's current pose to match our final pose estimate after sensor fusion. In this case, the fused pose is only

Figure 4.13: Autonomous flight using all four on-board cameras.

obtained through measurements of the downward-facing cameras and the IMU. This pose is unlikely to match the current map of the local SLAM system, which prevents the system from recovering by itself. Hence, we discard the entire map and begin mapping from scratch. We start by mapping the current frame at the currently available fused pose. Thus, the system should quickly recover once the cause for the tracking failure has disappeared. Usually, tracking failures result from quick camera movements. Hence, once the MAV has stabilized itself by using the less error prone pose estimates from the downward-facing cameras, local SLAM should continue functioning up to expectations.

4.4.7 Evaluation

Several experiments were conducted to evaluate the quality of the proposed MAV design. All flying experiments took place in the same indoor lab environment that we previously used in Section 4.3.4 on page 70 to evaluate the single-stereo solution. This time, however, the floor has been covered with a texture rich carpet in order to provide sufficient features for the downward-facing cameras. An example for the scene that is observed by the downward-facing cameras was previously shown in Figure 4.10 on page 79. A picture of our MAV during an autonomous flight, while using all four on-board cameras, can be seen in Figure 4.13. Accurate ground truth motion information was again recorded for all test flights using an optical tracking system. Furthermore, the sensor data (*i.e.* camera imagery and IMU measurements) were recorded for all test-flights, which allows us to re-process all test-runs offline.

Table 4.2: Position estimation errors for hovering flight.

Method	Average Error	RMSE
On-Board Estimate	1.60 cm	1.76 cm
Local SLAM Only	2.23 cm	2.30 cm
Ground Plane Only	24.4 cm	32.7 cm

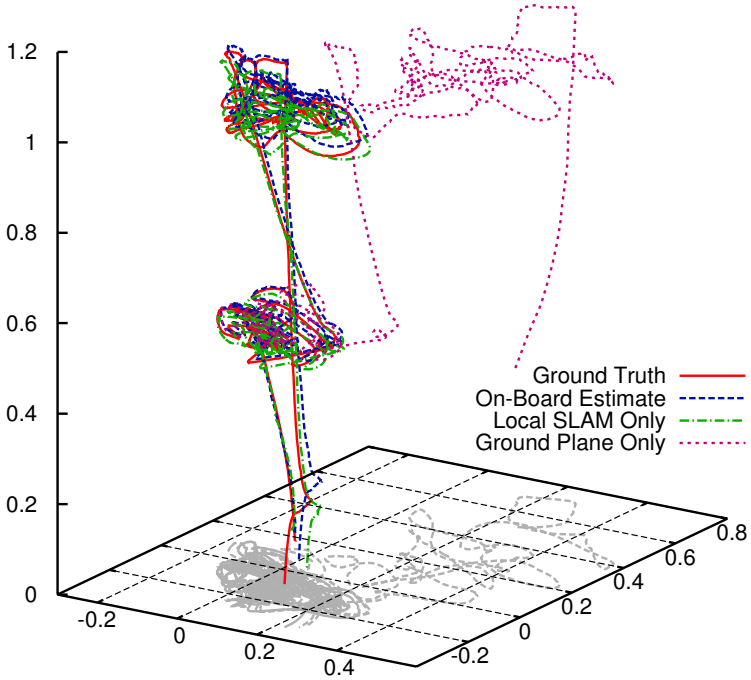
Hovering

In the first flight experiment, the MAV's ability to hover at a preset location has been examined. The MAV was programmed to hover at a height of about 0.5 m for one minute, and then ascend to a height of 1.0 m, where it hovers for one further minute. Unlike in the previous single-stereo experiment from Section 4.3.4, take-off and landing were performed fully autonomously. A perspective view of the recorded ground truth position and the position estimate obtained by the on-board software are shown in Figure 4.14a, while a top-down view can be seen in Figure 4.14b. Both figures contain two more curves, which are the position estimates received when offline re-processing the recorded sensor data with only the local SLAM system or only our ground-plane-based pose estimation. By plotting these offline results, we can compare how the MAV would have behaved, if it had been equipped with only two cameras. All plotted tracks have been aligned such that their position and orientation closely match for the first two seconds after take-off.

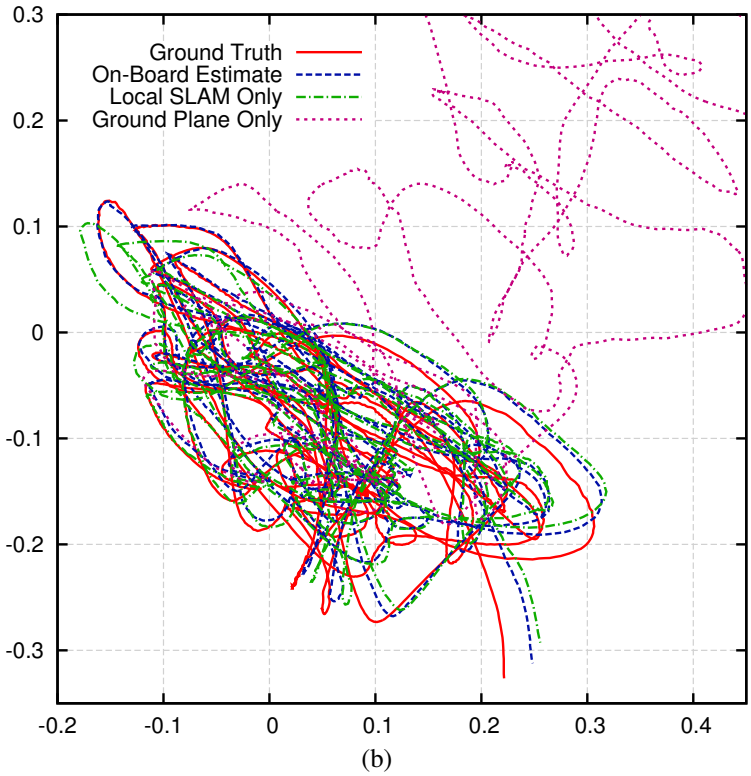
The slow take-off and landing in this experiment, as well as the stable hovering position, are also an easy challenge for the local-SLAM-only test run. Hence, the corresponding curve and the curve for our on-board pose estimates both closely match the recorded ground truth. The ground-plane-only based pose estimate, however, shows accurate height but exhibits high horizontal drift. While in this case the absolute height can be measured, the horizontal position is only obtained through frame-to-frame tracking, which is particularly prone to error accumulation.

Like for the previous single-stereo system, we can quantify the deviation from the ground truth. This happens through examination of the Euclidean distances between the estimated and ground truth positions, which is plotted in Figure 4.15. Furthermore, the average error and RMSE were computed for all position estimates received for this test flight, and are listed in Table 4.2. In both cases, we see that the errors received with our combined method are lower than for the local-SLAM-only system. Most of this improvement can be credited to the more accurate height that we obtain with the combined approach. As one can already anticipate from Figure 4.14, the errors received with the ground-plane-only based method are much higher than for the other two test runs. The corresponding curve has thus been truncated in Figure 4.15.

The more interesting question at hand, however, is how accurately the MAV is able to keep its hovering location. Just like for the single-stereo system, we measure the deviation of the MAV's position from the average position during the hovering periods. If we examine both hovering periods this way, then we receive the errors listed in Table 4.3.



(a)



(b)

Figure 4.14: Ground truth and estimated position during hovering flight in (a) perspective view and (b) top-down view. Scale is in meters.

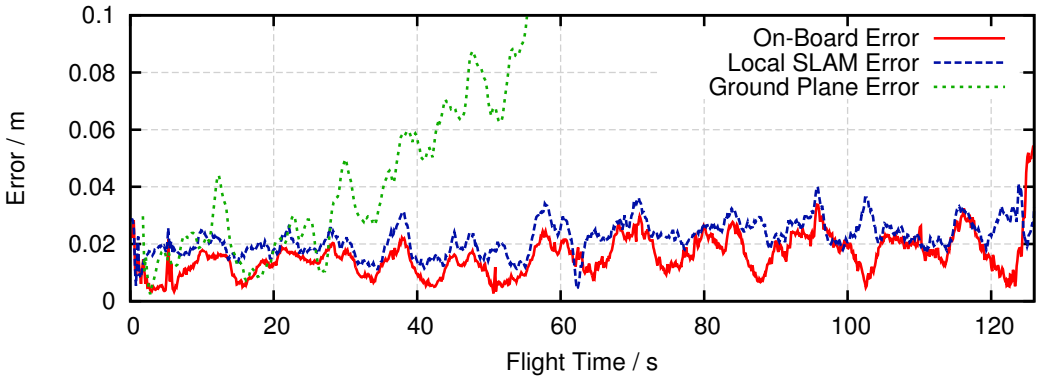


Figure 4.15: Position error during hovering flight.

Table 4.3: Position errors during autonomous hovering.

Height	Average Error	RMSE
0.5 m	10.2 cm	11.4 cm
1.0 m	12.5 cm	14.1 cm

The table reveals that in this test run the MAV was able to keep its position more accurately when hovering at 0.5 m, than when hovering at a height of 1.0 m. Nevertheless, the errors for both altitudes are comparable and should be small enough for safe indoor flight.

The position errors are less than half as large as the ones that we previously received for the single-stereo system in Section 4.3.4. Hence, by using our double-stereo system, our MAV is able to keep its hovering location more precisely. In fact, the reduction of the hovering error is much larger than the error reduction that we observe for the pose estimation. We can hence conclude that the improved hovering ability can in large parts be credited to our modified flight controller, and other system fine tunings that have happened since the construction of the single-stereo prototype.

Runtime Performance Evaluation

Unlike in our previous single-stereo system, most software components are now integrated into a single process. This reduces unnecessary performance overheads, but prevents us from measuring the CPU load that is caused by each component. Hence, for the double-stereo system, we instead measure the total processing time and per-frame average processing time of each processing step. The measurements gained by offline re-processing the data of a previously recorded hovering flight on our MAV hardware, are shown in Table 4.4. In total, this dataset comprised 3796 stereo pairs and 8360 IMU measurements. On average, our system provided 517 successful stereo matches for the forward-facing cameras (63% of the detected features), and 230 successful matches for

Table 4.4: Processing times for individual processing steps.

Proccesing Step	**Per-Frame Average**	**Total Time**
Forward-facing cameras feature detection and stereo matching	17.5 ms	40.4 s
Downward-facing cameras feature detection and stereo matching	18.8 ms	21.5 s
Local SLAM tracking	5.4 ms	12.2 s
Local SLAM mapping	–	2.6 s
Plane detection and tracking	5.0 ms	5.6 s
Sensor fusion	–	0.7 s

the downward-facing cameras (64% of the detected features).

Please note that our MAV is equipped with a dual core CPU. This means that the total processing time for one frame is less than the sum of the individual processing steps, as the CPU can execute two threads in parallel. Furthermore, the given times are the actual time that elapsed until a processing result was available, which does not reflect the actual CPU time that a thread was executed. For sensor fusion and local SLAM mapping, the table does not include an average per-frame processing time. This is due to the fact that the execution of these threads is not synchronized to the video frame rate.

According to the table, performing feature detection and stereo matching for the forward-facing cameras required the most time in total. This is not surprising since those cameras are operated at a higher frame rate than the downward-facing ones. The per-frame processing time, however, was slightly higher for the downward-facing cameras. This is not what one would expect, since our MAV detected more features in the imagery from the forward-facing cameras. The reason for this behavior is that the texture rich scene observed by the downward-facing cameras caused much more features to pass the initial feature detection stage than for the forward-facing cameras.

Since our local SLAM method employs two threads, one for tracking and one for mapping, it is also listed twice in Table 4.4. Because the MAV was hovering in the considered evaluation sequence, the mapping thread only caused a small CPU load, as only few key frames had to be created. Nevertheless, compared to the plane detection and tracking method used for the downward-facing cameras, local SLAM required about 2.6 times as much processing time. Sensor fusion on the other hand, only caused a small computational load with a total processing time of just 0.7 s.

In another experiment, the MAV performed a similar hovering flight while measuring the CPU load on-board. For this flight, all data recording was disabled in order to prevent an influence on the CPU measurements. The average CPU load throughout this

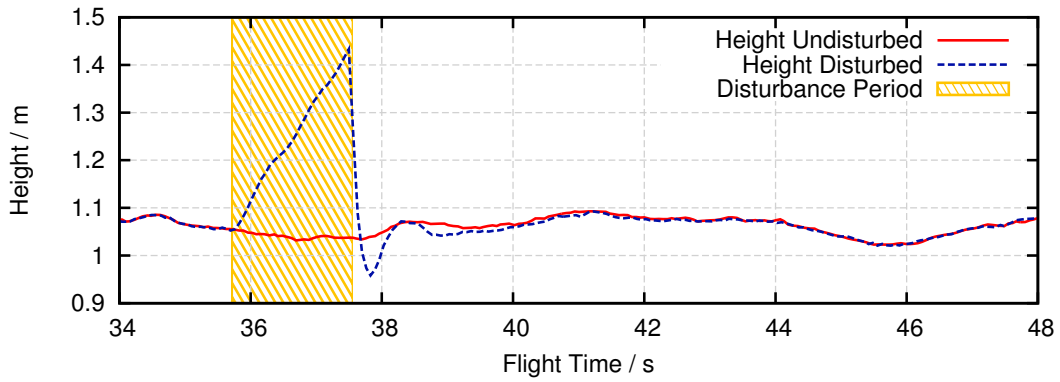


Figure 4.16: Recovery of height estimates after forceful disturbance.

flight was at only 58.9%. This is significantly less than the 85.6% that we received in Section 4.3.4 for our single-stereo system. The performance improvement can be credited to the optimizations discussed in Section 4.4.2 on page 75.

Drift Compensation

The next interesting characteristic of our double-stereo system is its ability to correct drift errors. Unfortunately it is difficult to evaluate the drift correction in a flying experiment. Hence, we instead simulate a flight with significant drift errors. For this purpose, we offline re-process the sensor data that was recorded during an autonomous hovering flight. While the MAV is hovering, an erroneous orientation and height are then forced into the system. This happens by disturbing the output of the sensor fusion for a short period of time. During this time, we keep on applying an erroneous rotation and vertical translation to the fused pose, which is fed back to local SLAM. The disturbance forces local SLAM into recovery, which means that mapping starts again from scratch with the erroneous pose.

The height that was recorded in this experiment is plotted in Figure 4.16. The disturbance was applied during the highlighted section, and lasted for a duration of 1.8 s. For comparison, the undisturbed height that was estimated on-board during autonomous hovering is also included in this figure. We can see that the height from both recordings diverge once the disturbance is applied. Once the disturbance period ends, however, the height measurements quickly converge again to the undisturbed on-board estimates. Similarly, the disturbed and undisturbed roll and pitch angles are shown in Figure 4.17. Again, the angular measurements converge to the undisturbed estimates after the disturbance period has ended. This successfully demonstrates the functioning of our drift correction methods according to expectations.

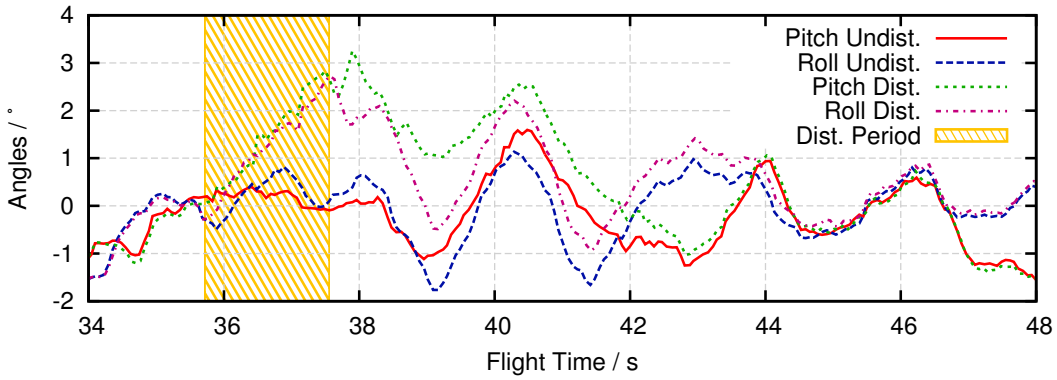


Figure 4.17: Recovery of roll- and pitch-angle after forceful disturbance.

Table 4.5: Average error and RMSE for on-board position estimates during different shape flights.

Shape	Average Error	RMSE
Square	2.20 cm	2.10 cm
Triangle	2.01 cm	2.39 cm
Circle	4.13 cm	4.56 cm

Shape Flight

The previous hovering flights are a particularly easy challenge for the local SLAM method, as the MAV remains mostly stationary and is thus not required to map many keyframes. In further flight experiments, we hence let the MAV fly different horizontal trajectories, while facing in a fixed direction. The flown trajectories resemble different horizontal shapes, which are a square with edge-length 1 m, an equilateral triangle with edge length 1.5 m, and a circle with a diameter of 1.5 m. For the square and triangle, the MAV approached each corner twice before landing autonomously. At each corner it hovered for about 5 s before continuing its flight to the next corner. For the circular shape, the MAV flew two full rotations before landing.

A top view of the recorded ground truth and on-board position estimates for each shape are given in Figures 4.18a – 4.18c. For clarity, the trajectories during the autonomous take-off and landing period have been omitted. It can be seen that our on-board pose estimates closely match the recorded ground truth for all shape flights. The average error and RMSE for the on-board position estimates are listed in Table 4.5. The square and triangle flight produced similar error magnitudes, while there are larger errors for the circle flight. These errors are considerably larger than the errors we previously analyzed for the hovering experiment. Nevertheless, the errors are still within the low centimeter range, which should be sufficient for safe indoor navigation.

A more detailed evaluation was performed for the square flight, by offline re-processing the recorded sensor data with our local-SLAM-only and ground-plane-only methods.

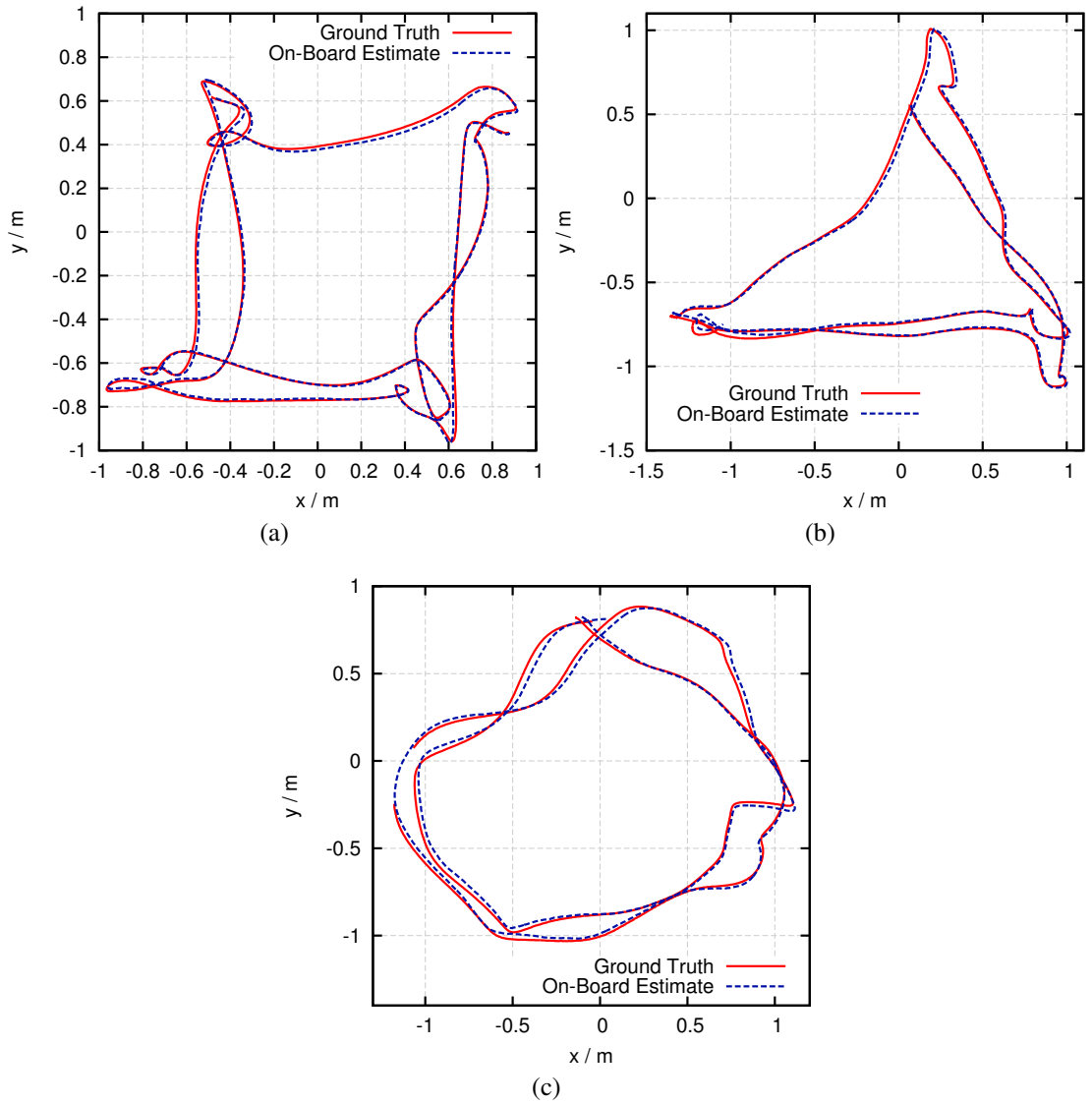


Figure 4.18: Flight of horizontal (a) square, (b) triangle and (c) circle shape. One can see that while MAV's controller can certainly be improved, the on-board pose estimates from our double-stereo system accurately match the recorded ground truth.

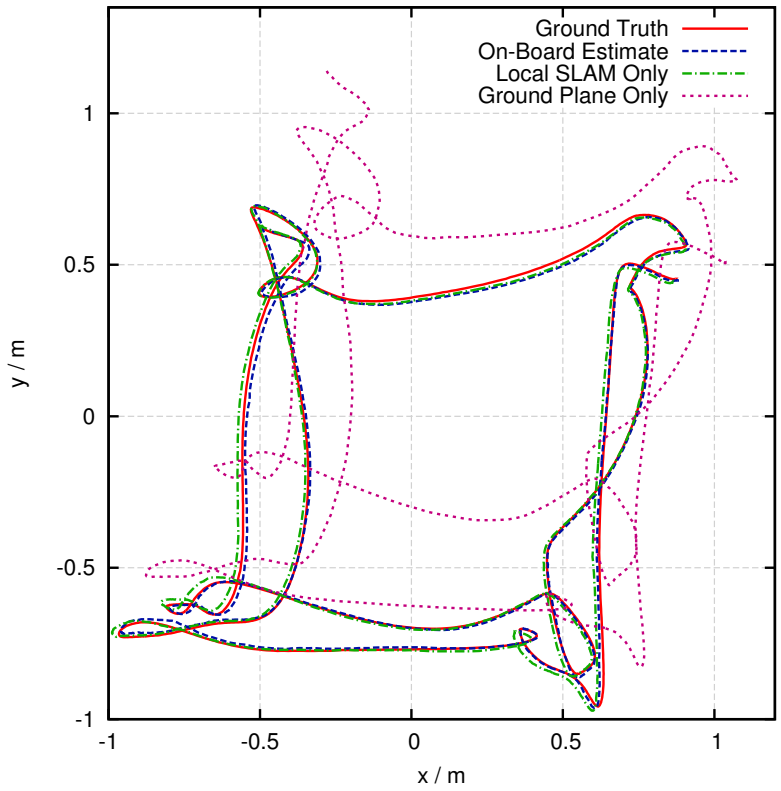


Figure 4.19: Square flight trajectory including offline processing results.

The trajectory estimates obtained with each approach are shown in Figure 4.19. These results show the same tendency that we previously observed in the hovering experiment: while our on-board estimates and the local-SLAM-only test run provide results that closely match the recorded ground truth, the ground-plane-only version exhibits significant drift.

This observation is also confirmed if we examine the deviation of the position estimates from the recorded ground truth, as shown in Figure 4.20. Like for the previous hovering experiment, the error curve that corresponds to our ground-plane-only version has been truncated, due to its large error magnitudes. Compared to our on-board estimates, the local-SLAM-only test run again produced higher position errors. The average error and RMSE for all tested methods are listed in Table 4.6.

Despite the relatively small dimensions of the flown shape trajectories, the movements performed by the MAV are large enough to force our local SLAM method into continuously adding new keyframes. Hence, even though this experiment was performed in a small confined space, the received performance results should resemble the MAV's performance on a long-range flight.

Table 4.6: Average error and RMSE for different pose estimation methods, when processing the recorded square flight.

Method	Average Error	RMSE
On-Board Estimate	2.20 cm	2.39 cm
Local SLAM Only	4.39 cm	4.53 cm
Ground Plane Only	29.35 cm	33.71 cm

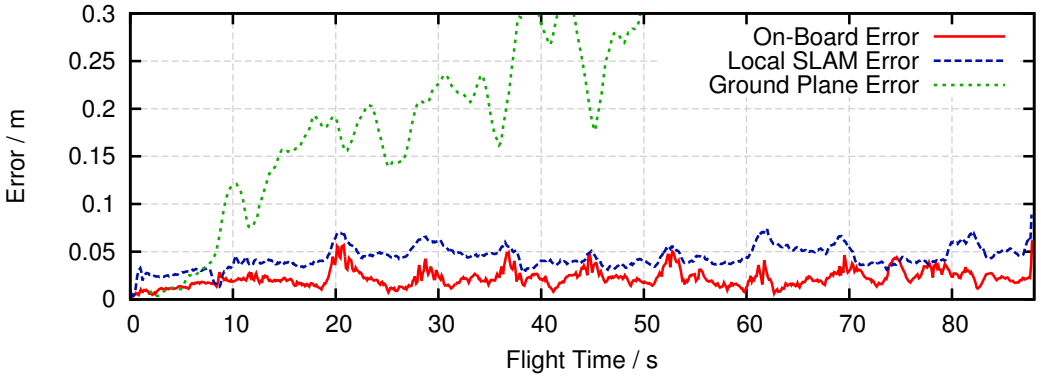


Figure 4.20: Position error during square flight.

Yaw Rotations

While the previous flight experiments are also feasible with only two cameras and our local SLAM system, the situation is different if we encounter yaw rotations. As we have discussed earlier, observing fast yaw rotations with the forward-facing cameras is particularly challenging. Thus, we expect that our MAV benefits much from our additional downward-facing cameras in this case. This assumption has been put to a test, by letting our MAV perform a 360° yaw rotation. This rotation was divided into four separate 90° turns, for which our MAV required an average time of 2.3 s each. After each turn, the MAV waited for itself to stabilize and then hovered for 5 seconds before continuing with the next turn. An example for the scene observed by the forward-facing cameras after each turn is shown in Figure 4.21.

Figure 4.22 contains the recorded ground truth and on-board position estimates for a typical test run of this experiment. Again, the recorded camera imagery and IMU measurements were re-processed offline with a local-SLAM-only and ground-plane-only version of our software system. These additional results are once more included in Figure 4.22. We can see that despite the yaw rotations, the MAV is able to maintain an accurate estimate of its current position. The ground-plane-only test run again shows the already observed behavior of accurate height estimates but strong horizontal drift.

The position estimated by the local-SLAM-only version, on the other hand, shoots off in a random direction after the first 90° turn. Please note that the diagram has been truncated and that the position estimation continues to show the same erroneous behavior

(a) 0° (b) 90°

(c) 180° (d) 270°

Figure 4.21: Scene observed by the forward-facing cameras during 360° yaw rotation.

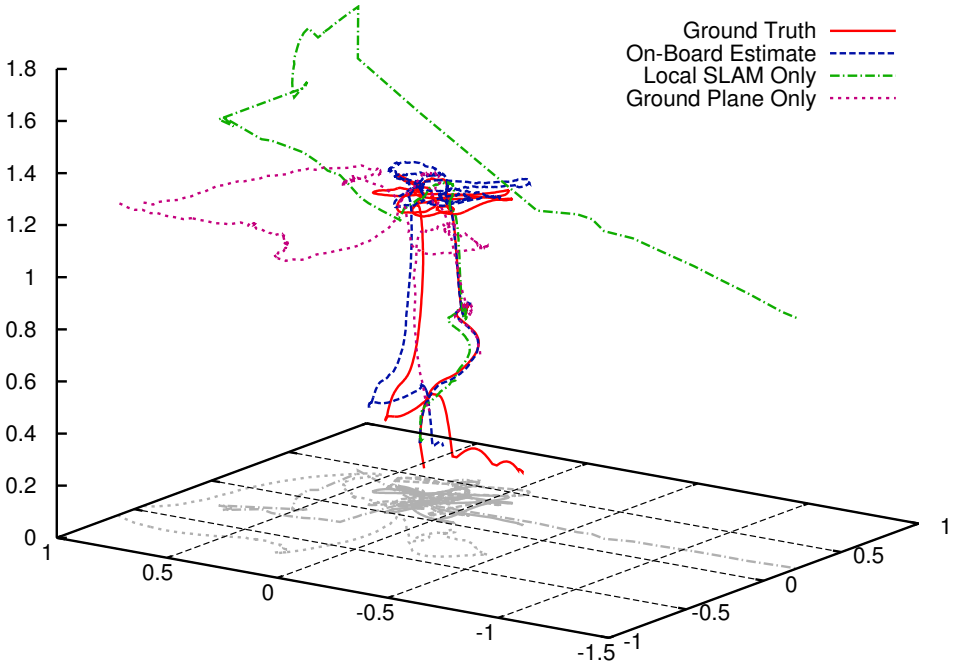


Figure 4.22: Ground truth and estimated position during yaw rotation. Scale is in meters.

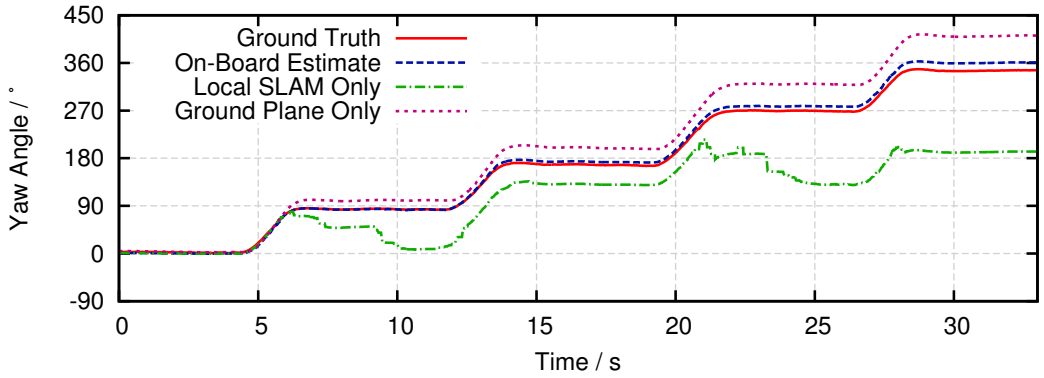


Figure 4.23: Yaw angles measured during 360° yaw rotation.

for each of the four turns. In fact, it has not been possible to obtain a valid position estimate beyond the first turn, for any test-run with the local-SLAM-only system. If the MAV had used this erroneous position estimate for navigation, this would have inevitably led to a crash.

The recorded and estimated yaw rotations are depicted in Figure 4.23. In this diagram we can see that the yaw rotation estimated with our on-board method closely follows the ground truth, while the ground-plane-only version follows the ground truth less accurately. The local-SLAM-only version starts deviating significantly after the first turn, which matches our previous observations from Figure 4.22.

The good performance of our method can in large parts be credited to our new recovery strategy. In fact, recovery of the local SLAM method was performed once during each turn. Because the more rotation-robust pose from the downward-facing cameras is used during recovery, our MAV was able to keep an accurate pose estimate throughout the experiment.

4.5 Summary and Discussion

The key problem in autonomous MAV navigation is enabling the MAV to estimate its current pose. Only if the MAV is aware of its position and orientation, it is able to steer itself towards a desired target location. While GPS and IMU measurements are frequently used for autonomous outdoor-flight, different methods are required for autonomous flight in GPS-denied spaces. In this chapter, we have discussed two such approaches that both rely on stereo vision. Furthermore, both methods have been implemented on our MAV platform and we have proven their autonomous flight capabilities in several experiments.

We first discussed the single-stereo system, which only uses the two forward-facing cameras that are present on our MAV research platform. This method relies on an adapted version of the PTAM visual SLAM system, which has been extended by Scherer *et al.* (2012) to incorporate depth information. We simplified this visual SLAM method such

that it only retains a small local map, which is achieved by continuously dropping old keyframes. The resulting SLAM system has been called local SLAM in this thesis. The obtained pose estimates are fused with inertial measurements that are received from the IMU. The fused pose is then passed on to the MAV's position controller and also fed back to local SLAM for motion prediction.

The second method that we discussed in this chapter is the double-stereo system that uses both the forward-facing and the downward-facing camera pairs. Particularly in the case of fast yaw rotations, the imagery of the downward-facing cameras is more suitable for visual motion estimation. For ground proximity flights, on the other hand, the forward-facing cameras are more viable. Hence, the initial motivation for simultaneously using a forward-facing and a downward-facing camera pair was the expectation that both camera pairs would complement one another.

Because the computer on-board our MAV does not provide sufficient computing resources for simultaneously running two instances of our local SLAM software, a more efficient technique has been designed for processing the imagery from the downward-facing cameras. This method expects a flat ground and fits a plane to the 3D points obtained through stereo matching. This allows the MAV to measure its relative height towards the ground plane. By assuming that the ground is level, the MAV can further estimate its roll and pitch angles. For observing horizontal translations and yaw rotations, a method based on image tracking has been used. Hence, a full 6-DoF pose estimate is obtained from the downward-facing cameras, which provides an alternative to the pose from local SLAM. Both pose estimates and measurements from an IMU are then fused, while also compensating drift errors for height, roll and pitch.

The single-stereo and double-stereo systems have both been evaluated in flight experiments. While initially only a brief proof-of-concept evaluation was performed for the single-stereo method, a thorough evaluation has been provided for the double-stereo system. For the latter one, we also re-processed the sensor data recorded during several test flights with only the local SLAM method, which matches our single-stereo system. This allowed us to judge the performance improvement gained by the additional downward-facing camera pair. As expected, the highest improvement is achieved when performing fast yaw rotations. In this case, the single-stereo method loses track and fails to provide a reliable pose estimate, while our method continues to deliver accurate pose estimation results.

The two presented visual navigation systems make our MAV truly autonomous, which means that it does not depend on external devices such as ground computers or tracking systems. Our MAV successfully demonstrated autonomous take-off, landing, hovering, shape flight and 360° yaw rotations. However, if we want our MAV to perform meaningful tasks, visual navigation alone is not sufficient. Rather, instead of just following pre-programmed flight maneuvers, the MAV should be able to react on its environment. This requires a method for environment perception, which is the focus of the next chapter.

Chapter 5

Stereo-Based Obstacle and Environment Perception

5.1 Introduction

In the previous chapter we have discussed methods for enabling our MAV to perceive its current pose. This allows the MAV to navigate autonomously, which facilitates the flight of different pre-programmed trajectories. Unfortunately, however, our MAV flies 'blind', which means that it does not attempt to perceive its environment. Our stereo matching method only matches a sparse set of features, of which a subset is added to the map of our local SLAM system. Only considering this small set of features is unfortunately not sufficient for the reliable detection or identification of obstacles and traversable free space. Hence, should an obstacle appear on the MAV's pre-programmed flying trajectory, then the MAV will not attempt to alter its trajectory but remain on a collision course. To facilitate intelligent behavior such as obstacle evasion, the MAV requires means to perceive its environment. In particular, it is required to detect space that is blocked by obstacles, and free space that is traversable for the MAV.

Even if the MAV is pre-programmed with a collision-free flying trajectory, the perception of its environment is still important. If the MAV is expected to operate in the close vicinity of obstacles, then drift errors in the MAV's pose estimate could cause the MAV to deviate from the intended trajectory and lead to potentially dangerous situations. If the MAV is expected to perform long-range flights, then the accumulation of drift errors can even render objects with a significant distance to the desired trajectory as a potential threat to the MAV.

Environment perception can further facilitate new and intelligent behavior and applications. If the MAV is aware of traversable and non-traversable space, it can perform fully autonomous path planning. Particularly in a dynamic environment, such on-board path planning is necessary for an MAV to reach its dedicated destination. An application for MAVs that are capable of autonomous path planning is the exploration of unknown environments. For this task, an MAV could autonomously map its environment or search for and locate known objects.

The mapping of free space and space that is occupied by obstacles is a problem that

has been well studied for ground-based robots, and is generally referred to as *occupancy mapping*. Because a ground-based robot can only move in two dimensions, most existing occupancy mapping methods are based on 2D grids. Typical sensors that are used for creating such 2D occupancy grid maps are 2D laser scanners and sonar sensors. Our MAV, however, is able to move in three dimensions, which is why we also need a 3D mapping method. Fortunately, the stereo cameras on-board our MAV allow a three-dimensional perception of the MAV's environment, and can be used for populating a 3D occupancy map.

When extending the common 2D grid representation to 3D, we arrive at a volumetric occupancy map. If implemented naïvely, such maps can consume excessive amounts of memory. However, an efficient method has recently been proposed by Wurm *et al.* (2010) and Hornung *et al.* (2013), which uses compressed octrees. The authors have made their implementation, which they named *OctoMap*, publicly available (see Hornung *et al.*, 2014) and it has since been used in numerous research projects.

In the initial publication of OctoMap, data from an accurate 3D laser scanner was used for evaluating the mapping performance. Compared to laser scanners, the data received from a stereo vision system is unfortunately much noisier. In particular, the stereo vision noise increases quadratically with the measured depth (see Point Grey Research, Inc., 2012), and it is often spatially and temporally correlated.

Stereo matching algorithms usually employ an explicit or implicit smoothness constraint that penalizes solutions with abruptly varying depth. Consequently, if a stereo matching result is wrong for one image location, all neighboring image locations are likely to exhibit a similar error. Furthermore, because stereo matches are determined by image similarity, similarly textured regions are likely to be repeatedly mismatched in subsequent frames. Thus, we cannot assume that on average measurement errors cancel out each other.

The dissimilar nature of the noise found in laser and stereo range measurements means that methods designed for processing laser range data do not necessarily perform well when applied to stereo vision. In terms of OctoMap, the correlated noise inherent in stereo measurements leads to many falsely mapped artifacts, as we will see in Section 5.4 on page 115. To ensure more accurate maps for our MAV, we hence require a more robust method.

In this chapter, one such method is presented, which is a modification of the original OctoMap approach. As shown by evaluation, this new method is more accurate when used in conjunction with stereo vision, while also exhibiting a smaller memory footprint. At the same time, this method achieves shorter processing times than the original OctoMap implementation for most of the test runs performed in this chapter. The presented work was first published at the IEEE International Conference on Robotics and Automation (ICRA) in 2014 (Schauwecker and Zell, 2014b).

5.2 Related Work

As mentioned above, occupancy maps can be created in two or three dimensions. Earlier work typically focused on 2D maps due to their low memory consumption and computational requirements. A compromise between 2D and 3D maps are 2.5D elevation maps, which however cannot be used to represent all 3D environments. We have a separate look at each of those map types and the relevant mapping methods for creating them. Furthermore, we have a look at existing work on creating occupancy maps with autonomous MAVs.

5.2.1 2D Occupancy Maps

The idea of using a regular 2D grid for mapping the occupancy of a robot's environment was first published by Moravec and Elfes (1985). The authors used a robot equipped with a wide-angle sonar sensor to measure the distance to close-by obstacles. For this method, the probabilities of a grid cell being occupied $\mathrm{P}(O)$ or free $\mathrm{P}(\neg O)$ are tracked independently. If, for a given cell, $\mathrm{P}(O)$ is greater than $\mathrm{P}(\neg O)$, then the cell is considered occupied. A sensor model is used that assigns occupancy and free probabilities to any point within the sonar beam. After several pre-processing steps, those probabilities are integrated into the occupancy map by using a probabilistic addition formula. An evaluation of the method from Moravec and Elfes among other popular 2D occupancy mapping approaches was published by Collins *et al.* (2007).

The probability integration scheme used by most of today's occupancy mapping methods follows the approach applied by Matthies and Elfes (1988) and Elfes (1989). Here, only the occupancy probability $\mathrm{P}(O)$ is integrated for each cell, as a cell's probability of being free is complementary, and can thus be inferred from $\mathrm{P}(O)$. Integration happens through recursive Bayesian estimation. Thus, this method again requires a probabilistic sensor model for obtaining the posterior occupancy probability, given the current sensor observation.

Without affecting the results, the performance of the recursive Bayesian estimation can be improved if the map stores the logarithm of the odds ratio, which is known as *log-odds* (see Thrun *et al.*, 2005), for the occupancy probability $\mathrm{L}(O)$. In this case, the update equation can be reduced to a simple log-odds summation. Such a log-odds based map has *e.g.* been used by Konolige (1997). This method also includes extensions to improve robustness against specular reflections and redundant readings, which are common problems for sonar sensors.

Although stereo cameras are 3D sensors, they can also be used for creating 2D occupancy maps. This requires the stereo matching results to be reduced to 2D measurements, which usually happens through a column-by-column projection of the disparity map. A robot that uses such a method to obtain a 2D occupancy map was shown by Murray and Little (2000). This robot is equipped with a trinocular stereo camera, whose imagery is processed with a multi-baseline stereo matching algorithm based on the method from

Okutomi and Kanade (1993). The mapping method selects the maximum disparity value of each disparity map column during the column-by-column projection. To account for the loss of depth resolution for distant points, each projected column results in an update of a trapezoid shaped section of the occupancy grid map.

Another stereo vision based method for 2D occupancy mapping, which is focused on vehicular applications, has been proposed by Perrollaz *et al.* (2012). In this approach, the y axis of the occupancy map is always aligned to the reference camera's optical axis. Furthermore, coordinates along this axis are measured in stereo disparity. Due to the axis alignment, this method does not allow for changes in the camera's viewing direction. Rather, the authors focus on the creation of 'instantaneous' occupancy maps that assume a static camera rotation.

For this method, the disparity map is first segmented into road surface and an obstacle maps, for which the technique developed by Perrollaz *et al.* (2010) is used. A column-by-column projection is then only performed for the obstacle disparity map, by counting the frequency of disparity values in each image column. The resulting representation was called u-disparity, due to its similarity with the popular v-disparity representation (Labayrade *et al.*, 2002). The observations from the u-disparity representation are integrated into the occupancy map by using a probabilistic approach. The integration scheme considers the visibility of a cell, which is modeled as the ratio of pixels with smaller disparity in the corresponding column of the disparity map, and the number of possible measurement points. For cells that are not visible, no update of the occupancy probability is performed.

5.2.2 2.5D Elevation Maps

Planar grid-based maps only provide a reduced view of the 3D world in which the robot operates. A more informative way for environment representation are 2.5D elevation maps, where the surface elevation is stored for each grid cell. Such methods have long been used in the robotics community to perform terrain mapping. As an example, Kweon and Kanade (1990) developed one such method alongside a 3D vision system, which were intended for a planetary exploration robot. While the grid-based representation of elevation maps allows for an efficient processing, other map representations are also possible. For example, Hadsell *et al.* (2009) model the elevation map with an elevation function over a 2D domain, which is found using a kernel-based method.

Independent of the used map representation, elevation maps have a significant limitation in the types of environments that they can represent. Using a single elevation map, it is not possible to model overhanging structures, which makes these methods particularly unsuitable for mapping indoor environments. This constraint can be weakened by allowing multiple surface levels as *e.g.* done by Triebel *et al.* (2006). Here, the authors store a list of 'surface patches' for each cell of a grid-based map. This method also supports vertical structures, which *e.g.* facilitates the mapping of vertical building walls.

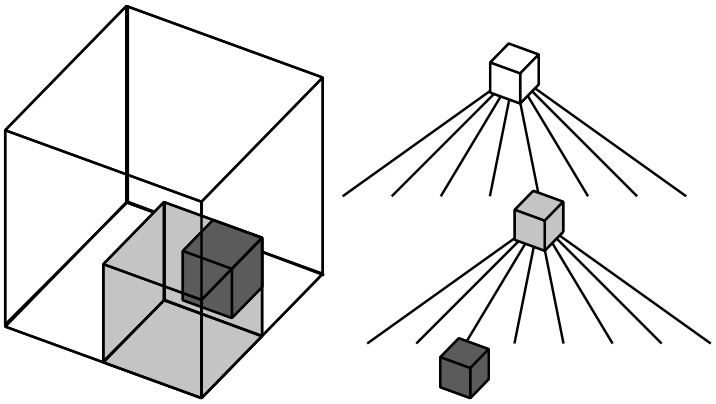


Figure 5.1: (a) Recursive volumetric subdivision and (b) tree based data structure used by octrees.

5.2.3 3D Occupancy Maps

A more general representation of 3D environments are volumetric occupancy maps. Examples for such methods are the systems developed by Ryde and Hu (2010) and Dryanovski *et al.* (2010). Both approaches employ map representations that are again based on 2D grids. Ryde and Hu store a list of occupied and free volumes for each grid cell. These volumes are allowed to have vertical non-integer extents, while restrictions apply for minimum height and vertical distance. Triebel *et al.*, on the other hand, only store occupied volumes. This time, the stored volumes are cube-shaped voxels that are kept in hash maps. In both cases, data recorded with 3D laser scanners was used for evaluating the mapping performance.

In this chapter, we focus on volumetric occupancy maps that are based on *octrees*. Octrees are efficient tree-based data structures that can be used for storing 3D points, and were initially pioneered by Meagher (1980). For this method, the available 3D space is represented as a cube. If a point within this 3D space shall be stored in the octree, the space is recursively divided into smaller and smaller sub-cubes. For this process, each cube is divided into eight child-cubes that are stored in a tree-based data structure. Figures 5.1a and 5.1b illustrate this recursive subdivision and tree representation. The advantage of using octrees for storing volumetric data is that memory is only allocated as needed, since empty cubes do not occupy memory. Otherwise, even a small volumetric map could consume large amounts of memory, as the size grows cubically with the map resolution.

A stereo vision based mapping method that uses a data structure similar to octrees was published by Bajracharya *et al.* (2012). Here, the authors use an N-tree that divides a cube into N child-cubes, which is used to store a set of occupied voxels. This data structure was chosen in order to allow for an efficient neighbor lookup, which is done excessively by the proposed method. A voxel is marked as occupied if part of the

obtained range measurements from stereo vision fall within the voxel's boundaries. Different filtering methods are then applied to this volumetric occupancy map. In a temporal filter, voxels are 'aged' if no corresponding observations are made despite the voxels being within current field of view. Once a voxel reaches a certain age, it is removed from the map. Similarly, a spatial filter is applied that is based on a weighted sum, which is computed on a local voxel neighborhood.

Another octree-based method for volumetric occupancy mapping is the already mentioned OctoMap (Wurm *et al.*, 2010; Hornung *et al.*, 2013). The authors have made their implementation publicly available (see Hornung *et al.*, 2014), which has led to its adoption in numerous research projects. Although OctoMap is mostly used for processing range measurements from laser scanners, there exist several examples where it has been applied to process data from stereo vision, like the autonomous exploration system developed by Shade and Newman (2011).

The map generated by OctoMap consists of voxels that store the log-odds of the occupancy probability $\mathrm{L}(O)$. A voxel is considered occupied if the occupancy probability $\mathrm{P}(O)$ is greater than 0.5, which is equivalent to the occupancy log-odds $\mathrm{L}(O) > 0$. For integrating new range measurements, OctoMap uses a ray casting scheme, which is based on the 3D extension of the Bresenham algorithm that was proposed by Amanatides and Woo (1987). A ray is cast from the sensor origin to each received measurement end point. A hit-measurement (*i.e.* an occupied observation) is then generated for the voxel containing the measurement end point, while miss-measurements (*i.e.* free observations) are generated for all remaining voxels on the ray. The occupancy log-odds $\mathrm{L}(O)$ of all affected voxels are then updated through recursive Bayesian estimation.

In order to keep the memory consumption low, OctoMap performs a continuous compression of the generated octree. This happens by introducing a maximum and minimum probability threshold p_{max} and p_{min}, to which the occupancy probabilities (or rather the corresponding log-odds) are clamped. If for a given node in the octree, all child-nodes reach the maximum or minimum probability, then the child-nodes can be pruned from the tree. In this case, it is sufficient to only keep one single copy of the occupancy log-odds in the parent-node.

The minimum and maximum probabilities also serve one further purpose: If a voxel's occupancy probability would reach a very high or very low value, then many observations would be required for changing the voxel's occupancy status. Hence, by introducing an upper and lower probability bound, the updatability of all voxels is assured.

5.2.4 Occupancy Mapping with Autonomous MAVs

Several autonomous MAVs have been presented in literature that are able to map their environments. Some of these MAVs were previously mentioned in Section 4.2.3 on page 60, when we discussed autonomous navigation. One such MAV is the autonomous quadrotor that was developed by Shen *et al.* (2011). This MAV is equipped with a 2D laser scanner that performs range measurements within a horizontal plane. Measure-

ments from the laser scanner are used to construct a 2D occupancy grid map, while only using on-board processing resources. Despite this map being only two-dimensional, the MAV is able to map indoor environments with multiple floors. This happens by automatically creating a new map layer when the MAV transitions from one floor to another.

Huang *et al.* (2011) presented a quadrotor MAV that is able to construct 3D maps. Although their MAV performs on-board motion estimation using a VO-based method, the 3D map is computed off-board. As range sensor serves a stripped-down Microsoft Kinect RGB-D camera. The map is created with the method developed by Henry *et al.* (2010), which is based on surface elements or *surfels*. Surfels, which are a concept that is known from computer graphics, (Pfister *et al.*, 2000) are small surface patches that have a size, location, surface orientation and color. In addition, Henry *et al.* also store a confidence measure for each surfel in the map. This confidence measure increases if the surfel is seen from multiple angles, while it decreases if the camera can 'see through' it. Surfels with a low confidence are then removed from the map.

One example for an MAV that performs occupancy mapping using stereo vision is the previously discussed PIXHAWK quadrotor, which was developed by Heng *et al.* (2011). Unlike the just mentioned MAV, this quadrotor is able to perform all computations on-board, and does not depend on external systems. At first, a dense stereo matching algorithm is run whose results serve as range measurements. These measurements are then processed with OctoMap to provide a 3D volumetric occupancy map. This MAV was extended by Meier *et al.* (2012) to allow for larger image resolutions, and by Fraundorfer *et al.* (2012) to facilitate path planning for autonomous exploration tasks.

Another MAV that uses OctoMap for creating volumetric occupancy maps was presented by Fossel *et al.* (2013). The range sensor that is used by this MAV is a planar 2D laser scanner. Using an estimate for the MAV's current pose, the 2D laser range measurements can be transformed to a set of 3D measurement end points, which are then processed by OctoMap. While the MAV's altitude and roll and pitch angles are determined using measurements from an ultrasound altimeter and IMU, the horizontal position and yaw rotation are inferred with a SLAM algorithm that relies on the generated occupancy map. The accuracy of this SLAM algorithm, however, is limited by the voxel size. This method was not run on-board of an MAV, but only applied offline to recorded or simulated sensor data.

5.3 Method

As mentioned previously, the correlated noise inherent in range measurements from stereo vision can promote the mapping of many erroneous artifacts. An example for this behavior can be seen in Figure 5.2, where two intersecting corridors have been mapped with OctoMap. For this example, range measurements were obtained by using a stereo camera and the Efficient LArge-scale Stereo (ELAS) algorithm, which was developed by Geiger *et al.* (2011). The erroneous artifacts occur in map regions that are occluded and

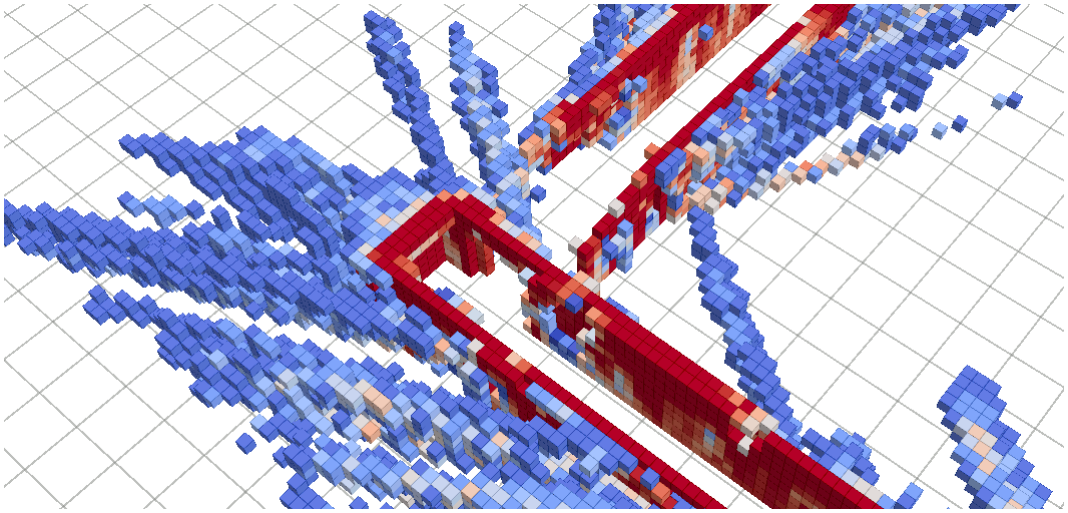


Figure 5.2: Example for artifacts occurring when using OctoMap with data from stereo vision.

thus not visible to the sensor. In the given example, these are the areas behind the walls of the mapped corridors.

The formation of these artifacts can be explained as follows: Because the occluded map regions are not visible to the sensor, no correct sensor measurement is ever received for the corresponding voxels. Instead, all measurements for these regions originate from errors in the stereo matching results. For the visible map region, such occasional errors do not pose a significant problem, as we receive a sufficient number of correct measurements for the voxels within the current field of view. For the occluded map area, however, no correct measurements are ever received, which means that these errors can accumulate. In fact, the error accumulation is much accelerated by the correlated nature of the stereo matching errors. This means that for an occluded voxel, we might quickly receive several similar erroneous measurements, which triggers the mapping of an erroneous artifact.

In the work from Bajracharya *et al.* (2012), this problem is resolved by using the mentioned spatial and temporal filtering. We, however, aim at finding a method that solves this problem inherently. This can be achieved by considering the visibility of a voxel when updating its occupancy probability. If we know that a given map voxel is currently not visible, then all measurements received for this voxel must be erroneous, which means that the voxel's occupancy probability should not be altered. The update behavior for voxels that are known to be visible, on the other hand, should remain unchanged.

Before updating the occupancy probability of a map voxel, we hence need to estimate the probability that this voxel is currently visible. In this sense, the presented method is similar to the one proposed by Perrollaz *et al.* (2012). However, since Perrollaz *et al.*

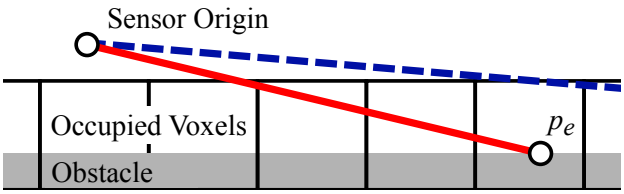


Figure 5.3: Observing an obstacle at a flat angle.

use a 2D method, the visibility is measured for a column of the disparity map only. The measure they chose is the fraction of pixels with a disparity less than a voxel's reference disparity. Hence, the visibility measure depends only on the current measurement, and does not incorporate knowledge from the created map.

The visibility measure that we propose in this chapter differs significantly from the aforementioned. First, our measure is not limited to 2D space, but can instead estimate the visibility of voxels in a 3D map. Second, the measure is derived from the created occupancy map, and not from the current range measurement. A map created by integrating multiple sensor measurements is generally more accurate than a single measurement alone. In the following, we discuss this visibility measure, the probability integration scheme and further enhancements that have been made to the original OctoMap implementation.

5.3.1 Visibility Estimation

Given a ray from the sensor origin that passes through a voxel v, the naïve model for the visibility of v would be the probability that all voxels that are on the ray and closer to the sensor origin are free (*i.e.* not occupied). Should a single voxel on the ray be occupied and closer to the sensor origin, then one could assume that this voxel would occlude v. This simple and intuitive model for the visibility of v unfortunately has a poor performance when observing an obstacle at a flat angle, as shown for the solid red ray in Figure 5.3. The elongated obstacle that is depicted in this example encompasses a row of voxels below the sensor origin. The measurement ray passes through several of these voxels before reaching the measurement end point p_e. Hence, the voxel containing p_e would be determined as not visible, despite the fact that the measurement end point is clearly observable.

We thus use a different method for modeling voxel visibility, which is based on *local occlusion*. We consider a voxel v as locally occluded if it is occluded by its direct neighbors for all possible rays that pass through the sensor origin and v. In the example in Figure 5.3, the voxel containing p_e is not occluded by its direct neighbors for the dashed blue ray.

A voxel v usually has three faces that are visible to the sensor. We consider a set N_v of three voxels that border these visible faces, as illustrated in Figure 5.4. We define v as locally occluded if all voxels in N_v are occupied, which we indicate with the event

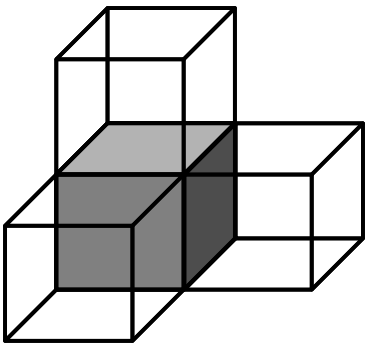


Figure 5.4: Visible faces of a voxel with neighboring voxels.

C_v. Unfortunately, there exists a strong conditional dependency among the occupancy probabilities of the three voxels in N_v, which complicates the calculation of the local occlusion probability $\mathrm{P}(C_v)$. Hence, we use a simple estimate for $\mathrm{P}(C_v)$, which is the smallest occupancy probability of the voxels in N_v. If $\mathrm{P}(O_v)$ denotes the probability that voxel v is occupied, then we can approximate $\mathrm{P}(C_v)$ as follows:

$$\mathrm{P}(C_v) \approx \min\{\mathrm{P}(O_a), \mathrm{P}(O_b), \mathrm{P}(O_c)\}, \text{ with } \{a,b,c\} \in N_v. \tag{5.1}$$

Now, we consider a ray R with a set of voxels $v_i \in R$. For a voxel v_i, we assume that its visibility depends on the event C_{v_i} that this voxel is locally occluded and the event $V_{v_{i-1}}$ that the previous voxel on the ray, which is closer to the sensor origin, is visible. Given the law of total probability, we can compute the probability $\mathrm{P}(V_{v_i})$ that voxel v_i is visible as follows:

$$\begin{aligned}\mathrm{P}(V_{v_i}) = {} & \mathrm{P}(V_{v_i}|C_{v_i}, V_{v_{i-1}})\,\mathrm{P}(C_{v_i})\,\mathrm{P}(V_{v_{i-1}}) \\ & + \mathrm{P}(V_{v_i}|\neg C_{v_i}, V_{v_{i-1}})\,\mathrm{P}(\neg C_{v_i})\,\mathrm{P}(V_{v_{i-1}}) \\ & + \mathrm{P}(V_{v_i}|C_{v_i}, \neg V_{v_{i-1}})\,\mathrm{P}(C_{v_i})\,\mathrm{P}(\neg V_{v_{i-1}}) \\ & + \mathrm{P}(V_{v_i}|\neg C_{v_i}, \neg V_{v_{i-1}})\,\mathrm{P}(\neg C_{v_i})\,\mathrm{P}(\neg V_{v_{i-1}}).\end{aligned} \tag{5.2}$$

We define the probability of voxel v_i being visible to be 0, if the previous voxel on the ray v_{i-1} was not visible. This means that the probability $\mathrm{P}(V_{v_i})$ will never be greater than $\mathrm{P}(V_{v_{i-1}})$. Hence, the visibility can only decrease when traversing the ray from the sensor origin to the measurement end point. With this additional restriction, we can simplify Equation 5.2 and receive our final visibility estimation formula:

$$\begin{aligned}\mathrm{P}(V_{v_i}) = {} & \mathrm{P}(V_{v_{i-1}})[\mathrm{P}(V_{v_i}|C_{v_i}, V_{v_{i-1}})\,\mathrm{P}(C_{v_i}) \\ & + \mathrm{P}(V_{v_i}|\neg C_{v_i}, V_{v_{i-1}})\,\mathrm{P}(\neg C_{v_i})].\end{aligned} \tag{5.3}$$

This simplified equation contains only two remaining conditional probabilities, which are $\mathrm{P}(V_{v_i}|C_{v_i}, V_{v_{i-1}})$ and $\mathrm{P}(V_{v_i}|\neg C_{v_i}, V_{v_{i-1}})$. We assume that both of these conditional

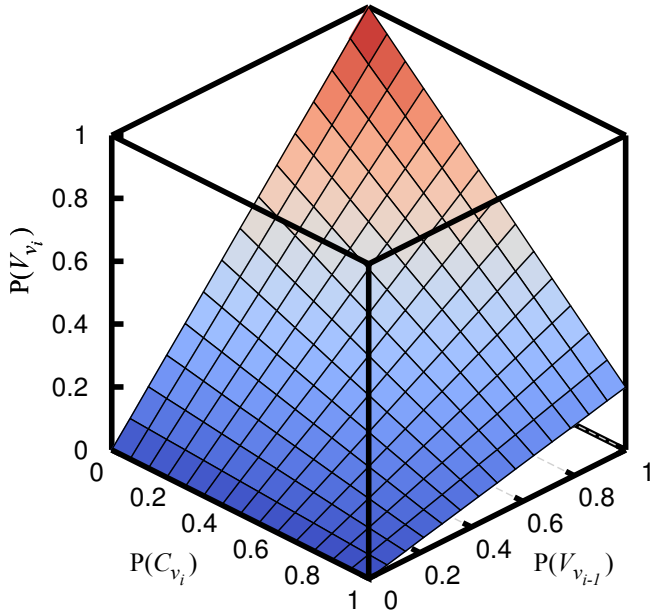


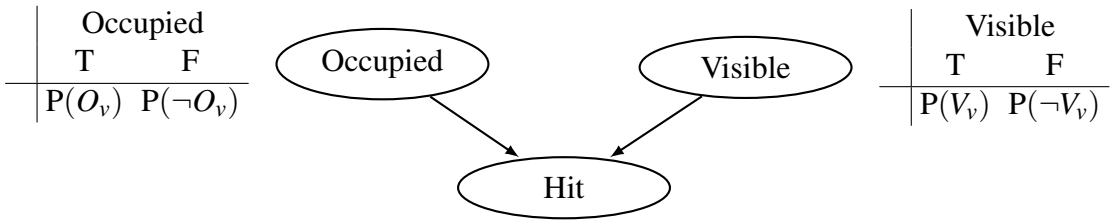
Figure 5.5: Visualization of the equation used for estimating the voxel visibility probability $P(V_{v_i})$.

probabilities are constant, and configure them according to the sensor properties. The probability $P(V_{v_i}|C_{v_i}, V_{v_{i-1}})$ controls how quickly the visibility reduces when encountering occupied voxels. $P(V_{v_i}|\neg C_{v_i}, V_{v_{i-1}})$ on the other hand controls how quickly the visibility reduces when encountering free voxels. For a stereo camera, this parameter should be set to 1, as there exist no range limitations for stereo vision. When using sensors with significant range limits, however, one might want to select smaller values.

In Figure 5.5, the received visibility probability $P(V_{v_i})$ has been plotted in relation to the local occlusion probability $P(C_{v_i})$ and the probability $P(V_{v_{i-1}})$ that the previous voxel on the ray was visible. The parameters that have been used for this figure are $P(V_{v_i}|C_{v_i}, V_{v_{i-1}}) = 0.2$ and $P(V_{v_i}|\neg C_{v_i}, V_{v_{i-1}}) = 1$. This matches the paramerization that is used later for our performance evaluation.

OctoMap performs a clamping of occupancy probabilities to the interval $[p_{min}, p_{max}]$. In our case, the upper threshold p_{max} can cause a slight underestimation of voxel occupancy probabilities, which leads to a slightly higher visibility probability $P(V_v)$. While this does not seem to be a significant problem, the effects of the lower threshold p_{min} appear more serious, which can cause an underestimation of a voxel's visibility. Since visibility only decreases along a measurement ray, a continuous underestimation can accumulate to a significant visibility drop, which limits the range of our mapping method.

We hence introduce a new threshold $q_{max} \leq p_{max}$, and clamp all visibility probabilities $P(V_v) \geq q_{max}$ to 1. This means that the visibility is over- instead of underestimated. Hence, q_{max} can also be used to influence the mapping range. We further introduce a lower threshold for the occupancy probability $q_{min} > 0$, and stop the processing of a



		Hit	
Visible	Occupied	T	F
F	F	$P(H\|\neg V_v)$	$P(\neg H\|\neg V_v)$
F	T	$P(H\|\neg V_v)$	$P(\neg H\|\neg V_v)$
T	F	$P(H\|\neg O_v, V_v)$	$P(\neg H\|\neg O_v, V_v)$
T	T	$P(H\|O_v, V_v)$	$P(\neg H\|O_v, V_v)$

Figure 5.6: Bayesian network of new probability integration scheme.

ray once the visibility probability drops below q_{min}. This allows us to save computing resources that would have otherwise been wasted on computing updates for almost invisible voxels.

5.3.2 Occupancy Probability Integration

To respect voxel visibility during map updates, we need a new probability integration scheme that incorporates the probability $P(V_v)$ of a voxel v being visible. The method that has been chosen for this task can be expressed as a Bayesian network, as shown in Figure 5.6. This network contains three random variables for the events that a voxel v is *visible*, that the received measurement was a *hit* (*i.e.* an occupied measurement), and that the voxel is actually *occupied*.

Let us have a closer look at the probability table for measuring a hit, as indicated by event H. For the case of voxel v not being visible, the probabilities of receiving a hit $P(H|\neg V_v)$ or a miss $P(\neg H|\neg V_v)$ do not depend on the voxel's occupancy. Instead, in the case of v not being visible, the hit and miss probabilities are solely determined by the *a priori* probabilities for measuring a hit or a miss for non-visible voxels. Hence, no matter what measurements we receive for non-visible voxels, we will gain no information on their occupancy status.

For cases where a voxel is visible, on the other hand, the hit probability depends on the voxel's occupancy. If a voxel is occupied and visible, the probability of a hit is $P(H|O_v, V_v)$. Similarly, the probability of a hit is set to $P(H|\neg O_v, V_v)$ if the voxel is visible but not occupied. Both, $P(H|O_v, V_v)$ and $P(H|\neg O_v, V_v)$, are assumed to be constant and configured according to the sensor properties. The probabilities of measuring a miss are complementary, and hence do not need to be configured.

We can solve the Bayesian network from Figure 5.6 for the occupancy probability.

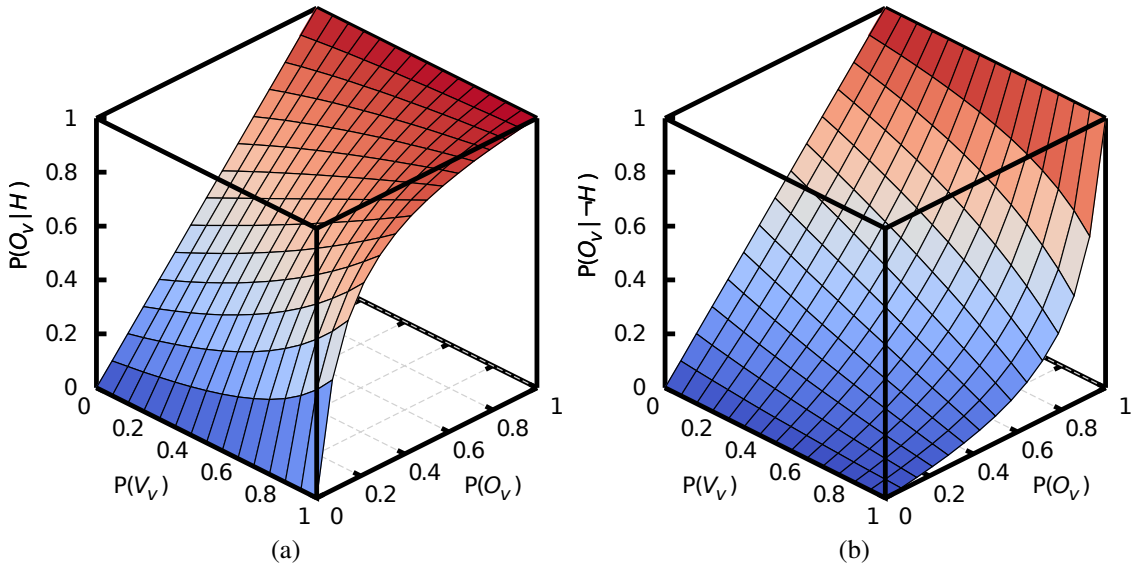


Figure 5.7: Visualization of new update equation for (a) hit- and (b) miss-measurements.

For this task, we use the previous occupancy probability $\mathrm{P}(O_v)$ as prior, and calculate the posterior probability $\mathrm{P}(O_v|M)$. Here, M is the current measurement, which is either a hit H or a miss $\neg H$. The new occupancy probability can be determined by applying Bayes Theorem:

$$\mathrm{P}(O_v|M) = \frac{\mathrm{P}(M|O_v)\,\mathrm{P}(O_v)}{\mathrm{P}(M)}. \tag{5.4}$$

The probabilities $\mathrm{P}(M|O_v)$ and $\mathrm{P}(M)$ that appear in the above equation can be determined by using the law of total probability. This leads to the following solutions:

$$\mathrm{P}(M|O_v) = \mathrm{P}(M|\neg V_v)\,\mathrm{P}(\neg V_v) + \mathrm{P}(M|O_v,V_v)\,\mathrm{P}(V_v), \tag{5.5}$$

$$\begin{aligned}\mathrm{P}(M) = {} & \mathrm{P}(M|\neg V_v)\,\mathrm{P}(\neg V_v) + \mathrm{P}(M|O_v,V_v)\,\mathrm{P}(O_v)\,\mathrm{P}(V_v) \\ & + \mathrm{P}(M|\neg O_v,V_v)\,\mathrm{P}(\neg O_v)\,\mathrm{P}(V_v).\end{aligned} \tag{5.6}$$

We insert Equations 5.5 and 5.6 into Equation 5.4, and arrive at our final update formula:

$$\mathrm{P}(O_v|M) = \frac{\mathrm{P}(O_v)\,[\mathrm{P}(M|\neg V_v)\,\mathrm{P}(\neg V_v) + \mathrm{P}(M|O_v,V_v)\,\mathrm{P}(V_v)]}{\begin{aligned}&\mathrm{P}(M|\neg V_v)\,\mathrm{P}(\neg V_v) + \mathrm{P}(M|O_v,V_v)\,\mathrm{P}(O_v)\,\mathrm{P}(V_v) \\ &+ \mathrm{P}(M|\neg O_v,V_v)\,\mathrm{P}(\neg O_v)\,\mathrm{P}(V_v)\end{aligned}}. \tag{5.7}$$

The solution of this new update equation in relation to the visibility probability $\mathrm{P}(V_v)$ and the previous occupancy probability $\mathrm{P}(O_v)$ is plotted in Figure 5.7a for a hit, and in Figure 5.7b for a miss measurement. For both figures, the following parameters have been used:

$$\mathrm{P}(H|O_v,V_v)=0.9;\ \ \mathrm{P}(H|\neg O_v,V_v)=0.1;\ \ \mathrm{P}(H|V_v)=0.5. \tag{5.8}$$

Please note that these parameters have been chosen for visualizing the shape of the probability distribution, and that more conservative parameters are used during our performance evaluation.

Our new update equation is considerably more complex than the simple recursive Bayes filter that is applied by OctoMap. Furthermore, while OctoMap's update equation can be expressed using log-odds, which allows the equation to be simplified to a simple summation, this is not possible for Equation 5.7. This makes our approach significantly more computationally expensive. Fortunately however, the probability integration is not particularly performance critical, as we see later in this chapter.

5.3.3 Sensor Depth Error Modeling

For a stereo camera, the range measurement error increases quadratically with the measured depth, which is also the case for other camera based depth sensors such as the original Microsoft Kinect. Hence, only integrating a hit for the single voxel containing the measurement end point p_e is particularly incorrect for distant points. Therefore, we attempt to respect the expected measurement error during our probability update.

This means that we first need an estimate for the average depth error, which we assume to be normally distributed. Given the depth measurement z, the standard deviation of the depth measurement σ_z can be computed as follows:

$$\sigma_z=\kappa\cdot z^2,\ \text{with}\ \ \kappa=\frac{\sigma_d}{b\cdot f}. \tag{5.9}$$

Here, κ is a constant factor that depends on the sensor properties. For a stereo camera, it can be computed from the disparity standard deviation σ_d, baseline distance b and focal length f. When using sensors with a non-quadratically increasing depth error, Equation 5.9 can be replaced by a different equation that models the expected sensor behavior.

With σ_z, we know the accuracy of the current range measurement at depth z. We can hence use σ_z to determine the probability that a voxel v on the current measurement ray is entirely or partly inside the obstacle that corresponds to the measurement end point p_e. The event of voxel v being inside this obstacle is denoted as I_v, and its probability is $\mathrm{P}(I_v)$. For determining $\mathrm{P}(I_v)$, we require the distribution function $f_z(z_t)$ for depth measurements at depth z. Since we assume a normally distributed depth error, $f_z(z_t)$ is a normal distribution function with expectation z and standard deviation σ_z. We can hence approximate $\mathrm{P}(I_v)$ with the help of the cumulative distribution function $F_z(z_t)$ of $f_z(z_t)$ (see Andrews, 1992):

$$F_z(z_t)=\frac{1}{2}\left(1+\mathrm{erf}\left(\frac{z_t-z}{\sigma_z\sqrt{2}}\right)\right), \tag{5.10}$$

where $\mathrm{erf}(x)$ is the Gauss error function (see Andrews, 1992).

For a depth z_t, the cumulative distribution function $F_z(z_t)$ provides the probability that the true depth of the detected obstacle is smaller than z_t. To determine the probability $\mathrm{P}(I_v)$ that voxel v is fully or partially inside this obstacle, we hence reduce v to a point $p_v = (x_v, y_v, z_v)$, which is located at the voxel's center. We can then approximate $\mathrm{P}(I_v)$ as follows:

$$\mathrm{P}(I_v) \approx F_z(z_v). \tag{5.11}$$

The knowledge of how likely v is located inside the detected obstacle can be incorporated into our map update scheme. If $\mathrm{P}(I_v) \notin \{0,1\}$, then the update that should be performed on the occupancy probability of v is an intermediate between the update for a miss $\mathrm{P}(O_v|\neg H)$ and the update for a hit $\mathrm{P}(O_v|H)$. In our case, we use a linear interpolation between the results of both update types as approximation for the new occupancy probability $\mathrm{P}(O_v|M)$. As interpolation factor we chose $\lambda = \mathrm{P}(I_v)$, and hence arrive at the following formula:

$$\mathrm{P}(O_v|M) \approx \lambda\,\mathrm{P}(O_v|H) + (1-\lambda)\,\mathrm{P}(O_v|\neg H), \text{ with } \lambda = \mathrm{P}(I_v). \tag{5.12}$$

For this method to work effectively, we need to extend OctoMap's original ray casting scheme. We continue to traverse voxels in ray direction, even after we have already reached the measurement end point p_e. Ray casting is stopped once $\mathrm{P}(I_v)$ is close to 1. However, to make sure that at least one full hit is integrated, we continue ray casting for one further voxel with $\mathrm{P}(O_v|M) = \mathrm{P}(O_v|H)$.

5.3.4 Performance Considerations and Optimizations

Although our method is considerably more complex than OctoMap, the performance impact remains limited. One reason for this circumstance is that probability integration only makes up for a small fraction of the total processing time spent by OctoMap. This is due to the fact that no matter how many observations are received for a voxel v, OctoMap only performs one update of v's occupancy probability. For this purpose, OctoMap employs an update reduction step after ray casting, in which preference is given to hit- rather than miss-updates

We can hence delay the computation of our visibility estimate and our new update formula, until the updates have been reduced to one update per voxel. We are, however, required to extend OctoMap's update reduction scheme. With the introduced depth error consideration, we no longer receive hit- and miss-updates only, but instead our updates are an interpolation between those two extremes. Thus, instead of giving preference to hit-updates, we modify the update reduction to give preference to updates with a large interpolation factor λ.

Furthermore, we perform an optimization of the update reduction method. OctoMap employs a hash table for this task, in which it inserts the updates that are generated by

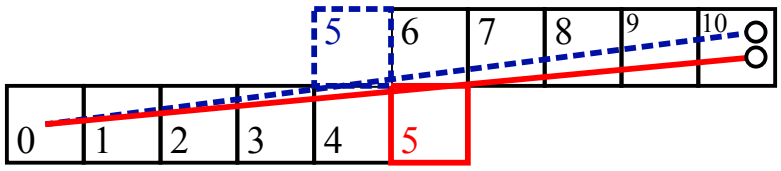


Figure 5.8: Voxels traversed by two rays for neighboring end points.

ray casting. For each new update, a look-up in the hash table is performed, to determine whether an update for this voxel already exists. If this is the case, preference is given to hit-updates, or in our case updates with large λ. For dense measurements, such as received from dense stereo matching, a high number of rays have to be processed. Hence, despite the constant time complexity of the used hash table, the time required for a single voxel look-up has a critical performance impact.

We can speed-up the voxel look-up by exploiting the fact that the rays generated from stereo matching are sorted. Neighboring rays, which originate from neighboring pixels in the disparity map, usually progress within close proximity to one another. This means that two neighboring rays traverse mostly the same voxels, as shown for one example in Figure 5.8. In this figure, the voxels traversed by both rays are identical except for the two voxels with index 5.

In such cases, we can perform voxel look-ups against the previous ray. For each voxel, we thus compare whether it matches the voxel from the previous ray with the same index. Only if this comparison is not true, then we perform a more expensive look-up that uses OctoMap's hash table. This update reduction strategy is described in Algorithm 5.1 for the simplified case without giving preference to large λ. In the case of dense measurements such as received from stereo vision, this method allows us to perform most voxel look-ups without accessing the hash table.

Finally, we can speed-up the computation of the linear-interpolation factor λ, which matches the probability $\mathrm{P}(I_v)$ that v is inside the detected obstacle. Since $\mathrm{P}(I_v)$ needs to be determined for each processed voxel, its calculation can result in a severe increase of the overall processing time. However, for cases where the considered voxel v is far from the measurement end point p_e, the probability $\mathrm{P}(I_v)$ is close to 0. This means that we can neglect the effects of depth uncertainty, and instead keep the original update strategy from Equation 5.7 with $M = \neg H$. Only for voxels close to the measurement end point p_e, we have to compute $\mathrm{P}(I_v)$ and perform the linear update interpolation from Equation 5.12.

We can further accelerate the calculation of $\mathrm{P}(I_v)$ through pre-computation. The value of $\mathrm{P}(I_v)$ depends on the currently processed depth measurement z and the Euclidean distance $\Delta_v = \|p_e - p_v\|$ between the centroid of the current voxel p_v and the measurement end point p_e. Hence, we precompute $\mathrm{P}(I_v)$ for a discretized set of z and Δ_v, and store the results in a two-dimensional look-up table. Thus, we require only a single memory-lookup operation to determine the value for $\mathrm{P}(I_v)$.

Algorithm 5.1: Simplified voxel look-up algorithm that exploits ray proximity.

```
previousRay := ∅;
foreach currentRay ∈ allRays do
    /* Look-up against previous ray.                                  */
    for i := 1 to min(length(currentRay), length(previousRay)) do
        if currentRay[i] ≠ previousRay[i] then
            /* Voxel not found in previous ray.                       */
            if hashMap.find(currentRay[i]) = ∅ then
                /* Voxel not found in hash map.                       */
                scheduleUpdate(currentRay[i]);
            end
        end
    end
    /* Look-up remaining voxels against hash map.                     */
    if length(currentRay) > length(previousRay) then
        for i := length(previousRay) + 1 to length(currentRay) do
            if hashMap.find(currentRay[i]) = ∅ then
                /* Voxel not found in hash map.                       */
                scheduleUpdate(currentRay[i]);
            end
        end
    end
    /* Swap buffers for previous and current ray.                     */
    swap(currentRay, previousRay);
end
```

5.4 Evaluation

For evaluating our new occupancy mapping method, we use a dataset from the rawseeds project (Bonarini *et al.*, 2006; Ceriani *et al.*, 2009). In particular, we use the dataset *Bicocca_2009-02-25b*, which is publicly available online (see Raw Seeds Project, 2009). This dataset comprises a 29 min indoor recording, which was performed with a mobile robot that traversed a trajectory of about 774 m. The robot was equipped with various sensors, including laser scanners and a trinocular stereo system. The cameras of this stereo system had a resolution of 640×480 pixels, and each camera recorded approximately 26.000 fames. An example for a left camera frame from this dataset is shown in Figure 5.9a.

For stereo matching, we use the previously mentioned ELAS algorithm by Geiger *et al.* (2011), which is applied to the left and right camera image of the trinocular stereo system. For this stereo method, the authors have made their implementation available online (see Geiger *et al.*, 2013), which is used in this evaluation. This method was

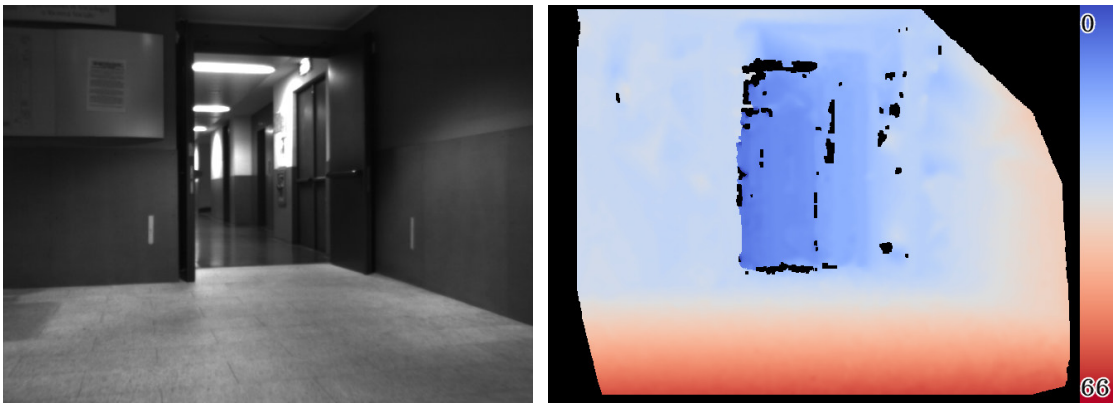


Figure 5.9: (a) Left input image and (b) corresponding disparity map created by ELAS. The color scale corresponds to the disparity in pixels.

selected for its fast processing rate and accurate matching results. An example for the disparity map created for a scene from the evaluation dataset is shown in Figure 5.9b. In this figure, red and blue hues represent large and small disparities, while black represents regions with no disparity estimate.

For time critical applications, ELAS can produce disparity maps with only half the size of an input image. Despite the smaller dimensions of the disparity map in this mode, the disparity range is still processed at full resolution. Hence, while the lateral resolution is reduced in this case, the resolution of the depth measurements is preserved. This approach was extended for this thesis, in order to produce quarter resolution disparity maps at an even faster processing rate. All three resolution options are included in this evaluation.

In addition to range sensing, we further require a method for localizing the current robot pose, in order to evaluate our mapping method. Several estimates have been published for the robot poses in the considered dataset. We use the solution provided by Ruhnke (2009), which has been obtained using the graph-based laser SLAM methods from Grisetti *et al.* (2007, 2008). To overcome the low update rate of the provided pose estimates, we perform a cubic spline interpolation of the robot poses.

5.4.1 Map Quality Analysis

ELAS was run with the standard parameter set that is provided for robotics applications, except for the maximum disparity d_{max}, which was set to 96 pixels. For the disparity standard deviation σ_d, a value of 0.3 pixels was assumed. With the stereo matching results delivered by ELAS and the selected pose estimates, we are able to create a volumetric occupancy map of the environment that the robot traversed during the recording of the test dataset.

Table 5.1: Parameters selected for the proposed method and OctoMap.

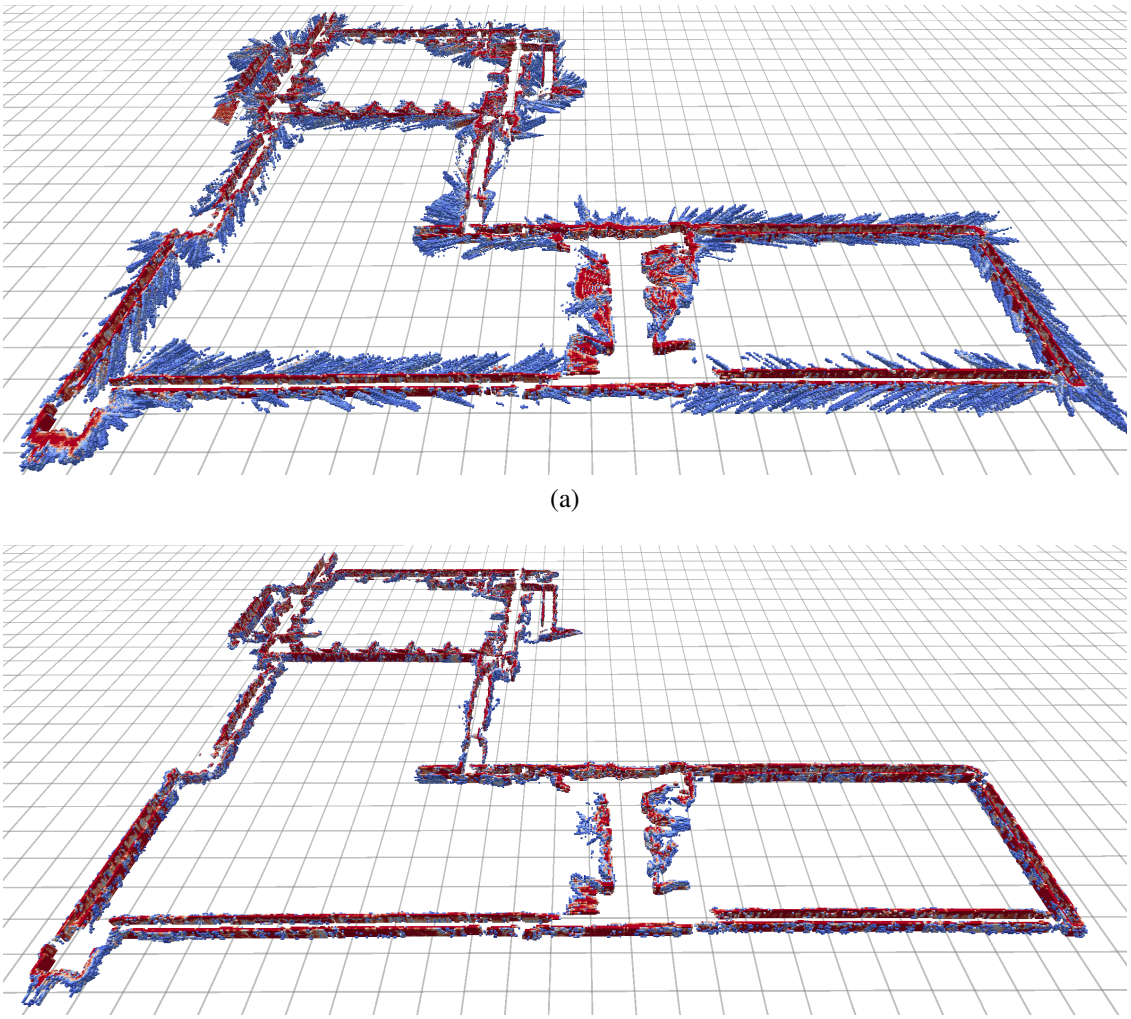
Proposed Method		**OctoMap**	
$P(H\|V_v, \neg O_v)$	$= 0.43$	$P(O_v\|\neg H)$	$= 0.45$
$P(H\|V_v, O_v)$	$= 0.55$	$P(O_v\|H)$	$= 0.55$
$P(H\|\neg V_v)$	$= 0.05$		
$P(V_{v_i}\|C_{v_i}, V_{v_{i-1}})$	$= 0.20$		
$P(V_{v_i}\|\neg C_{v_i}, V_{v_{i-1}})$	$= 1.00$		
$\{q_{min}, q_{max}\}$	$= \{0.1, 0.7\}$		

Two such maps were created, of which one was obtained with OctoMap 1.6.0, and one with the mapping method proposed in this chapter. Since both methods use a different probability integration scheme, each method requires its own set of parameters. The parameters that have been selected for each method are shown in Table 5.1. Having a different parameterization for each method unfortunately limits the comparability of both approaches, as the obtained results are only valid for one particular parameter selection. However, for each method the parameters have been adjusted in order to achieve accurate mapping results. Hence, we expect that the received results are representative for the general performance of each method.

Figure 5.10a and 5.10b show the maps that result when using the half-resolution version of ELAS with OctoMap and with our mapping method respectively. In both figures, red hues indicate high occupancy probabilities, while dark blue indicates probabilities just above 0.5. For the creation of both maps, a voxel size of 0.2 m has been used. Furthermore, all voxels below and above a minimum and maximum height have been cut-off, which effectively removes the floor and ceiling of the mapped indoor environment.

One can clearly see that the map created by OctoMap contains a high number of erroneous artifacts, which are not visible in the map obtained with our approach. A close-up view on the maps for a corridor in this dataset is shown in Figure 5.11a and 5.11b. These maps have been created with OctoMap and our method, and the full resolution version ELAS. Even though the erroneous artifacts are mostly removed in our results, the wall is still densely mapped. Particularly when mapping neighboring rooms or intersecting corridors, the produced artifacts can lead to a disruption of previously correct map areas. For comparison, Figure 5.11c and 5.11d contain the maps for the same corridor with half and quarter resolution ELAS and our mapping approach. For both cases, the corridor is still densely mapped despite the smaller image resolutions.

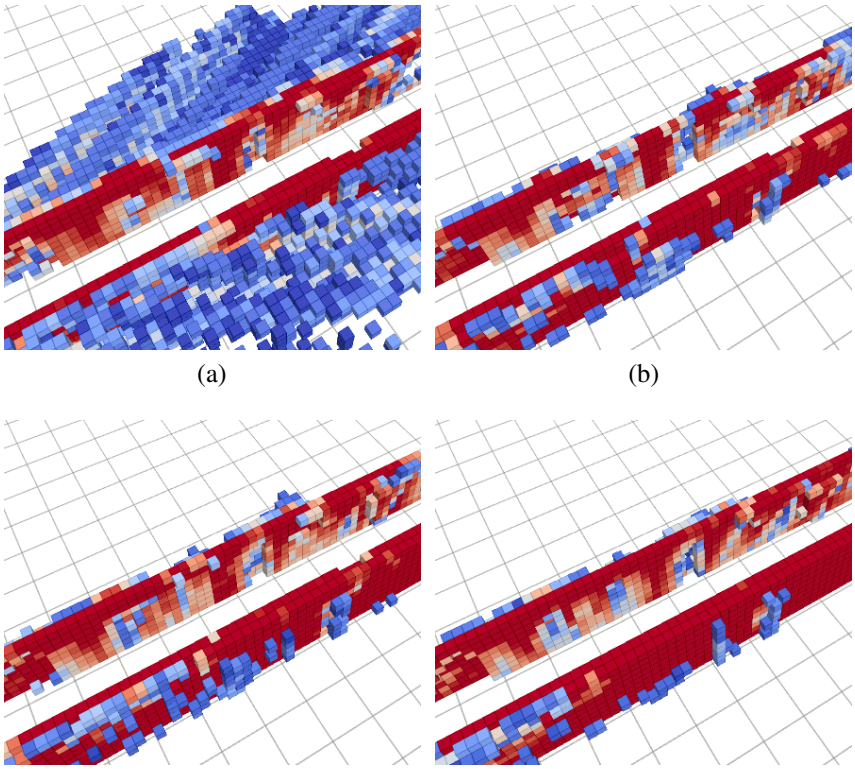
The mapping behavior of our method differs significantly from OctoMap. When facing in a direction that has previously not been observed, OctoMap immediately expands its map to the maximum visible distance. Our method, however, gradually increases the mapped distance after each sensor update. This behavior has been analyzed for the parameters from Table 5.1 and different voxel sizes. The received results are shown in Figure 5.12.



(a)

(b)

Figure 5.10: Full map of evaluation dataset, created with (a) OctoMap and (b) our method for half resolution ELAS.



(a) (b) (c) (d)

Figure 5.11: Corridor mapped with (a-b) full, (c) half and (d) quarter resolution ELAS and (a) OctoMap and (b-d) our method.

For this analysis, one stereo pair of the evaluation sequence was repeatedly processed, in which the cameras are facing a long corridor. After each processing iteration, the distance to the farthest voxel with an occupancy probability $P(O_v) > 0.5$ was measured. With a voxel size of 0.2 m, 65 updates were required to reach a distance of approximately 10 m. For our test dataset with a frame rate of 15 Hz, 65 updates is equivalent to a time span of 4.3 s. This time span strongly depends on the voxel size, q_{max} and $P(V_{v_i}|V_{v_{i-1}}, C)$. Hence, by adjusting these parameters, one can chose a compromise between the speed of map expansion and map quality.

5.4.2 Run-Time Performance Evaluation

In addition to the map quality assessment, the run-time performance has been analyzed on a commodity PC with a 3.3 GHz Intel i5 dual core CPU. On this computer, a one-minute section of the evaluation dataset was processed with varying voxel sizes and the three different resolutions of ELAS. Figure 5.13a shows the average processing times that have been observed in this experiment, when using our mapping method and OctoMap.

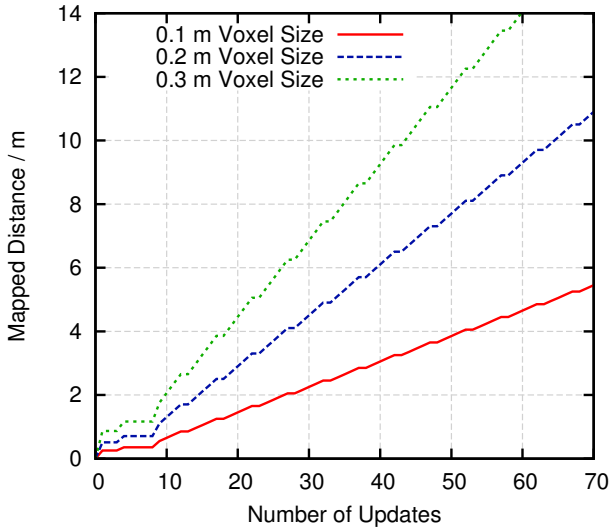


Figure 5.12: Growth of mapped distance for different voxel sizes.

Please note that this diagram uses a logarithmic scale for the processing time, and that the time for stereo matching has not been measured.

The diagram reveals that except for small voxel sizes and the quarter resolution version of ELAS, our method provides significantly lower processing times. The largest performance difference was observed for full-resolution ELAS with 0.2 m voxel size, where our method required only 66% of the average computation time that was spent by OctoMap. This result might be surprising, given that our method is more complex, but it can be explained with the optimizations performed in Section 5.3.4 on page 113. Our optimized voxel look-up works particularly well if neighboring rays traverse almost the same voxels, which is the case for high sensor resolutions and / or large voxel sizes. Hence, we observe high speed-ups in these cases.

For our mapping method, a processing time of 48 ms was received when using half-resolution ELAS and a voxel size of 0.2 m. This should be fast enough to facilitate real-time processing of the used test dataset, *i.e.* 15 frames per second. However, we also need to account for the processing time that is required for stereo matching. For the full, half and quarter resolution versions of ELAS, we require an average processing time of 122 ms, 48 ms and 24 ms respectively. Hence, for half- and quarter-resolution ELAS, real-time processing of the evaluation dataset is possible, if stereo matching and occupancy mapping are run in parallel on both CPU cores.

Since OctoMap tends to map many erroneous artifacts with our stereo matching results, it also requires more memory. The memory consumptions that were measured for the previously examined one minute test run are shown Figure 5.13b, which was again plotted using a logarithmic scale. On average over all test runs, our method required only 37% of the memory consumed by OctoMap. The largest difference was observed

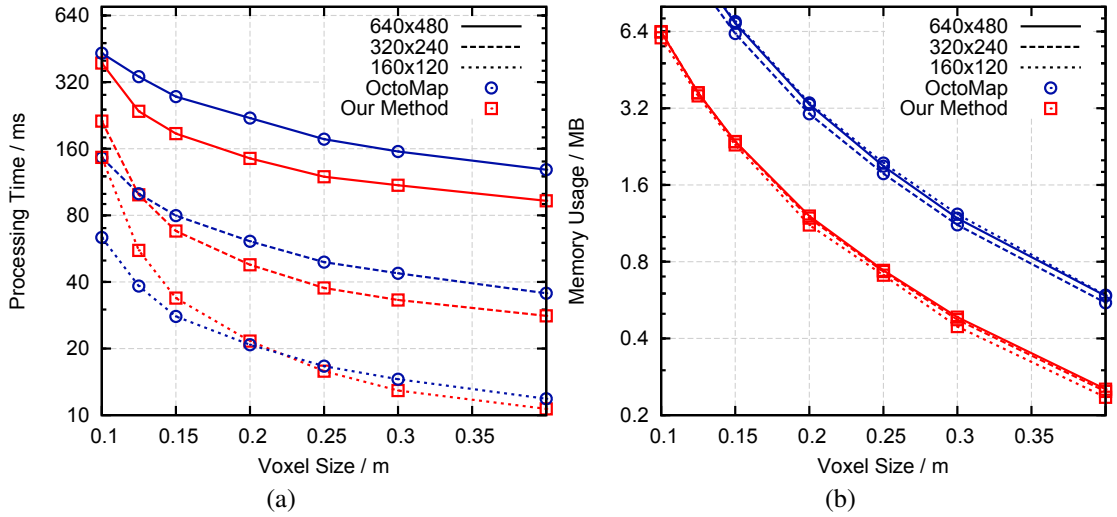


Figure 5.13: (a) Processing times and (b) memory consumption that have been measured for our method and OctoMap.

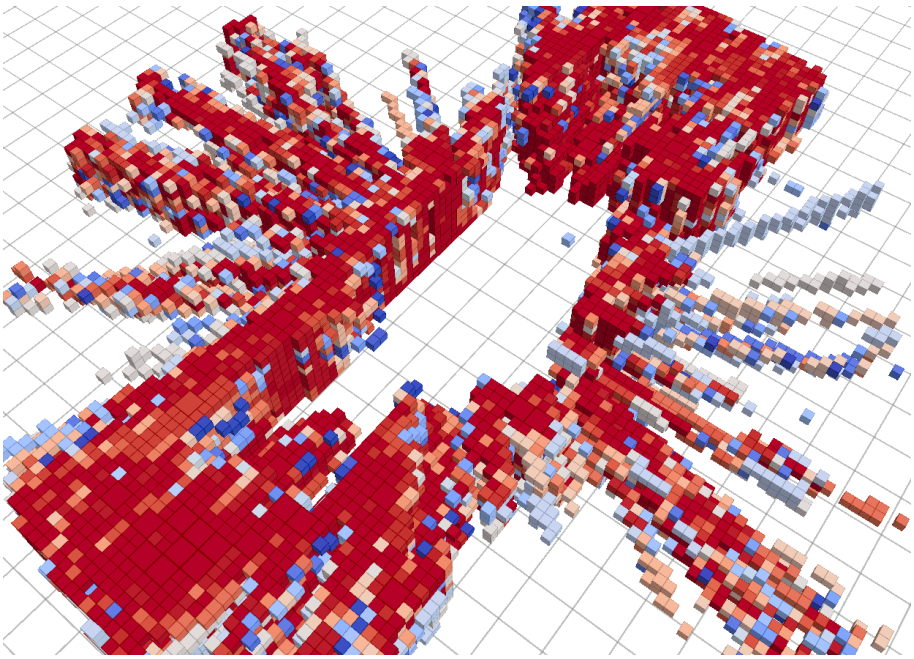
for quarter-resolution ELAS with 0.1 m voxel size. In this case, the map generated by our method required 6.0 MB of memory, which is only 32% of the 19.1 MB required by OctoMap. For a map of the full dataset that has been created with half-resolution ELAS, our method required 24.3 MB of memory, which is only 37% of the 65.7 MB consumed by OctoMap. Figure 5.13b also reveals that the resolution of ELAS only has a marginal impact on the memory consumption.

The reason for the poor performance of OctoMap in this experiment is the unstructured appearance of the mapped artifacts. In this case, OctoMap's octree compression cannot be applied effectively. Hence, the memory overhead caused by the artifacts exceeds the memory required for the actual map.

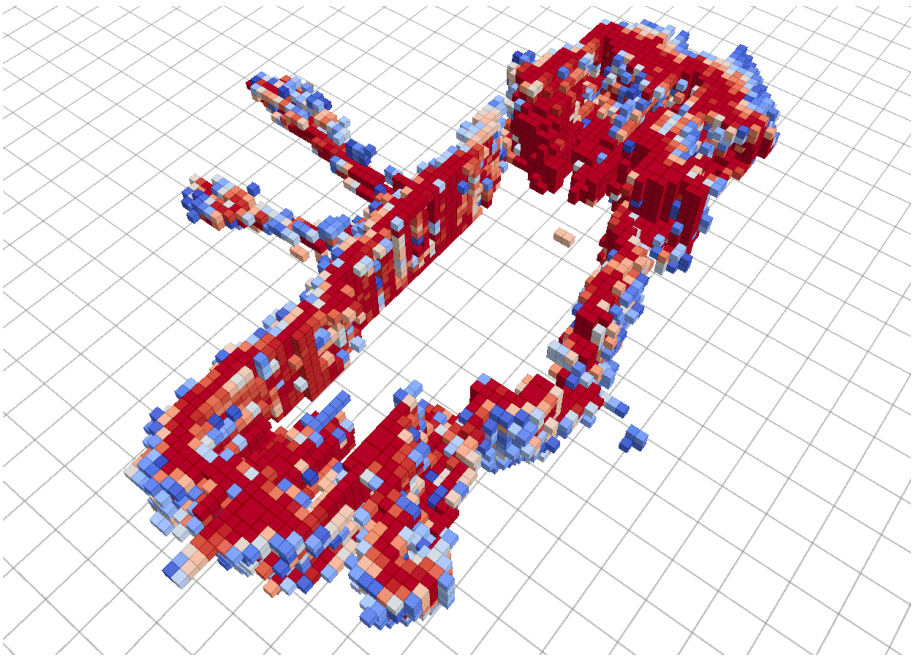
5.4.3 Mapping of MAV Environment

We have shown that our occupancy mapping system is fast enough for real-time processing, when run on the considered PC with an Intel i5 dual core CPU. Our MAV platform, however, is equipped with an Intel Core 2 Duo CPU, which is significantly less powerful. Furthermore, we require estimates of the current camera pose for running our occupancy mapping method. While we presented a system for on-board pose estimation with our MAV in Chapter 4, this software system already consumes a large part of the available processing resources.

Thus, it is not possible to run our mapping method in real-time on-board of our MAV. We can, however, use our MAV for sequence recording, and then process these stereo sequences offline. Hence, once the MAV has landed after an autonomous flight, we can either run our mapping method on the MAV's on-board computer, or transfer the



(a)



(b)

Figure 5.14: Maps created off-board from imagery recorded with our MAV and (a) OctoMap and (b) our method.

recorded imagery to a ground computer for off-board processing.

To test the performance of our mapping method when applied to data recorded with our MAV, it was used to process the imagery recorded during the autonomous 360° yaw rotation flight from Section 4.4.7 on page 94. This flight was performed in an indoor environment that contained many visible objects, such as tables, chairs, shelves, cabinets, computers, or other robots. This cluttered environment poses a significant challenge for any vision-based mapping system. What makes this flight even more challenging for occupancy mapping is the fact that the MAV remained mostly at the same hovering location. This means that many of the visible objects are only ever observed from one angle at a large distance.

The sequence was processed with OctoMap and our method and half-resolution ELAS, while using the parameters of our previous experiments from Table 5.1 on page 117. As pose estimates served the estimates that were computed on-board during the autonomous flight, which have been recorded by the MAV. The maps that were created with each mapping method are shown in Figure 5.14a and 5.14b.

Similar to our previous experiments, the map received with OctoMap shows a high number of erroneous artifacts in all directions. While most artifacts have disappeared in the map from our mapping system, two streak-like artifacts remain in the left section of the map. These artifacts are caused by a cabinet in the indoor environment, whose glass doors reflect the scene behind the MAV. Hence, stereo matching tends to match the reflections, and thus registers large depth values at this place. Such systematic errors can, unfortunately, not be corrected by a mapping method alone. For most other parts of this environment, however, our method provides a clearer map with much less artifacts.

5.5 Summary and Discussion

In this chapter we have introduced a new method for volumetric occupancy mapping. Unlike most existing methods in this area, our method has been specifically designed for use with stereo vision. Compared to laser scanners, which are commonly used for occupancy mapping, the data received from a stereo vision system contains a significantly higher measurement noise. Furthermore, this noise tends to be spatially and temporally correlated, which makes the processing of this data even more challenging. We have shown in our evaluation that this circumstance can lead to the mapping of many erroneous artifacts, when using a common occupancy mapping method designed for laser range measurements.

Hence, a new method has been designed that is based on the well-known OctoMap system. OctoMap was extended with a new probability integration scheme, which respects the probability that a voxel is currently visible. If we are certain that a given voxel is not visible, then all measurements received for this voxel must be erroneous and should be ignored. If we are certain that the voxel is visible, on the other hand, then the probability integration should match the one from an unmodified OctoMap. Usually, however, we

do not have absolute certainty on a voxel's visibility. Hence, the general behavior will be in between those two extremes.

For estimating the visibility probability of a voxel, we traverse the corresponding measurement ray. For each voxel on the ray, we determine the probability that this voxel is locally occluded by its direct neighbors. The visibility probability is then computed depending on the local occlusion probability, and the probability that the previous voxel on the ray that is closer to the sensor origin is visible.

Furthermore, we model the sensor depth error and consider it when updating the occupancy probabilities. The expected depth error for measurements from stereo vision increases quadratically with the measured depth. Hence, it is important that we consider the expected measurement error when updating the occupancy map. This is done by a gradual transition from the updates for miss-measurements to the updates for hit-measurements. Thus, not only one voxel receives a hit-update, but rather a group of voxels within the uncertainty range.

In our evaluation it was shown that this new occupancy mapping method effectively removes the artifacts caused by the noisy stereo vision data. At the same time, the map remains dense for the correctly observed regions. Due to the removal of the map artifacts, our method also requires less memory. In fact, our method consumed as little as 32% of the memory allocated by OctoMap. Furthermore, our method achieved lower processing times than OctoMap for most of the performed test runs, despite the fact that our method is significantly more complex. This can mainly be credited to an optimization of OctoMap's update reduction. In principle, this optimization could also be ported to OctoMap, however, a performance benefit should only be observable for dense measurements such as provided by a dense stereo algorithm.

Our resulting system is fast enough to enable real-time mapping and stereo matching, when using both cores of a commodity PC with an Intel i5 dual-core CPU. The Intel Core 2 Duo CPU available on our quadrotor MAV is unfortunately not powerful enough for this task. However, we are able to create a map offline, after the MAV has finished its autonomous flight. The next generation of MAVs might already provide sufficient computing resources for on-board occupancy mapping. At the time of writing, powerful embedded computers in the COM-Express Compact from factor (95×95 mm) are already available, which feature an Intel i7 quad-core CPU (*e.g.* see American Portwell Technology, Inc., 2013). With such hardware, it should be possible to simultaneously run dense stereo matching, our occupancy mapping method and our visual navigation system from Chapter 4 on-board our MAV in real-time.

Finally, even though the method presented in this chapter was specifically designed for use with stereo vision, it can also be applied to other range sensors. In cases where the range measurements exhibit a high measurement noise, our method should deliver more robust results. Hence the presented method might also be useful for improving the maps created with laser scanners, sonars or RGB-D cameras.

Chapter 6

Summary and Conclusions

In this thesis we have investigated the challenges involved in designing an autonomous MAV that uses stereo vision as its primary sensor technology. Compared to laser scanners, which are commonly deployed on wheeled robots, cameras offer the advantage of lower weight and power consumption. In particular laser scanners with multiple beams that obtain measurements from several sensing-planes are expensive, heavy and require much energy. It is thus tempting to rely on cameras for a payload and energy constrained MAV.

Compared to a monocular camera, a stereo camera pair offers the advantage of depth perception. This allows us to reconstruct the metric 3D position for a point that is observed by both cameras. Hence, a stereo camera is a rich sensor, which in principle offers an extensive three-dimensional perception of the surrounding environment.

The construction of an autonomous MAV that primarily relies on stereo vision requires us to find solutions for several problems. The first problem is of course stereo matching, which allows us to reconstruct the 3D position of points that are observed by both cameras. Next, the MAV has to be able to determine its current pose, *i.e.* its 3D position and orientation. Finally, the MAV has to be able to map its environment, in order to determine which space is traversable and which one is not. In this thesis, we have examined solutions to all of these three problems.

In Chapter 3, we introduced a new and efficient stereo matching method. Unlike most current research on stereo vision this method is *sparse*, which means that it only delivers matching results for a small set of salient image features. This enables the method to achieve very high processing rates, which is crucial if we want to use the stereo matching results to facilitate fast and responsive control of our MAV. The accuracy of the stereo method is improved by a new feature detector, which was specifically designed for stereo matching.

Despite our new stereo matching method being sparse, it densely examines the valid disparity range in the opposite matching direction for each found stereo correspondence. This allows us to identify features that received non-consistent matching results, or whose matching results are not sufficiently unique. Once these features have been removed, the remaining features show a high matching accuracy. We further proposed a fast method for performing this dense consistency and uniqueness check on real camera

images, without the need for prior image rectification.

This fast and accurate stereo matching method was used in Chapter 4 for tracking the pose of our MAV. The first method proposed in this chapter is based on a visual SLAM algorithm that processes both image and depth information. Because this SLAM method relies on a sparse set of image features, it integrates well with our sparse stereo matching system. To meet the performance requirements for our MAV, the SLAM method was simplified such that it only retains a small local map. Using this *local SLAM* method, an autonomous MAV was constructed that relies on a forward-facing stereo camera pair and an IMU as only sensors. The MAV demonstrated its control capabilities in an autonomous flight experiment.

The second method that was proposed in Chapter 4 relies on a downward-facing stereo camera pair. For this approach it was assumed that the ground is flat and level, which is a valid assumption when flying in man-made indoor environments. The ground plane is detected by fitting a plane to the 3D points received from stereo matching. From this plane, it is then possible to extract the MAV's current height, and its roll and pitch angles. Horizontal translations and yaw rotations are observed by using another method, which is based on frame-to-frame tracking.

With this method we hence receive a full 6DoF pose estimate that can be used as an alternative to the estimate obtained by local SLAM. Both pose estimation methods were integrated on one MAV that has been equipped with two stereo camera pairs. The two redundant pose estimates are fused using an EKF. The resulting MAV was successfully evaluated in several flight and offline-processing experiments. Compared to the first autonomous MAV prototype, this MAV exhibits a more robust and more precise pose estimation, which improves the quality of the autonomous flight.

The problem of environment perception was approached in Chapter 5, where we introduced a new method for volumetric occupancy mapping. This method is based on the popular OctoMap approach, which creates voxel-based maps that are stored in octrees. For each voxel, the map stores the probability that this voxel is occupied by an obstacle. While OctoMap has shown to provide good results when used with measurements from laser scanners, we have demonstrated that this is not the case for dense measurements from a stereo vision system.

We thus introduced an extension of OctoMap, which considers the visibility of a voxel when updating the voxel's occupancy probability. Furthermore, the depth error of a stereo vision system is modeled and considered during the map update procedure. Despite the higher complexity of this method, it achieved shorter processing times in most of the conducted performance measurements. This result can be credited to an optimization of OctoMap's original ray casting scheme.

The occupancy mapping method is the only presented technique in this thesis that cannot be run in real-time on-board of our MAV. However, the method is already fast enough for real-time processing on a commodity PC, including dense stereo matching. We expect that the next generation of MAVs will feature sufficient processing resources for running both, our volumetric occupancy mapping method and the pose estimation

from Chapter 4, on-board and in real time. Until then, this method can be used for generating occupancy maps offline, once the MAV has finished an autonomous flight.

In this thesis we have covered a broad set of problems that appear when designing a stereo vision based autonomous MAV. Nevertheless, further problems remain to be solved before we will see the first practical applications of such MAVs outside of a laboratory environment. One important topic that we have not covered is autonomous path planning. The map created with our occupancy mapping method will have to be used by the MAV to plan a collision-free and safe trajectory to its designated target location. The MAV should further be able to make autonomous decisions, which however, strongly depend on the intended application. Furthermore, to guarantee safe and fail-proof operation, more redundancies are required for pose estimation and environment mapping.

We can conclude that there still remains much work to be done before we can expect to see autonomous MAVs in everyday life. However, I hope that this thesis has made a contribution towards this goal.

Bibliography

Achtelik, M., Zhang, T., Kuhnlenz, K., and Buss, M. (2009). Visual Tracking and Control of a Quadcopter Using a Stereo Camera System and Inertial Sensors. In *International Conference on Mechatronics and Automation (ICMA)*, pages 2863–2869. IEEE.

Achtelik, M., Achtelik, M., Weiss, S., and Siegwart, R. (2011). Onboard IMU and Monocular Vision Based Control for MAVs in Unknown in-and Outdoor Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3056–3063.

Agrawal, M., Konolige, K., and Blas, M. R. (2008). CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching. In *European Conference on Computer Vision (ECCV)*, pages 102–115. Springer.

Amanatides, J. and Woo, A. (1987). A Fast Voxel Traversal Algorithm for Ray Tracing. In *Annual Conference of the European Association for Computer Graphics (Eurographics)*, volume 87, pages 3–10.

Amazon.com, Inc. (2013). Amazon Prime Air. http://amazon.com/b?node=8037720011. Accessed: 07.01.2014.

American Portwell Technology, Inc. (2013). PCOM-B219VG. http://portwell.com/products/detail.asp?CUSTCHAR1=PCOM-B219VG. Accessed: 04.12.2013.

Andrews, L. C. (1992). *Special Functions of Mathematics for Engineers*. SPIE Press, Bellingham.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous Localization and Mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, **13**(3), 108–117.

Bajracharya, M., Ma, J., Howard, A., and Matthies, L. (2012). Real-Time 3D Stereo Mapping in Complex Dynamic Environments. In *International Conference on Robotics and Automation-Semantic Mapping, Perception, and Exploration (SPME) Workshop*.

Baker, H. and Binford, T. (1981). Depth from Edge and Intensity Based Stereo. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 631–636. Morgan Kaufmann Publishers, Inc.

Bay, H., Tuytelaars, T., and Van Gool, L. (2006). SURF: Speeded Up Robust Features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. Springer.

Benhimane, S. and Malis, E. (2004). Real-Time Image-Based Tracking of Planes Using Efficient Second-Order Minimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 943–948.

Bonarini, A., Burgard, W., Fontana, G., Matteucci, M., Sorrenti, G., and Tardos, J. D. (2006). RAWSEEDS: Robotics Advancement through Web-publishing of Sensorial and Elaborated Extensive Data Sets. In *IROS 2006 Workshop on Benchmarks in Robotics Research*.

Bouabdallah, S. (2007). *Design and Control of Quadrotors with Application to Autonomous Flying*. Ph.D. thesis, Ecole Polytechnique Federale de Lausanne.

Bouguet, J.-Y. (2013). Camera Calibration Toolbox for Matlab. http://vision.caltech.edu/bouguetj/calib_doc. Accessed: 09.12.2013.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23**(11), 1222–1239.

Brown, D. C. (1966). Decentering Distortion of Lenses. *Photogrammetric Engineering*, **7**, 444–462.

Bry, A., Bachrach, A., and Roy, N. (2012). State Estimation for Aggressive Flight in GPS-Denied Environments Using Onboard Sensing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8.

Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). BRIEF: Binary Robust Independent Elementary Features. In *European Conference on Computer Vision (ECCV)*, pages 778–792. Springer.

Carrillo, L. R. G., López, A. E. D., Lozano, R., and Pégard, C. (2012). Combining Stereo Vision and Inertial Navigation System for a Quad-Rotor UAV. *Journal of Intelligent & Robotic Systems (JINT)*, **65**(1), 373–387.

Ceriani, S., Fontana, G., Giusti, A., Marzorati, D., Matteucci, M., Migliore, D., Rizzi, D., Sorrenti, D. G., and Taddei, P. (2009). Rawseeds Ground Truth Collection Systems for Indoor Self-Localization and Mapping. *Autonomous Robots*, **27**(4), 353–371.

Chang, C., Chatterjee, S., and Kube, P. R. (1991). On an Analysis of Static Occlusion in Stereo Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 722–723.

Collins, T., Collins, J., and Ryan, C. (2007). Occupancy Grid Mapping: An Empirical Evaluation. In *IEEE Mediterranean Conference on Control & Automation (MED)*, pages 1–6.

Corke, P., Strelow, D., and Singh, S. (2004). Omnidirectional Visual Odometry for a Planetary Rover. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 4, pages 4007–4012.

Crow, F. C. (1984). Summed-area tables for texture mapping. In *Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, volume 18, pages 207–212. ACM.

Datta, A., Kim, J.-S., and Kanade, T. (2009). Accurate Camera Calibration Using Iterative refinement of control points. In *ICCV Computer Vision Workshops*, pages 1201–1208. IEEE.

Davison, A. J. (2003). Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1403–1410.

Devernay, F. and Faugeras, O. (2001). Straight Lines Have to be Straight. *Machine Vision and Applications*, **13**(1), 14–24.

Dhond, U. R. and Aggarwal, J. K. (1989). Structure from Stereo – A Review. *IEEE Transactions on Systems, Man and Cybernetics*, **19**(6), 1489–1510.

Doucet, A., De Freitas, N., Murphy, K., and Russell, S. (2000). Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 176–183. Morgan Kaufmann Publishers, Inc.

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)*, pages 1553–1559.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous Localization and Mapping: Part I. *IEEE Robotics & Automation Magazine*, **13**(2), 99–110.

Durrant-Whyte, H., Rye, D., and Nebot, E. (1996). Localization of Autonomous Guided Vehicles. In *International Symposium of Robotics Research (ISRR)*, pages 613–625. Springer.

Elfes, A. (1989). Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, **22**(6), 46–57.

Engel, J., Sturm, J., and Cremers, D. (2012). Camera-Based Navigation of a Low-Cost Quadrocopter. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2815–2821.

Engels, C., Stewénius, H., and Nistér, D. (2006). Bundle Adjustment Rules. In *Photogrammetric Computer Vision (PCV)*, pages 266–271. Institute of Photogrammetry Bonn.

Eric, W. and Grimson, L. (1985). Computational Experiments with a Feature Based Stereo Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **7**(1), 17–34.

Felzenszwalb, P. and Huttenlocher, D. (2006). Efficient Belief Propagation for Early Vision. *International Journal of Computer Vision*, **70**(1), 41–54.

Feron, E. and Paduano, J. (2004). A Passive Sensor for Position and Attitude Estimation Using an Interferometric Target. In *IEEE Conference on Decision and Control (CDC)*, volume 2, pages 1663–1669.

Festo AG & Co. KG (2011). *SmartBird – Bird Flight Deciphered*. Esslingen, Germany.

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 (CACM)*, **24**(6), 381–395.

Fly & Check by Drone (2014). Luftbildinspektion. http://flyandcheck.de. Accessed: 05.04.2014.

Fossel, J.-D., Hennes, D., Alers, S., Claes, D., and Tuyls, K. (2013). OctoSLAM: a 3D mapping approach to situational awareness of unmanned aerial vehicles. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1363–1364. IEEE.

Fraundorfer, F. and Scaramuzza, D. (2012). Visual Odometry – Part II: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, **19**(2), 78–90.

Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., and Pollefeys, M. (2012). Vision-Based Autonomous Mapping and Exploration Using a Quadrotor MAV. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4557–4564.

Gauglitz, S., Höllerer, T., and Turk, M. (2011). Evaluation of Interest Point Detectors and Feature Descriptors for Visual Tracking. *International Journal of Computer Vision*, **94**(3), 1–26.

Gehrig, S. K. and Rabe, C. (2010). Real-Time Semi-Global Matching on the CPU. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 85–92.

Gehrig, S. K., Eberli, F., and Meyer, T. (2009). A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching. *Computer Vision Systems*, **5815**, 134–143.

Geiger, A., Roser, M., and Urtasun, R. (2011). Efficient Large-Scale Stereo Matching. In *Asian Conference on Computer Vision (ACCV)*, pages 25–38. Springer.

Geiger, A., Roser, M., and Urtasun, R. (2013). LIBELAS: Library for Efficient Large-scale Stereo Matching. http://cvlibs.net/software/libelas. Accessed: 03.12.2013.

Gordon, N., Salmond, D., and Smith, A. F. M. (1993). Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation. *Radar and Signal Processing, IEE Proceedings F*, **140**(2), 107–113.

Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. (2007). A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent. In *Robotics: Science and Systems (RSS)*.

Grisetti, G., Rizzini, D. L., Stachniss, C., Olson, E., and Burgard, W. (2008). Online Constraint Network Optimization for Efficient Maximum Likelihood Map Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1880–1885.

Gross, J. N., Gu, Y., Rhudy, M. B., Gururajan, S., and Napolitano, M. R. (2012). Flight-Test Evaluation of Sensor Fusion Algorithms for Attitude Estimation. *IEEE Transactions on Aerospace and Electronic Systems*, **48**(3), 2128–2139.

Hadsell, R., Bagnell, J. A., Huber, D., and Hebert, M. (2009). Accurate Rough Terrain Estimation with Space-Carving Kernels. In *Robotics: Science and Systems (RSS)*.

Haller, I. and Nedevschi, S. (2010). GPU Optimization of the SGM Stereo Algorithm. In *IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 197–202.

Haralick, B. M., Lee, C.-N., Ottenberg, K., and Nölle, M. (1994). Review and Analysis of Solutions of the Three Point Perspective Pose Estimation Problem. *International Journal of Computer Vision*, **13**(3), 331–356.

Harmat, A., Sharf, I., and Trentini, M. (2012). Parallel Tracking and Mapping with Multiple Cameras on an Unmanned Aerial Vehicle. In *International Conference on Intelligent Robotics and Applications (ICIRA)*, volume 1, pages 421–432. Springer.

Harris, C. and Stephens, M. (1988). A Combined Corner and Edge Detector. In *Alvey Vision Conference*, volume 15, pages 147–151.

Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge.

Hartung, J., Knapp, G., and Sinha, B. K. (2011). *Statistical Meta-Analysis with Applications*. John Wiley & Sons, Inc., Hoboken.

Heng, L., Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). Autonomous Obstacle Avoidance and Maneuvering on a Vision-Guided MAV Using On-Board Processing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2472–2477.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2010). RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *International Symposium on Experimental Robotics (ISER)*, volume 20, pages 22–25.

Hermann, S., Morales, S., Vaudrey, T., and Klette, R. (2011). Illumination Invariant Cost Functions in Semi-Global Matching. In *ACCV Workshop on Computer Vision in Vehicle Technology: From Earth to Mars (CVVT:E2M)*, pages 245–254. Springer.

Hermann, S. and Morales, S. and Klette, R. (2011). Half-Resolution Semi-Global Stereo Matching. In *IEEE Intelligent Vehicle Symposium (IV)*, pages 201–206.

Hirschmüller, H. (2005). Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 807–814.

Hirschmüller, H. and Scharstein, D. (2008). Evaluation of Stereo Matching Costs on Images with Radiometric Differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **31**(9), 1582–1599.

Honegger, D., Meier, L., Tanskanen, P., and Pollefeys, M. (2013). An Open Source and Open Hardware Embedded Metric Optical Flow CMOS Camera for Indoor and Outdoor Applications. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1736–1741.

Honeywell International, Inc. (2014). T-Hawk MAV. http://aerospace.honeywell.com/thawk. Accessed: 07.01.2014.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, pages 1–18.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2014). OctoMap – 3D Occupancy Mapping. http://octomap.github.io. Accessed: 20.02.2014.

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., and Roy, N. (2011). Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. In *International Symposium of Robotics Research (ISRR)*, pages 1–16. Springer.

Humenberger, M., Zinner, C., Weber, M., Kubinger, W., and Vincze, M. (2010). A Fast Stereo Matching Algorithm Suitable for Embedded Real-Time Systems. *Computer Vision and Image Understanding*, **114**(11), 1180–1202.

Husban (2013). The Husban X4 (H107). http://hubsan.com/products/HELICOPTER/H107.htm. Accessed: 19.12.2013.

IMAV (2013). International Micro Air Vehicle Conference and Flight Competition. http://www.imav2013.org. Accessed: 01.11.2013.

Itseez (2013). OpenCV. http://opencv.org. Accessed: 07.10.2013.

Keennon, M., Klingebiel, K., Won, H., and Andriukov, A. (2012). Development of the Nano Hummingbird: A Tailless Flapping Wing Micro Air Vehicle. In *AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, pages 1–24.

Kitt, B., Geiger, A., and Lategahn, H. (2010). Visual Odometry Based on Stereo Image Sequences with RANSAC-Based Outlier Rejection Scheme. In *IEEE Intelligent Vehicle Symposium (IV)*, pages 486–492.

Klein, G. and Murray, D. (2007). Parallel Tracking and Mapping for Small AR Workspaces. In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 1–10.

Klein, G. and Murray, D. (2009). Parallel Tracking and Mapping on a Camera Phone. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–86.

Klein, G. and Murray, D. (2010). Parallel Tracking and Mapping for Small AR Workspaces – Source Code. http://www.robots.ox.ac.uk/~gk/PTAM. Accessed: 25.10.2013.

Klette, R. (2014). *Concise Computer Vision*. Springer, London.

Klose, S. (2011). imu_filter. http://ros.org/wiki/imu_filter. Accessed: 30.10.2013.

Knuth, D., Larrabee, T., and Roberts, P. (1989). *Mathematical Writing*. Cambridge University Press, Cambridge.

Konolige, K. (1997). Improved Occupancy Grids for Map Building. *Autonomous Robots*, **4**(4), 351–367.

Konolige, K., Agrawal, M., and Sola, J. (2011). Large-Scale Visual Odometry for Rough Terrain. In *International Symposium of Robotics Research (ISRR)*, pages 201–212. Springer.

Kontron AG (2013). COMe-cPC2. http://us.kontron.com/products/computeronmodules/com+express/com+express+compact/comecpc2.html. Accessed: 20.12.2013.

Kroo, I. and Prinz, F. (2001). The Mesicopter: A Meso-Scale Flight Vehicle NIAC Phase II Technical Proposal. Technical report, Stanford University.

Kweon, I. S. and Kanade, T. (1990). High Resolution Terrain Map from Multiple Sensor Data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 127–134.

Labayrade, R., Aubert, D., and Tarel, J.-P. (2002). Real Time Obstacle Detection in Stereovision on Non Flat Road Geometry Through "V-disparity" Representatioion. In *IEEE Intelligent Vehicle Symposium (IV)*, volume 2, pages 646–651.

Lang, M. (2013). Drohnen-Auslieferung: DHL führt Paketkopter vor. Heise Online, http://heise.de/-2063059. Accessed: 07.01.2014.

Lowe, D. (1999). Object Recognition from Local Scale-Invariant Features. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1150–1157.

Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, volume 81, pages 674–679. Morgan Kaufmann Publishers, Inc.

Mair, E., Hager, G., Burschka, D., Suppa, M., and Hirzinger, G. (2010). Adaptive and Generic Corner Detection Based on the Accelerated Segment Test. In *European Conference on Computer Vision (ECCV)*, pages 183–196. Springer.

Matthies, L. and Elfes, A. (1988). Integration of Sonar and Stereo Range Data Using a Grid-Based Representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 727–733.

Meagher, D. (1980). Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer. Technical report, Electrical and Systems Engineering Department, Rensseiaer Polytechnic Institute.

Medioni, G. and Nevatia, R. (1985). Segment-Based Stereo Matching. *Computer Vision, Graphics, and Image Processing*, **31**(1), 2–18.

Mei, C., Benhimane, S., Malis, E., and Rives, P. (2008). Efficient Homography-Based Tracking and 3-D Reconstruction for Single-Viewpoint Sensors. *IEEE Transactions on Robotics*, **24**(6), 1352–1364.

Mei, C., Sibley, G., Cummins, M., Newman, P. M., and Reid, I. D. (2009). A Constant-Time Efficient Stereo SLAM System. In *British Machine Vision Conference (BMVC)*, pages 1–11.

Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). PIXHAWK: A System for Autonomous Flight Using Onboard Computer Vision. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2992–2997.

Meier, L., Tanskanen, P., Heng, L., Lee, G., Fraundorfer, F., and Pollefeys, M. (2012). PIXHAWK: A Micro Aerial Vehicle Design for Autonomous Flight Using Onboard Computer Vision. *Autonomous Robots*, pages 1–19.

Meier, L., Tanskanen, P., Heng, L., Honegger, D., and Fraundorfer, F. (2013). PIXHAWK Research Project. https://pixhawk.ethz.ch. Accessed: 20.12.2013.

Mellinger, D., Michael, N., and Kumar, V. (2012). Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors. *International Journal of Robotics Research*, **31**(5), 664–674.

Microdrones GmbH (2013). Technical Specification for the md4-1000. http://microdrones.com/products/md4-1000/md4-1000-technical-specification.php. Accessed: 19.12.2013.

Mikolajczyk, K. and Schmid, C. (2001). Indexing Based on Scale Invariant Interest Points. In *IEEE International Conference on Computer Vision (ICCV)*, pages 525–531.

Moravec, H. and Elfes, A. (1985). High Resolution Maps from Wide Angle Sonar. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 116–121.

Moravec, H. P. (1980). *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. Ph.D. thesis, Stanford University.

Murray, D. and Little, J. J. (2000). Using Real-Time Stereo Vision for Mobile Robot Navigation. *Autonomous Robots*, **8**(2), 161–171.

Nevatia, R. and Babu, K. R. (1980). Linear Feature Extraction And Description. *Computer Graphics, and Image Processing*, **13**(3), 257–269.

Nieuwenhuisen, M., Droeschel, D., Schneider, J., Holz, D., Läbe, T., and Behnke, S. (2013). Multimodal Obstacle Detection and Collision Avoidance for Micro Aerial Vehicles. In *European Conference on Mobile Robots (ECMR)*, pages 7–12. IEEE.

Nistér, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 1–652.

Nistér, D., Naroditsky, O., and Bergen, J. (2006). Visual Odometry for Ground Vehicle Applications. *Journal of Field Robotics*, **23**(1), 3–20.

Northrop Grumman Corp. (2014). Global Hawk. http://northropgrumman.com/capabilities/globalhawk. Accessed: 15.01.2014.

Okutomi, M. and Kanade, T. (1993). A Multiple-Baseline Stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(4), 353–363.

Palmisano, S., Gillam, B., Govan, D. G., Allison, R. S., and Harris, J. M. (2010). Stereoscopic Perception of Real Depths at Large Distances. *Journal of Vision*, **10**(6).

Pebrianti, D., Kendoul, F., Azrad, S., Wang, W., and Nonami, K. (2010). Autonomous Hovering and Landing of a Quad-Rotor Micro Aerial Vehicle by Means of on Ground Stereo Vision System. *Journal of System Design and Dynamics*, **4**(2), 269–284.

Peli, E. (1990). Contrast in Complex Images. *Journal of the Optical Society of America A (JOSA A)*, **7**(10), 2032–2040.

Perrollaz, M., Spalanzani, A., and Aubert, D. (2010). Probabilistic Representation of the Uncertainty of Stereo-Vision and Application to Obstacle Detection. In *IEEE Intelligent Vehicle Symposium (IV)*, pages 313–318.

Perrollaz, M., Yoder, J.-D., Nègre, A., Spalanzani, A., and Laugier, C. (2012). A Visibility-Based Approach for Occupancy Grid Computation in Disparity Space. *IEEE Transactions on Intelligent Transportation Systems*, **13**(3), 1383–1393.

Pfister, H., Zwicker, M., Van Baar, J., and Gross, M. (2000). Surfels: Surface Elements as Rendering Primitives. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 335–342.

Point Grey Research, Inc. (2012). Stereo Accuracy and Error Modeling. *Technical Application Note TAN2004006*. Richmond, Canada.

Point Grey Research, Inc. (2013). Firefly MV. http://ww2.ptgrey.com/USB2/fireflymv. Accessed: 25.12.2013.

Popper, B. (2013). UPS Researching Delivery Drones that Could Compete with Amazon's Prime Air. The Verge, http://vrge.co/1gB9mth. Accessed: 07.01.2014.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: An Open-Source Robot Operating System. In *ICRA 2009 Workshop on Open Source Software*.

Ranson, R. (2002). An Overview of VSTOL Aircraft and Their Contributions. In *AIAA International Powered Lift Conference and Exhibit*.

Raw Seeds Project (2009). Capture Session "Bicocca_2009-02-25b". http://www.rawseeds.org/rs/capture_sessions/view/5. Accessed: 03.12.2013.

Ribeiro, M. I. (2004). Kalman and Extended Kalman Filters: Concept, Derivation and Properties. Technical report, Institute for Systems and Robotics, Paris.

Rosten, E. (2013). libCVD. http://edwardrosten.com/cvd. Accessed: 07.10.2013.

Rosten, E. and Drummond, T. (2005). Fusing Points and Lines for High Performance Tracking. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1508–1515.

Rosten, E. and Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. In *European Conference on Computer Vision (ECCV)*, pages 430–443. Springer.

Rosten, E., Porter, R., and Drummond, T. (2010). FASTER and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **32**, 105–119.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). ORB: An Efficient Alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571.

Ruhnke, M. (2009). Benchmark Solution "GraphSLAM". http://www.rawseeds.org/rs/solutions/view/52. Accessed: 03.12.2013.

Rusu, R. B. and Cousins, S. (2011). 3D is Here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*.

Ryde, J. and Hu, H. (2010). 3D Mapping with Multi-Resolution Occupied Voxel Lists. *Autonomous Robots*, **28**(2), 169–185.

Sarkis, M. and Diepold, K. (2008). Sparse Stereo Matching Using Belief Propagation. In *IEEE International Conference on Image Processing (ICIP)*, pages 1780–1783.

Scaramuzza, D. and Fraundorfer, F. (2011). Visual Odometry – Part I: The First 30 Years and Fundamentals. *IEEE Robotics & Automation Magazine*, **18**(4), 80–92.

Scaramuzza, D. and Siegwart, R. (2008). Appearance-Guided Monocular Omnidirectional Visual Odometry for Outdoor Ground Vehicles. *IEEE Transactions on Robotics*, **24**(5), 1015–1026.

Scaramuzza, D., Achtelik, M. C., Doitsidis, L., Fraundorfer, F., Kosmatopoulos, E. B., Martinelli, A., Achtelik, M. W., Chli, M., Chatzichristofis, S. A., Kneip, L., Gurdan, D., Heng, L., Lee, G. H., Lynen, S., Meier, L., Pollefeys, M., Renzaglia, A., Siegwart, R., Stumpf, J. C., Tanskanen, P., Troiani, C., and Weiss, S. (2013). Vision-Controlled Micro Flying Robots: from System Design to Autonomous Navigation and Mapping in GPS-denied Environments. *IEEE Robotics & Automation Magazine*. Preprint.

Scharstein, D. and Pal, C. (2007). Learning Conditional Random Fields for Stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8.

Scharstein, D. and Szeliski, R. (2002). A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, **47**(1), 7–42.

Schauwecker, K. and Klette, R. (2010). A Comparative Study of Two Vertical Road Modelling Techniques. In *ACCV Workshop on Computer Vision in Vehicle Technology: From Earth to Mars (CVVT:E2M)*, pages 174–183. Springer.

Schauwecker, K. and Zell, A. (2013). On-Board Dual-Stereo-Vision for Autonomous Quadrotor Navigation. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 332–341. IEEE.

Schauwecker, K. and Zell, A. (2014a). On-Board Dual-Stereo-Vision for the Navigation of an Autonomous MAV. *Journal of Intelligent & Robotic Systems (JINT)*, **74**(1-2), 1–16.

Schauwecker, K. and Zell, A. (2014b). Robust and Efficient Volumetric Occupancy Mapping with an Application to Stereo Vision. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6102–6107.

Schauwecker, K., Morales, S., Hermann, S., and Klette, R. (2011). A Comparative Study of Stereo-Matching Algorithms for Road-Modeling in the Presence of Windscreen Wipers. In *IEEE Intelligent Vehicle Symposium (IV)*, pages 7–12.

Schauwecker, K., Klette, R., and Zell, A. (2012a). A New Feature Detector and Stereo Matching Method for Accurate High-Performance Sparse Stereo Matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5171–5176.

Schauwecker, K., Ke, N. R., Scherer, S. A., and Zell, A. (2012b). Markerless Visual Control of a Quad-Rotor Micro Aerial Vehicle by Means of On-Board Stereo Processing. In *Autonomous Mobile System Conference (AMS)*, pages 11–20. Springer.

Scherer, S. A., Dube, D., and Zell, A. (2012). Using Depth in Visual Simultaneous Localisation and Mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5216–5221.

Schmid, C., Mohr, R., and Bauckhage, C. (2000). Evaluation of Interest Point Detectors. *International Journal of Computer Vision*, **37**(2), 151–172.

Seddon, J. M. and Newman, S. (2011). *Basic Helicopter Aerodynamics*, volume 40. John Wiley & Sons, Inc.

Shade, R. and Newman, P. (2011). Choosing Where to Go: Complete 3D Exploration with Stereo. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2806–2811.

Shen, S., Michael, N., and Kumar, V. (2011). Autonomous Multi-Floor Indoor Navigation with a Computationally Constrained MAV. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 20–25.

Shen, S., Michael, N., and Kumar, V. (2012). Autonomous Indoor 3D Exploration with a Micro-Aerial Vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 9–15.

Shen, S., Mulgaonkar, Y., Michael, N., and Kumar, V. (2013). Vision-Based State Estimation for Autonomous Rotorcraft MAVs in Complex Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1758–1764.

Shi, J. and Tomasi, C. (1994). Good Features to Track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600.

Sim, R., Elinas, P., Griffin, M., and Little, J. J. (2005). Vision-Based SLAM Using the Rao-Blackwellised Particle Filter. In *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, volume 14, pages 9–16.

Smith, S. M. and Brady, J. M. (1997). SUSAN – A New Approach to Low Level Image Processing. *International Journal of Computer Vision*, **23**(1), 45–78.

Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2010). Real-Time Monocular SLAM: Why Filter? In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2657–2664.

Strasdat, H., Davison, A. J., Montiel, J. M. M., and Konolige, K. (2011). Double Window Optimisation for Constant Time Visual SLAM. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2352–2359.

Sun, J., Zheng, N., and Shum, H. (2003). Stereo Matching using Belief Propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **25**(7), 787–800.

Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2007). A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**(6), 1068–1080.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press, Cambridge.

Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., Grixa, I., Ruess, F., Suppa, M., and Burschka, D. (2012). Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue. *IEEE Robotics & Automation Magazine*, **19**(3), 46–56.

Tournier, G. P., Valenti, M., How, J., and Feron, E. (2006). Estimation and Control of a Quadrotor Vehicle Using Monocular Vision and Moiré Patterns. In *In AIAA Guidance, Navigation and Control Conference*, pages 2006–6711.

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)*, pages 2276–2282.

Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (1999). Bundle Adjustment – A Modern Synthesis. In *ICCV Workshop on Vision Algorithms*, pages 298–372. Springer.

Tuytelaars, T. and Mikolajczyk, K. (2008). Local Invariant Feature Detectors: A Survey. *Foundations and Trends in Computer Graphics and Vision*, **3**(3), 177–280.

Vincent, E. and Laganiere, R. (2001). Matching Feature Points in Stereo Pairs: A Comparative Study of Some Matching Strategies. *Machine Graphics and Vision*, **10**(3), 237–260.

Viola, P. and Jones, M. (2002). Robust real-time object detection. *International Journal of Computer Vision*, **57**(2), 137–154.

Weiss, S., Scaramuzza, D., and Siegwart, R. (2011). Monocular-SLAM–Based Navigation for Autonomous Micro Helicopters in GPS-Denied Environments. *Journal of Field Robotics*, **28**(6), 854–874.

Wenzel, K. E. and Zell, A. (2009). Low-Cost Visual Tracking of a Landing Place and Hovering Flight Control with a Microcontroller. In *International Symposium on Unmanned Aerial Vehicles (UAV)*, pages 1–15. Springer.

Witt, J. and Weltin, U. (2012). Sparse Stereo by Edge-Based Search Using Dynamic Programming. In *IEEE International Conference on Pattern Recognition (ICPR)*, pages 3631–3635.

Wood, R. J. (2008). The First Takeoff of a Biologically Inspired At-Scale Robotic Insect. *IEEE Transactions on Robotics*, **24**(2), 341–347.

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 *ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*.

Yang, Q., Wang, L., Yang, R., Wang, S., Liao, M., and Nistér, D. (2006). Real-Time Global Stereo Matching Using Hierarchical Belief Propagation. In *British Machine Vision Conference (BMVC)*, pages 989–998.

Yang, Q., Wang, L., and Ahuja, N. (2010). A Constant-Space Belief Propagation Algorithm for Stereo Matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1458–1465.

Yang, S., Scherer, S. A., and Zell, A. (2012). An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle. *Journal of Intelligent & Robotic Systems (JINT)*, **69**, 499–515.

Yang, S., Scherer, S. A., and Zell, A. (2013a). An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle. *Journal of Intelligent & Robotic Systems (JINT)*, **69**(1–4), 499–515.

Yang, S., Scherer, S. A., Schauwecker, K., and Zell, A. (2013b). Onboard Monocular Vision for Landing of an MAV on a Landing Site Specified by a Single Reference Image. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 317–324. IEEE.

Yang, S., Scherer, S. A., Schauwecker, K., and Zell, A. (2014). Autonomous Landing of MAVs on an Arbitrarily Textured Landing Site Using Onboard Monocular Vision. *Journal of Intelligent & Robotic Systems (JINT)*, **74**(1-2), 27–43.

Zabih, R. and Woodfill, J. (1994). Non-Parametric Local Transforms for Computing Visual Correspondence. In *European Conference on Computer Vision (ECCV)*, volume 801, pages 151–158. Springer.

Zhang, Z. (2000). A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(11), 1330–1334.