Stereo vision and mapping with aerial robots

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

M.Sc. Radouane Ait Jellal
aus Taznakhte, Marokko

Tübingen
2020

Tag der mündlichen Qualifikation:	07.05.2020
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Andreas Zell
2. Berichterstatter:	Prof. Dr. Andreas Schilling

To my family: my mother Kabira Laamraoui, the memory of my father El-houcine, my wife Ilham and my twin children Mohamed Shahin and Ilyas.

Abstract

Micro air vehicles (MAVs) have attracted researchers and industrials in recent years. However, most of existing commercial MAVs rely on external positioning systems and thus they have limited usage. Providing more autonomy to these robots will extend their range of applications to include a set of challenging real-life problems. In this thesis, we use a stereo camera mounted on a quadcopter MAV to achieve full autonomous flights in indoor as well as outdoor environments. We study problems related to binocular dense stereo matching, stereo visual SLAM and 3D obstacle avoidance using stereo vision.

First, we propose a hybrid stereo SLAM which combines feature-based SLAM with direct image alignment. We address the following important question: is it better to estimate the pose from a set of pre-computed feature correspondences or to estimate the pose and the feature correspondences simultaneously? We propose to combine both methods. We start by computing a pose estimate using feature correspondences. This can efficiently and effectively handle large movements. The abstraction of the image content to a set of sparse features might introduce a loss of accuracy because useful details might be dropped. In a second step, we refine the pose using direct image alignment. In this step of the algorithm, we take into consideration the details that have been ignored on the first step to simultaneously estimate the pose and the pixel correspondences.

Secondly, we investigate efficient dense binocular stereo matching algorithms that can run in real-time on-board our experimental MAV. We use the dense stereo matching to estimate disparity maps from the stereo camera. The disparity maps are the key component of our MAV. We use these disparity maps in the refinement step of our hybrid stereo SLAM as well as to create a consistent 3D occupancy grid map for the path planner. We show that by using edge features and a special variant of Delaunay triangulation, we could improve both the robustness and efficiency of the popular ELAS local stereo matching algorithm. We have submitted our results to the Middleburry stereo benchmark and our algorithm is listed on the permanent ranking table of this benchmark.

Finally, we show an application of the afore-mentioned algorithms in a system for autonomous stereo-based 3D obstacle avoidance in outdoor environment. We designed a modular system. We added additional modules for building 3D occupancy grid mapping and for planning collision-free paths in 3D space. In our experiments, we show that we can map large environments with several hundreds of keyframes.

Kurzfassung

In den letzten Jahren haben Roboter für Mikro-Luftfahrzeuge (MAVs) Forscher und Industrie angezogen. Die meisten handelsüblichen MAVs sind jedoch auf externe Positionierungssysteme angewiesen und haben daher eine begrenzte Nutzung. Durch die Bereitstellung von mehr Autonomie für diese Roboter wird das Anwendungsspektrum um eine Reihe anspruchsvoller realer Problemstellungen erweitert. In dieser Arbeit verwenden wir eine Stereokamera, die auf einem Quadcopter-MAV montiert ist, um vollständige autonome Flüge in Innen- und Außenbereichen durchführen zu können. Wir untersuchen Probleme im Zusammenhang mit binokularem dichtem Stereo-Matching, Stereovisuellem SLAM und 3D-Hindernisvermeidung mit Stereo-Vision.

Zunächst schlagen wir ein Hybrid-Stereo-SLAM vor, das merkmalsbasiertes SLAM mit direkter Bildausrichtung kombiniert. Wir sprechen die folgende wichtige Frage an: Ist es besser, die Position aus einem Satz vorberechneter Merkmalskorrespondenzen zu schätzen oder die Position und die Merkmalskorrespondenz gleichzeitig zu schätzen? Wir schlagen vor, beide Methoden zu kombinieren. Wir beginnen mit der Berechnung einer Positionsschätzung unter Verwendung von Merkmalskorrespondenzen. In einem zweiten Schritt verfeinern wir die Position durch direkte Bildausrichtung. In diesem Schritt des Algorithmus berücksichtigen wir die Details, die im ersten Schritt ignoriert wurden, um gleichzeitig die Pose und die Pixelkorrespondenzen zu schätzen.

Zweitens untersuchen wir effiziente dichte binokulare Stereo-Matching-Algorithmen, die in Echtzeit an Bord unseres experimentellen MAV ausgeführt werden können. Wir verwenden das dichte Stereo-Matching, um die Disparitätskarten der Stereokamera zu schätzen. Die Disparitätskarten sind der Hauptbestandteil unseres MAV. Diese Karten verwenden wir im Verfeinerungsschritt unseres Hybrid-Stereo-SLAMs sowie zur Erstellung einer 3D-Belegungsrasterkarte für den Pfadplaner. Wir zeigen, dass wir durch die Verwendung von Kantenmerkmalen und einer speziellen Variante der Delaunay-Triangulation sowohl die Robustheit als auch die Effizienz des bekannten lokalen ELAS-Stereomatching-Algorithmus verbessern können. Wir haben unsere Ergebnisse dem Middleburry-Stereo-Benchmark übermittelt, und unser Algorithmus ist in der permanenten Rangliste dieses Benchmarks aufgeführt.

Schließlich zeigen wir eine Anwendung der oben genannten Algorithmen in einem System zur autonomen stereobasierten 3D-Hindernisvermeidung in Außenbereichen. Wir haben ein modulares System entwickelt. Wir haben zusätzliche Module hinzugefügt, um 3D-Belegungsrasterkarten der Roboterumgebung zu erstellen und zur Planung kollisionsfreier Wege im 3D-Raum. In unseren Experimenten zeigen wir, dass wir große Umgebungen mit mehreren hundert Keyframes abbilden können.

Acknowledgments

I am deeply grateful to Prof. Andreas Zell for giving me the opportunity to work on this interesting research topic and for keeping me focused in my research. I am also grateful to Konstantin Schauwecker, Sebastian Scherer and Andreas Masselli who helped me to gain my footing in stereo vision and to set-up my research platform.
My co-authors, Manuel Lange and Benjamin Wassermann, have been instrumental in completing this dissertation. I would like to thank them for their successful collaboration. I appreciate all the fruitful discussions I had with Prof. Andreas Schilling and my co-authors.
Thanks to my colleagues at the chair of cognitive systems: Yann Berquin, Goran Huskic, Robert Pech, Ya Wang, Ran Liu and Julian Jordan for the many interesting discussions we have had over the last few years. Special thanks should be given to Klaus Beyreuther and Vita Serbakova who offer assistance to our work.
Thanks to all members and to the organizing committee of the Mini-GRK (mini-graduate school) for the initial founding of my research, for the fruitful collaboration and for the joint participation at the EuRoC competition. I would like to thank, in particular, Prof. Andreas Schilling, Prof. Heinrich Bülthoff, Prof. Hanspeter Mallot, Yuyi Liu, Manuel Lange, Marcin Odelga and Gerrit Ecke.
Further thanks go to Thorsten Linder, my team leader at the ZF Friedrichshafen AG company, who encouraged me to finish writing my dissertation. I would like to thank Yamna Moumni for taking care of my children during the last days of writing this dissertation.
Finally, I must express my very profound gratitude to my mother, Kabira Laamraoui, and to my wife, Ilham Laouaj, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. Thank you.

Contents

Chapter 1

Introduction

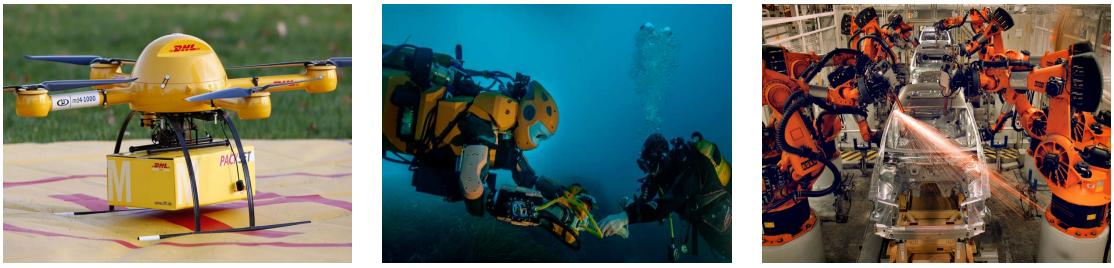
1.1 Mobile robots

Robots are playing an important role in modern life. The introduction of industrial robot arms has revolutionized the assembly line. Since the invention of the first mass produced industrial robot arm for factory automation in 1961 by the General Motors company (see Kelly (2018)), the development of robot arms kept increasing. The industrial robot arms can operate continuously and in most cases they are faster and more precise than human workers. Therefore, massive production of goods at cheap price became possible and also jobs which are unpleasant or dangerous for humans were efficiently performed by robots. Although the stationary industrial robot arms have made a big success, they are limited by being attached to a fixed surface. Applications for which the robot needs to freely navigate in its environment require providing the robots with mobility and with more intelligence. These kinds of robots are called mobile robots. Fig. 1.1 shows some of the different civil applications for which robots can be used.

Multirotor micro aerial vehicles (MAVs) are a kind of mobile robots which has recently gained much interest in research as well as in industry. The multirotors are able to fly over uneven terrains, which might be inaccessible for ground vehicles, and have more degrees of freedom. Unlike fixed wing flying robots, multirotors can hover at a fixed position and can efficiently navigate in cluttered indoor and outdoor environments. MAV based applications are diverse and steadily growing. For example, the octacopter AscTec Falcon 8[1] is a commercial multirotor which is used for industrial inspection (see Ascending Technologies (2018)). This robot has a range of 1 km and up to 22 minutes endurance time. The same robot, equipped with different sensors, has been used also for MAV assisted precision agriculture and aerial imaging. Companies such as Amazon, UPS and DHL have started projects for parcel delivery using multirotors (see Desjardins (2018) and Nicas (2014)). One motivation for these companies to use MAVs is the express delivery of parcels to isolated areas. Additionally, MAVs have shown their advantages on disaster relief. On the Fukushima nuclear disaster, the T-Hawk MAV has been used to photograph the nuclear plant from above to get a detailed view of the interior damage (see Honig (2011)). We believe that MAVs will make a revolution on the robotic field

[1] Falcon 8 is a product of the Ascending technologies company

(a) (b) (c)



(d) (e) (f)

Figure 1.1: A collection of robots which shows the diversity of robotic applications. (a) the rover "Curiosity" from NASA. This rover is equipped with a stereo camera and has landed on Mars on 6 August 2012. (b) the ROBOCLEANER RC 3000 from the German company Kärcher (Alfred Kärcher SE & Co. KG). (c) self-driving delivery ground robot from the UK company Starship Technologies. (d) quadcopter for parcel delivery from the German post DHL. (e) underwater humanoid robot from the university of Stanford USA. (f) a Volkswagen factory equipped with Kuka industrial fixed robots.

similar to what industrial robot arms have made in the past.

In most of the applications listed above, the MAVs are either manually controlled by expert pilots or use external positioning systems (e.g. GPS). In recent years, much interest has been shown in developing new methods for fully autonomous MAVs which rely only on the on-board sensors with no human intervention. Depending on whether the MAV environment is known in advance (i.e. the MAV has an offline map), partially known or completely unknown, the complexity of the navigation task varies significantly. The case of a completely unknown environment is the most challenging one. This is the case that we address in this dissertation. These are MAVs which can fully autonomously achieve some tasks in an unknown environment without any human intervention.

(a) (b) (c)

Figure 1.2: Depth can be perceived from single 2D views by the human visual system. Monocular depth cues are used for this purpose. (a): Texture gradient. (b): Linear perspective. (c): Aerial perspective.

1.2 Depth cues in the human visual system

Stereo vision in robotic is inspired by the human visual system. Human perceive depth using monocular and binocular depth cues. In this thesis, we mimic one type of the binocular depth cues (retinal disparity) to provide our quadcopter with depth information.

1.2.1 Monocular depth cues:

Thanks to monocular depth cues, humans have the ability to perceive depth when viewing with only one eye or when viewing a 2D image of a 3D scene (see Fig. 1.2). Some monocular depth cues rely on prior knowledge, which can be used to judge depth in similar environments. There are local (e.g. blur) monocular depth cues for which depth can be judged using only local image properties. There are global monocular depth cues (e.g. overlap), which require contextual information to judge depth (see Saxena *et al.* (2007)). In the following, we describe some relevant monocular depth cues:
Retinal image size: Combined with prior knowledge, the retinal image size of an object can help judge its depth. For example, humans have a prior knowledge about the size of a bicycle. If they see a bicycle on an image they can judge the depth of the bicycle.
Texture gradient: The texture appears smoother for farther away objects. While sharp details might be clearly seen for closer objects, these sharp details might be invisible for farther away objects. An example is shown in Fig. 1.2(a) with two trees located at different distances.
Linear perspective: The 2D projections of 3D parallel lines appear to converge at the horizon. This give an indication of the depth. The closer together the lines are the greater the distance. Fig. 1.2(b) shows an illustration of the linear perspective depth cue.
Overlap (or interposition): When an object overlaps another object, the object which blocks the view is perceived (by humans) as closer than the obscured object.

Motion parallax: Let's consider a moving person viewing a static environment. Closer objects seem to move faster (across the retina) than farther away objects.
Aerial perspective: Vapor and dust particles on the air might cause distant objects to appear bluish and hazy (see Fig. 1.2(c)). This phenomena is due to light scattering. The blue light, which has the shortest wave lengths in the visible spectrum, is the most sensitive to this phenomena.
Accommodation: The eye lens become thinner when focusing on distant objects. This information can be used by the brain to judge depth.

1.2.2 Binocular depth cues:

In addition to monocular depth cues, humans use binocular depth cues to perceive depth. The main binocular depth cue is retinal disparity (or stereopsis). In the following, we introduce two important binocular depth cues: retinal disparity and convergence.
Retinal disparity: Human capture two images (each eye captures one) from slightly different viewpoints. These differences are known as retinal disparity (or binocular parallax). The human visual system is sensitive to these differences. The human brain fuses the two views to perceive depth.
Convergence: Focusing on very close-up objects causes the eyes to turn inward instead of moving together. This information can be used by the brain to judge depth.

1.3 Depth estimation and its robotic applications

Depth estimation is an important task in mobile robotics. It is essential for many tasks such as 3D reconstruction, obstacle avoidance, navigation, recognition, and object grasping. Laser scanners were the favorite sensor for a long time, they were used to estimate the geometry of the robot environment. The main reason for this is the high accuracy of their distance measurements. However, laser scanners have many drawbacks that constrain their usage. These drawbacks include the relatively low frame rate, the sparsity of the measurements, the weight, and the cost. Recently, the robotic community has shown an increasing interest in computer vision. Cameras tackle many drawbacks of laser scanners. The introduction of RGB-D sensors[2] with their sufficiently accurate depth map and their real time capability has accelerated the research on indoor mobile robotics. Since these Kinect-style RGB-D sensors are based on the projection and capturing of structured infra-red light, they are not designed for outdoor usage with substantial sunlight. There is a need for more adequate sensors. Stereo cameras are the most adequate depth estimation sensors to be used for a wide range of outdoor and indoor applications. They provide data at high frame rates, they are lightweight, passive, energy efficient, and customizable. The major drawback of a stereo camera with no dedicated computing unit is

[2] e.g. Microsoft Kinect, Asus Xtion Pro or Intel RealSense

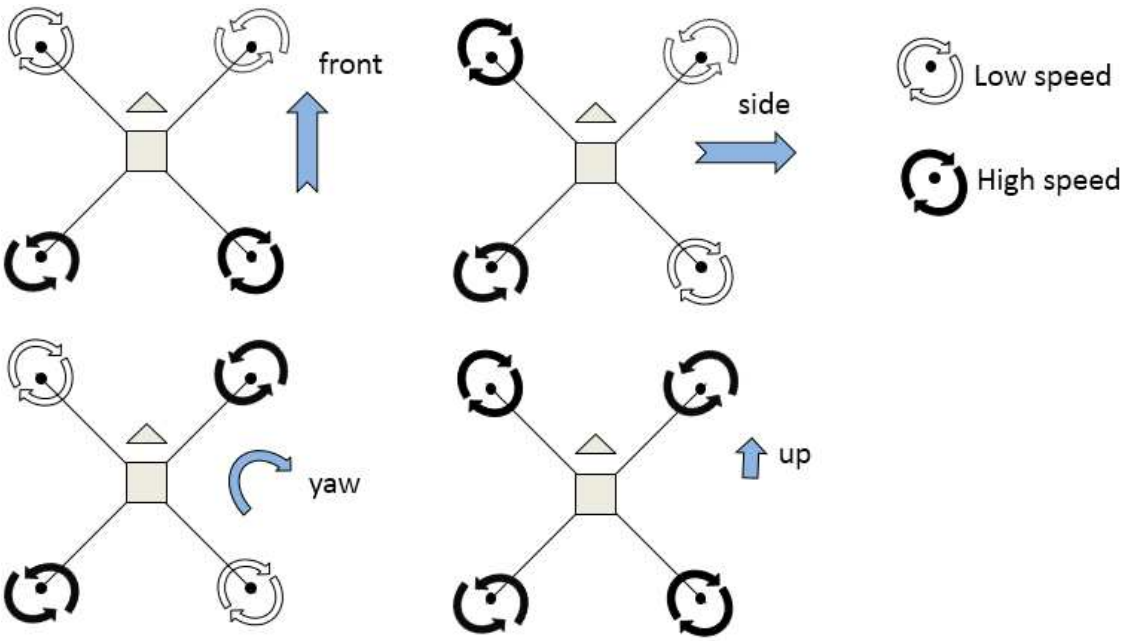


Figure 1.3: Illustration of a quadcopter MAV in X-configuration. Flying in different directions can be achieved by individually controlling the speed of each motor.

the need to use the robot CPU for performing the stereo matching. There is a need for efficient stereo matching algorithms in order not to exhaust the robot CPU through stereo matching. In this thesis, we address this drawback and focus our research on developing and using efficient stereo vision methods.

1.4 The quadcopter MAVs

Quadcopter MAVs are small unmanned aircrafts that use four motors with fixed pitch propellers (see Fig. 1.1(d) and Fig. 1.3). While spinning, the motors create thrust that can lift the quadcopter. A spinning motor generates a drag torque in the direction opposite to the propeller spin. In order to cancel out the torques generated by the four motors, the motors are configured in two pairs. The motors of one pair spin in a clockwise direction and the motors of the second pair spin in counter-clockwise direction. Unlike the conventional helicopter aircraft which uses a complex cyclic-pitch mechanism to control the aircraft's pitch and roll angle, the quadcopter uses a simple principle to control them. A quadcopter uses fixed pitched propellers and individually controls the speed of each motor to create a variable thrust between the four motors. Fig 1.3 illustrates motor speeds for basic quadcoptor maneuvers. Moving forward (+ X-axis) would require the two motors in the back to spin faster than the two motors in the front. Controlling the

yaw angle (rotation around the Z-axis) would require two rotors, which spin in the same direction, to spin faster than the other two motors.

1.5 Benefits and challenges of quadcopters

Quadcopters MAVs are interesting small robots that can be used in environments which are inaccessible for other types of robots. A quadcopter can for example easily fly above water from one side of a river to the other side and can land on a building roof. Unlike a fixed wing MAV, a quadcopter can fly in cluttered environments and it can hover at a fixed position. Thus, it is convenient to use quadcopters indoors in rescue applications. A quadcopter is highly flexible and fly in almost any direction while fixed wing MAVs have limited flexibility. Well known limitations of quadcopter MAVs are the payload and the endurance time. This has a direct impact on the choice of on-board sensors that can be mounted on a quadcopter. The afore-mentioned limitations of quadcopters are in particular challenging for the fully autonomous quadcopters. A quadcopter which builds a sufficiently detailed map of its environment and uses that map to autonomously navigate, requires sensors and computing processors that are usually heavy and power consuming. In this thesis, we focus on the development of efficient algorithms for safe navigation based on stereo vision.

1.6 Contributions and outline

To achieve fully autonomous flights with a quadcopter in unkonwn environments using a stereo camera requires the quadcopter to use many vision algorithms. The perception of the quadcopter environment and the localization are important tasks to achieve full autonomous flights. This thesis makes contributions in computer vision and robotic domains that address these tasks.

Chapter 3 (page 39)

The first contribution is a hybrid stereo algorithm which combines the feature-based SLAM with direct image alignment methods. The algorithm extends the popular feature-based algorithm ORB-SLAM2 (see Mur-Artal and Tardós (2016)) to include a refinement step, which is based on direct image alignment. The refinement step initializes its pose using the final pose estimated by the feature based ORB-SLAM2 tracking thread. The depth map is then used for warping the reference (left) image to synthesize the (left) image as if it is captured from the pose of the target image. The motion parameter vector that achieves a small error between the warped image and the target image is selected. The refinement step uses a conceptually different method for iteratively estimating the best pose increment. In particular, no feature detection and matching steps are required and considerably more pixels are used on the computation of the pose increment than features. This refinement step has shown improvement on the performance in particular

when the trajectory does not include loops. The hybrid stereo SLAM algorithm was published at the 2017 ECMR Conference in Paris, France (Ait Jellal and Zell (2017)).

Chapter 4 (page 69)

The second contribution is an efficient dense binocular stereo matching algorithm. It is used to estimate depth from stereo. The depth is used on our autonomous quadcopter for two purposes. First, on the hybrid stereo SLAM for warping the (left) intensity images. Second, for constructing a volumetric occupancy grid map. We call our stereo matching algorithm LS-ELAS since it extends the popular stereo matching ELAS (see Geiger *et al.* (2011a)). LS-ELAS is based on line segments, therefore LS in LS-ELAS refers to line segments. As in ELAS, LS-ELAS starts by finding robust matches for a small set of pixels: the support points. The search for the correspondences of the support points is done in linear-time using the whole disparity search range. The support points are used to compute the mean disparity using Delaunay triangulation to generate a triangle mesh from the support points. A prior based on the mean disparity can be set for all the remaining pixels on the image. Thus, the matching of the remaining pixels can be achieved in constant-time by searching for matches in a fixed interval of candidates disparities around the mean disparity. This results in a near constant-time algorithm for stereo matching. There are two main differences between the algorithms LS-ELAS and ELAS. In LS-ELAS, the support point candidates are sampled along edges which allow for depth discontinuity awareness. The depth discontinuity is further enforced by using the constrained Delaunay triangulation. ELAS uses a uniform grid for identifying the support point candidates and it uses the original Delaunay triangulation. The results on the popular Middlebury stereo benchmark showed that LS-ELAS improves the performance across different error metrics. The LS-ELAS stereo matching algorithm was published at the 2017 ICRA Conference in Singapore (see Ait Jellal *et al.* (2017)).

Chapter 5 (page 105)

The last contribution of this thesis shows a full system that uses stereo vision for outdoor obstacle avoidance in an unknown environment. The two afore-mentioned contributions constitute the core of this system. Besides the hybrid stereo SLAM and LS-ELAS, a range of other vision algorithms was needed to run in real-time on-board the quadcopter. The system builds a volumetric occupancy grid map from the disparity images and performs 3D path planning on this map. This way, obstacle avoidance can be achieved by following the collision-free paths from the planner.

Chapter 2

Background

This chapter intends to give fundamentals that might be necessary for the understanding of this thesis. In this dissertation, we focus on the vision part of the autonomous quadcopter. First, we introduce the mathematical foundations with an emphasis on the least squares method for model fitting and optimization algorithms. The last part of the background chapter explains different aspects of stereo vision. We start with the explanation of a popular camera model. We then, present the case of two views which can be captured either at the same time using two physical cameras or by a single moving camera. Given two pictures of a scene captured from different viewpoints, stereo vision aims at recovering the geometry of the 3D scene by establishing pixel correspondences on both images and then triangulating the 3D scene points. This process is the inverse of the image projection process. We will introduce the geometrical properties for the two views perspective. Two important applications for the two views perspective are the depth estimation and the motion estimation. We review the Essential matrix, which is useful for estimating the motion from 2D-2D correspondences. We also review the PnP algorithm, which uses a conceptually different method for motion estimation from 3D-2D correspondences. The two views perspective constitutes the elementary case for sequences of images which are captured while the camera is moving. For the case of image (or stereo pair) sequences, we review the techniques used for efficiently solving simultaneous localization and mapping (SLAM). Finally, we review techniques for the case of image sequences. We start by frame to frame visual odometry and then introduce efficient techniques for solving globally consistent full SLAM in large scale.

2.1 Mathematical foundations

2.1.1 Least squares method

The least squares (LS) algorithms are used for optimizing non-linear cost functions in many parts of our system. We use least squares in the feature based tracking to optimize re-projection distances. Then, we use least squares in the direct image alignment to minimize photometric errors (intensity differences). In the local mapping thread of the SLAM algorithm, least squares are used in the local bundle adjustment (LBA) to locally

optimize the map. In the SLAM back-end, least squares are used to optimize a pose graph when loops are detected and afterwards to optimize the whole graph (all keyframe poses and all map points) by global bundle adjustment (GBA).
In general, given an error function $e(x) = (e_1(x), ..., e_m(x))^T \in \mathbb{R}^m$, where the variable $x = (x_1, ..., x_n)^T \in \mathbb{R}^n$ is a vector of n parameters. We seek to estimate the optimal solution x^* that minimizes the energy function $E(x)$ given by Eq. 2.2 (see Madsen and Tingleff (2004)).

$$x^* = \underset{x}{\operatorname{argmin}} E(x) \tag{2.1}$$

$$= \underset{x}{\operatorname{argmin}} \sum_{i=0}^{m} |e_i(x)|^2 \tag{2.2}$$

$$= \underset{x}{\operatorname{argmin}} e(x)^\intercal e(x) \tag{2.3}$$

In general, e is non-linear. So, we compute a linear approximation using the first order Taylor expansion and solve the optimization problem iteratively. Let us consider a small perturbation δx around the initial guess $\breve{x}$. We get the following approximation: $e(\breve{x} + \delta x) \approx e(\breve{x}) + J\delta x$, here J is the Jacobian of $e(x)$ evaluated at $\delta x = 0$ (at $x = \breve{x}$). The elements E_i of the energy function $E(x)$ can then be approximated (see Madsen and Tingleff (2004)) such that:

$$\begin{aligned} E(\breve{x} + \delta x) &= e(\breve{x} + \delta x)^T e(\breve{x} + \delta x) \\ &\approx (e(\breve{x}) + J\delta x)^T (e(\breve{x}) + J\delta x) \qquad (2.4) \\ &= E(\breve{x}) + 2\delta x^\intercal J^\intercal e(\breve{x}) + \delta x^\intercal J^\intercal J \delta x \\ &= E(\breve{x}) + 2\delta x^\intercal J^\intercal e(\breve{x}) + \delta x^\intercal H \delta x \qquad (2.5) \end{aligned}$$

Where $H = J^\intercal J$ is the approximate Hessian.

Gauss-Newton method

By computing the derivative with respect to δx and setting it to zero we can compute the increment δx^* which minimizes Eq. 2.5 and solves the following linear system (Eq. 2.6):

$$H\delta x^* = -J^\intercal e(\breve{x}) \tag{2.6}$$

This linear system is called "normal equations" and it can be written as:

$$A\mathrm{x} = b \tag{2.7}$$

Where: $A = H$, $b = -J^\intercal e(\breve{x})$ and $\mathrm{x} = \delta x^*$. There are many methods for solving this linear system. Usually, the linear system in Eq. 2.6 is solved by some kind of factorization,

without inverting the matrix A (see Madsen and Tingleff (2004)).
The increment δx^* is then added to the initial guess $\breve{x}$.

$$x^* = \breve{x} + \delta x^* \tag{2.8}$$

The Gauss-Newton solver iterates the following steps: First, it locally approximates the non-linear cost function with a linear function according to Eq. 2.4. Then, it computes in closed form the increment δx^* according to Eq. 2.6. Finally, it updates the parameter vector according to Eq. 2.8 and starts a new iteration. While the intermediate problem (linear approximation) is solved in closed form, the initial non-linear problem in Eq. 2.2 needs to be iteratively solved. At every iteration, the currently updated parameter vector is used as an initial guess (linearization point). The iterative process continues until some termination criteria are met, e.g., when the algorithm converges or we reach a maximum number of iterations.

Levenberg-Marquart method

The Gauss-Newton method can converge very quickly but it is very sensitive to the initial estimate $\breve{x}$. The Gauss-Newton method requires the initial estimate to be close enough to the optimal solution x^*. Otherwise, the Gauss-Newton might converge very slowly or fail to converge. To tackle this problem, Levenberg (1944) and Marquardt (1963) proposed to add a damping term into Gauss-Newton method. The Levenberg-Marquardt method solves the following normal equations (see Madsen and Tingleff (2004)):

$$(J^\intercal J + \lambda I)\delta x^* = -J^\intercal e(\breve{x}) \tag{2.9}$$

The damping term is controlled by a parameter λ. Different strategies have been proposed for updating the parameter λ (at each iteration).
The Levenberg-Marquardt method can be seen as a combination of the Gauss-Newton method and the gradient decent method. The Gauss-Newton uses a second order approximation and allows for fast convergence when the initial is close enough to the optimum (λ small). On the other hand, the gradient decent allows to (relatively) slowly moving towards the optimum when the initial estimate is far away from the optimum (λ large).

2.1.2 Lie algebra parameterization for motion estimation

A rigid body transform g transforms a point $P \in \mathbb{R}^3$ in the 3D space to a point $g(P) \in \mathbb{R}^3$ in 3D space:

$$\begin{aligned} g &: \mathbb{R}^3 \to \mathbb{R}^3 \\ P &\to g(P) \end{aligned} \tag{2.10}$$

The rigid body transform g can be expressed by a 4×4 matrix $T \in SE(3)$, which is composed of a rotation matrix $R \in SO(3)$ and a translation vector $t = (t_x, t_y, t_z) \in \mathbb{R}^3$.

$$g(P) = g(P,T) = TP = RP + t \tag{2.11}$$

Where

$$T = \begin{bmatrix} R & t \\ 0_{1\times 3} & 1 \end{bmatrix}; and\, R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.12}$$

We use the Lie group $SE(3)$ and its corresponding Lie algebra $se(3)$ to get a minimal parameterization of the 3D rigid body transforms. On the associated Lie algebra $se(3)$, the corresponding transform is a 6-dimentional vector $\xi = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6)$. These coordinates are called the twist coordinates, where the first (ξ_1, ξ_2, ξ_3) are the linear velocities and (ξ_4, ξ_5, ξ_6) are the angular velocities. Here, we represent the rotation part of the transformation with exactly three parameters (ξ_4, ξ_5, ξ_6) rather than nine parameters as in the rotation matrix representation. The use of the Lie group solves the singularities of the compact Euler-angles representation.
A rigid body transform T can be mapped to its corresponding ξ using the logarithmic map as follows:

$$\begin{aligned} log &: SE(3) \to se(3) \\ g &\to \xi = log(T) \end{aligned} \tag{2.13}$$

The inverse of this operation is the exponential map.

$$\begin{aligned} exp &: se(3) \to SE(3) \\ \xi &\to T = T(\xi) = exp(\xi) \end{aligned} \tag{2.14}$$

2.2 Image formation and camera model

This background section is based on the work of Moons *et al.* (2010) and the book Hartley and Zisserman (2004).

A camera is an optical sensor for capturing 2D images of its surrounding 3D environment. The 2D images are obtained by projecting the 3D scene onto the 2D image plane. The camera modeling is important in computer vision. Many camera models have been proposed (see Hartley and Zisserman (2004)). The pinhole camera model is a simple yet the most used model for a camera with a thin lens. This model accurately captures the geometrical properties of perspective projection. This model is inspired by the simple camera: the camera Obscura. The camera has the shape of light-proof box with a infinitesimally small aperture (pinhole) on the front wall (see Fig. 2.1). Rays of

light coming from an object of the 3D scene enter the camera through the pinhole and intersect with the back wall of the camera. The back wall contains a film on which an inverted image is formed. The plane containing this film (imager) is called the image plane or retinal plane. The pinhole is referred as the center of projection. The distance between the center of projection C and the image plane π is called the focal length f. The line perpendicular to the image plane and passing through the center of projection C is known as optical axis. The optical axis intersects with the image plane at the principal point.

The pinhole camera produces an inverted image since the scene is reflected in the center of projection C. The image appears upside down. One needs to rotate the image 180° to compensate for the reflection. To avoid the inversion of coordinates, and thus simplify the mathematical formulas, it is common to consider the virtual image plane parallel to the image plane (back wall) and located in front of the camera at the distance f. In the remaining of this thesis, we will refer to this virtual image plane as the image plane.

This background section is based on the work of Moons *et al.* (2010) and the book Hartley and Zisserman (2004).

Perspective projection

The 2D projection $p = (x, y)$, through C, on the image plane of a 3D point $P = (X_c, Y_c, Z_c)$ with coordinates in the camera frame $\mathcal{C}$ can be derived via the law of similar triangles. It is given by:

$$p = \begin{pmatrix} x \\ y \end{pmatrix} = -f \begin{pmatrix} X_c/Z_c \\ Y_c/Z_c \end{pmatrix} \tag{2.15}$$

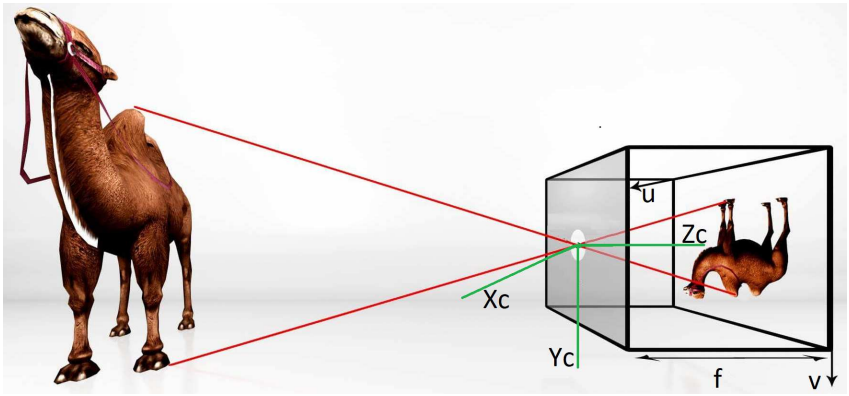


Figure 2.1: An illustration of the pin-hole camera model. We created this figure using the software *Autodesk 3ds Max*.

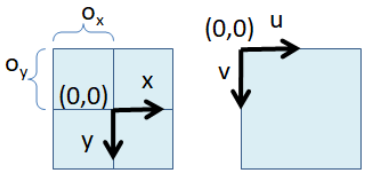


Figure 2.2: Changing the origin of the image coordinate system to the top left corner of the image.

Note that the 3D point $P = (X_c, Y_c, Z_c)$ is described by its coordinates in the camera frame $\mathcal{C}$. The general case of a point given by its coordinates in the word frame $\mathcal{W}$ is discussed later (See page 17).
The minus sign shows that the image is upside down on the (true) image plane. For simplifying the mathematical equations it is common to consider a virtual image plane on which the image is not upside down and the minus sign disappears. This virtual image plane is located between the projection center and the object at a distance f (focal length) from the center of projection (see Fig. 2.1).
The equation 2.15 can be written in matrix form by using the homogeneous coordinates as follows:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = M \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{2.16}$$

The matrix M can be decomposed into the product of two sub-matrices M_1 and M_2.

$$M = M_1 M_2 \tag{2.17}$$

Where:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} ; M_2 = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.18}$$

The matrix M_1 presents the dimension reduction (3D to 2D) and the matrix M_2 represents the scaling made by the camera.
The expression in Eq. 2.16 assumes that the principal point (o_x, o_y) is the origin of the image coordinate system. In practice, however, it is convenient to set the top left corner of the image as the origin (see Fig. 2.2). This conforms to the way an image is usually read out from the sensor: starting from the top left and reading out the image line by line.

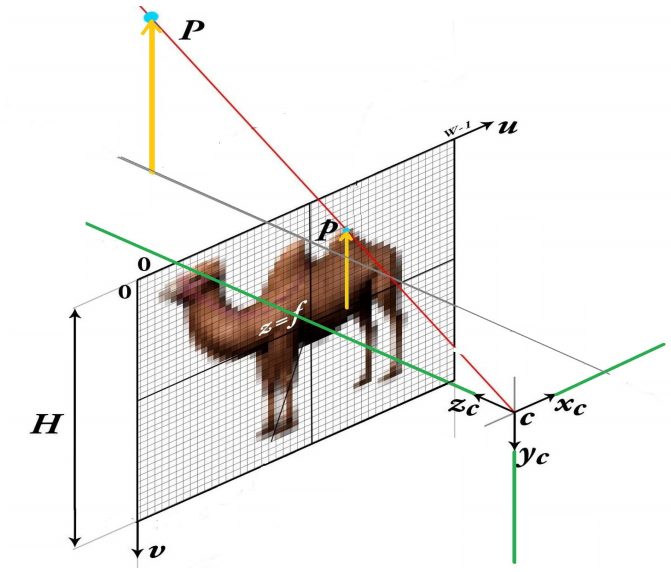


Figure 2.3: The points in 3D space are projected onto the image plane. The image is a grid of $(W \times H)$ discrete pixels. We created this figure using the software *Autodesk 3ds Max*.

The offset can be applied as follow (see Eq. 2.19):

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.19}$$

Camera intrinsic parameters

The coordinates (u_x, u_y) in the image plane are given in meters. However, the digital image has picture element (pixel) units and is composed of discrete $(W \times H)$ pixels (see Fig. 2.3). In order to convert the image of a point on the image plane to the pixel coordinates, we need to divide the coordinates by the size of a pixel. Let ρ_u and ρ_v be the width and height of a pixel, respectively. In practice, we usually work with square pixels i.e. $\rho_u = \rho_v$. The ratio ρ_u/ρ_v is called as the aspect ratio of the pixels. The pixel coordinates can be expressed (see Corke (2011)) as follows:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} 1/\rho_u & 0 & o_u & 0 \\ 0 & 1/\rho_v & o_v & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.20}$$

Camera extrinsic parameters

In our experimental platform, the camera is rigidly mounted on the quadcopter and moves as the quadcopter flies. It is convenient to represent the 3D points in the global world frame ($\mathcal{W}$). Then, for each camera pose, convert the coordinate of the 3D point into the camera centered local frame. The camera can undergo rotations around the three axes (X,Y,Z) and a translation $C_w \in \mathbb{R}^3$. The rotations can be represented by a 3×3 rotation matrix (R_w^c). In the following, we will show the mapping from the world frame $\mathcal{W}$ to camera centered frame $\mathcal{C}$.

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = R \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + t \tag{2.21}$$

The conversion of the coordinates of a point P from the world coordinates $P_w = (X_w, Y_w, Z_w)$ into the camera centered frame $P_c = (X_c, Y_c, Z_c)$ can be done by projecting the relative vector $(P_w - C_w = P_w - C)$ orthogonally onto each of the coordinate axes (r_1, r_2, r_3) of the camera local frame. This corresponds to the dot product of the matrix R_w^c with the relative position $(P - C)$.

$$P_c = R_w^c (P_w - C) \tag{2.22}$$

$$P_c = R P_w - R_w^c C \tag{2.23}$$

We note here that P_w and P_c represent the same physical point P, which is described in the two different coordinate systems $\mathcal{W}$ and $\mathcal{C}$, respectively.
The matrix R_w^c and the vector $t = -R_c^w . C_w$ represent the extrinsic parameters of the camera and can be put in a 4×4 homogenous matrix $M_{ext} = [R|t]$. As shown in Eq. 2.26.

$$P_c = R_w^c P_w + t \tag{2.24}$$

$$P_c = [R|t] P_w \tag{2.25}$$

$$M_{ext} = [R|t] = \begin{pmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.26}$$

General camera model

In this section, we will introduce the general camera model described by the camera projection matrix P_{proj}. The projection P_{proj} describes the mapping from a 3D point in an arbitrary coordinate system (the world frame $\mathcal{W}$ for convenience) into the pixel coordinates (x, y) on the image coordinate system $\mathcal{I}$. The camera projection matrix P_{proj} can be constructed by putting together the transformations given in Eq. 2.20 and Eq. 2.26 as described in Eq. 2.27 (see Corke (2011)).

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \widehat{=} \begin{pmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_W \\ Z_W \\ 1 \end{pmatrix} \tag{2.27}$$

Lens and non-linear distortions

The pinhole (aperture) of the camera has to be small to avoid image blur which occurs when the camera aperture is much larger. The blur effect happens because the light coming from a 3D point, of an object on the scene, will contribute to form multiple pixels on the image plane. On the other hand, using a very small aperture reduces the light intensity accumulated by the photo-receptors of the camera sensor. As a result, the required exposure time increases and frame rate decreases accordingly. Replacing the pinhole with a focusing lens which is both well placed and well sized allows solving the problem. A focusing lens has a larger aperture allowing more light rays to get inside the camera and focuses the light to avoid the blur (see Fig. 2.4). Furthermore, a lens allows adjusting the camera field of view. The benefits of using a focus lens come with the drawback of introducing non-linearity on the linear model of the camera perspective projection. The co-linearity property of the points P_c, C and p will no longer hold. To model this geometrical distortions introduced by the lens, we use the popular Barrel model which models the radial distortions. The radial distortion models the systematic variation of the optical magnification when radially moving away from the center of projection. The magnitude of the deviation increases as we move further away from the camera center of distortion. We also use the tangential model to model the lens misalignment in which the imager is not exactly perpendicular to the optical axis. We will always start by correcting the distortions from the camera images to be able to use the linear camera model. For that purpose we need to perform camera calibration and apply the inverse operations.

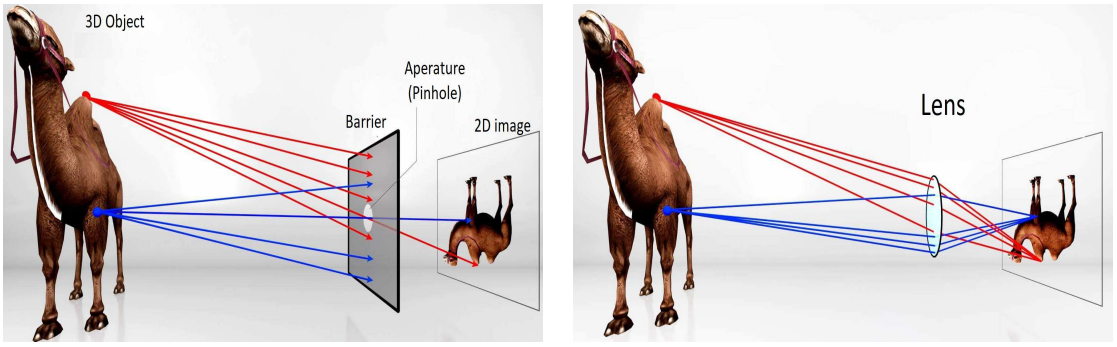


Figure 2.4: Left: A pinhole camera with no lens. Many light rays are blocked and the exposure time needs to be large to accumulate enough light intensity to form the image. Right: A camera with a thin lens. Light rays are accumulated from different directions and focused on the image sensor. Image formation becomes faster. We created these figures using the software *Autodesk 3ds Max*.

2.3 Epipolar geometry

This background section is based on the work of Moons *et al.* (2010) and the book Hartley and Zisserman (2004).

During the navigation, the quadcopter processes the sequence of images captured from different viewpoints to perform both localization and mapping. We study in this section the elementary case: two views perspective projection. Let us consider a point P in 3D space and its projections p_1 and p_2 onto two cameras C_1 and C_2, respectively (see Fig. 2.5). In the case of a static stereo, the camera center will also be referred to as C_l and C_r (for left and right) instead of C_1 and C_2. We show the geometric properties resulting from the triangle C_1C_2P. The vectors C_1C_2, C_1P and C_2P are co-planar. These properties introduce constraints on the locations of the projections p_1 and p_2. These constraints are captured by the Fundamental matrix of computer vision, the matrix F. The Fundamental matrix and its properties are presented in Sec. 2.3.1(see page 21). For convenience, we summarize here its most important properties. F states that the corresponding pixel p_2 for a pixel p_1 lies on the epipolar line Fp_1 on the image I_2. An immediate result of F is that the search for corresponding pixels reduces to 1D search instead of 2D. If the camera matrices K_1 and $K2$ are known then the Fundamental matrix F takes a special form called the Essential matrix E. The matrix E, which has less DoF than F, is very practical for estimating the relative motion $[R|t]$ between the two cameras.

The term ”two views perspective” was chosen to describe many configurations which arise in vision-based mobile robotics. In particular, it handles the following cases:

- Temporal stereo: a moving physical camera which captures two images from different locations at different times. In this case, the two views have the same calibration matrix $K_1 = K_2 = K$. The two images could be any two images from the

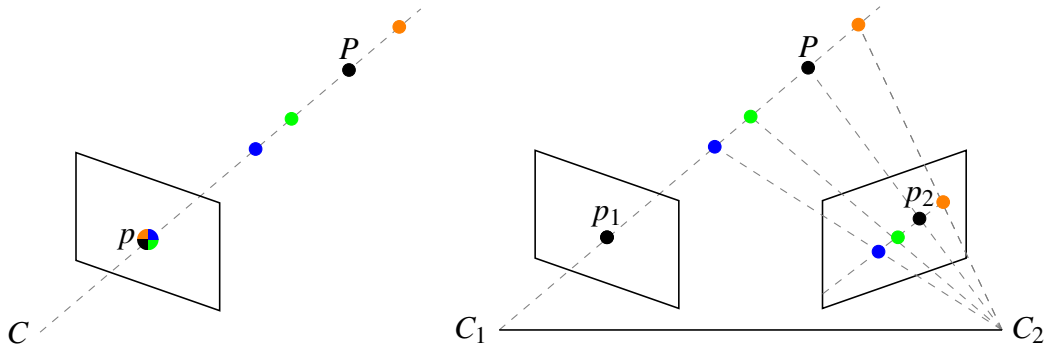


Figure 2.5: Left: monocular vision. The depth of the 3D P cannot be recovered from its projection p alone. All 3D points along the ray (CP) get projected to the same point 2D point on the image plane. Right: a second view can resolve the depth ambiguity.

sequence which have enough overlap on their field of views. Relevant cases are: consecutive image pairs and image to keyframe pairs. In our experiments, we use the two views perspective configuration for the left camera (reference camera) of our stereo camera.

- Static stereo: two physical cameras located in different positions capturing the scene at the same time. In our experiments, this configuration corresponds to our stereo camera (left and right cameras) which is mounted on the quadcopter.

The points P, p_1, C_1, C_2 and p_2 belong to the same plane, which is called the epipolar plane. The projection of the camera center C_1 onto the camera image I_2 is called the second (right) epipole e_2. The first (left) epipole e_1 is defined in a similar way. The base line C_1C_2 intersects with image planes at the epipoles e_1 and e_2.

Monocular camera

In the case of single view perspective the robot observes the scene (3D point P) using one camera. From one single image, it is not possible to determine the depth of the 3D point P given its 2D projection p_1 because all the points belonging to the ray C_1P project to the same image point p (see left image of Fig. 2.5).

Stereo camera.

A second image which is captured from a different perspective can resolve the depth ambiguity. Different points on the 3D line C_1P are projected onto different 2D locations on the second view (see right image of Fig. 2.5). The projection of the 3D line C_1P onto the second image I_2 is a 2D line which passes through the right epipole e_2. This 2D line is the epipolar line corresponding to p_1.

Two 2D epipolar lines l_1 and l_2 can be associated for each 3D point P as illustrated in Fig. 2.5. All epipolar lines (of different point on the space) pass through the epipoles.

Now that we have presented the geometric constraints induced by the epipolar geometry, we will show the resulting mathematical formulation of these geometric properties: the

Fundamental matrix F.
In the following we assume that the camera intrinsic parameters (K_1 and K_2) are known. We also assume that we work with undistorted images (I_1 and I_2) so that we can apply the general linear camera model. The projecting ray of the 2D point $p_1 = (x_1, y_1, 1)$ (given in homogeneous coordinates) is given by inverting the transformations shown in the general camera model (see Eq. 2.27 on page 17). First, we get the 3D point on the camera coordinate system using Eq. 2.28.

$$P_c = \rho_1 K_1^{-1} p_1 \tag{2.28}$$

Where $\rho_1 \in \mathbb{R}$.
Then, we get the coordinates on the world frame $\mathcal{W}$ by applying the inverse of the extrinsic transformation. Since C_1 is the position of first camera in the world coordinate system, the coordinates of P in the world coordinate system $\mathcal{W}$ can be given by Eq. 2.29 (see Moons *et al.* (2010)).

$$P_w = C_1 + \rho_1 R_1 K_1^{-1} p_1 \tag{2.29}$$

Similarly, we get the formula for P_w (see Eq. 2.30) using the second projection p_2. Where $\rho_2 \in \mathbb{R}$.

$$P_w = C_2 + \rho_2 R_2 K_2^{-1} p_2 \tag{2.30}$$

From Eq. 2.29 and Eq. 2.30, we get a system of 6 linear equations with 5 unknowns. The unknowns are X_w , Y_w , Z_w, ρ_1 and ρ_2. If we do not consider the degenerate cases, we need exactly 5 independent linear equations to uniquely solve the problem. Fortunately, p_1 and p_2 are corresponding (projecting rays must intersect) and thus we have a rank-deficient system. This means that we can uniquely solve the system. By substituting 2.29 in 2.30 (see Moons *et al.* (2010)), we get:

$$\lambda_2 p_2 = \lambda_1 K_2 R_2^T R_1 K_1^{-1} p_1 + K_2 R_2^T (C_1 - C_2) \tag{2.31}$$

Where $\lambda_1 \in \mathbb{R}$ and $\lambda_2 \in \mathbb{R}$.
Let us investigate this equation (see Eq. 2.31) as it encapsulates the Fundamental matrix of computer vision. We start from a pixel p_1 on the image I_1.

- $K_1^{-1} p_1$: describes the ray in 3D space corresponding to p_1. It has the direction $C_1 p_1$. This ray passes through P. The coordinates are given in the coordinate system of camera C_1.
- $R_1 K_1^{-1} p_1$: multiplying by R_1 transforms the coordinates of the ray to the world coordinates $\mathcal{W}$.
- $R_2^T R_1 K_1^{-1} p_1$: the ray expressed in the coordinate system of camera C_2.

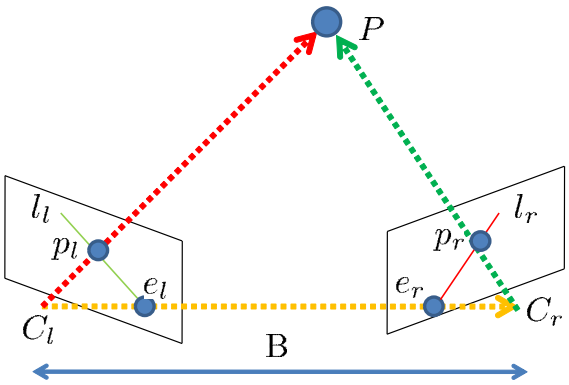


Figure 2.6: Epipolar constraint. The points P, its projections, the epipoles and the cameras centers are co-planar.

- $K_2 R_2^T R_1 K_1^{-1} p_1$: 3D to 2D transformation. The ray is projected onto the coordinate system of image I_2. The point $K_2 R_2^T R_1 K_1^{-1} p_1$ is the vanishing point corresponding to p_1. That is the endpoint of the epipolar line corresponding to p_1 on the image I_2.

- $K_2 R_2^T (C_1 - C_2) = e_2$: the epipole e_2 which is the projection of the point C_1 onto the image I_2.

- $\lambda_1 K_2 R_2^T R_1 K_1^{-1} p_1$: the distance from the epipole e_2 along the epipolar line at which p_2 is located.

We can read the Eq. 2.31 as follow: given a point p_1 on the image I_1 then, its corresponding p_2 is located at a distance λ_1 on the line starting from e_2 and having the direction $K_2 R_2^T R_1 K_1^{-1} p_1$. This is a important property from which the Fundamental matrix of computer vision can be derived.

2.3.1 The Fundamental matrix of computer vision

In Eq. 2.31, we see that the vectors p_2, e_2 and $K_2 R_2^T R_1 K_1^{-1} p_1$ are linearly dependent. Thus, their triple-product is equal to zero (see Moons *et al.* (2010)). We get:

$$p_2^T (e_2 \times K_2 R_2^T R_1 K_1^{-1} p_1) = 0 \tag{2.32}$$

By writing the vector product using the matrix expression (skew matrix $[e_2]_\times$ of vector e_2),we get:

$$9 p_2^T [e_2]_\times K_2 R_2^T R_1 K_1^{-1} p_1 = 0 \tag{2.33}$$

By defining the Fundamental matrix F as:

$$F = [e_2]_\times K_2 R_2^T R_1 K_1^{-1} \tag{2.34}$$

which is a 3×3 matrix, we get the formula for the constraints of 2D-2D correspondences.

$$p_2^T F p_1 = 0 \tag{2.35}$$

F is the Fundamental matrix of the image pair (I_1, I_2). It describes the relationship between corresponding pixels on both images. For a point p_1 on the first image, its corresponding pixel p_2 must lie on the corresponding epipolar line defined by Fp_1 on the second image (see Moons *et al.* (2010)). F and its special cases (i.e. Essential matrix) are powerful tools of computer vision. Even without reconstructing the scene, F provides constraints of how the scene changes when the scene is viewed from a different perspective. Let's now determine the degrees of freedom (DoF) of F. The Fundamental matrix F is a 3×3 matrix. We can remove two DoFs from F. One DoF can be removed from F because F can only be defined up to a scale. This is due to the fact that F is defined using homogeneous coordinates. Furthermore, F can be decomposed into a product (see Eq. 2.34) of matrices including a skew symmetric matrix ($[e2]_\times$) which is a singular matrix of rank 2 by formation. Thus, the Fundamental matrix is also singular of rank 2. This removes the second DoF from the matrix F. As a result, the 3×3 Fundamental matrix F has 7 DoFs.

In the following, we will show how to estimate F from 2D-2D pixel correspondences. Without any knowledge about the camera parameters (extrinsic and intrinsic), F can be estimated from at least 7 known point correspondences between the images I_1 and I_2 (minimal case). In practice however, usually 8 or more known points correspondences (see Longuet-Higgins (1981)) are used to avoid non-linearity of the minimal case. Assuming there are no wrong matches (outliers) on the correspondences, we can use the 8-point algorithm (see Longuet-Higgins (1981)) to estimate F. For a pair $(p_1^{(i)}, p_2^{(i)})$ of corresponding pixels, Eq. 2.35 (Viz. $p_2^T F p_1 = 0$) can be written using the form $A^{(i)} f = 0$ (see Olsson (2013)) as follows:

$$\begin{aligned} F_{11} p_{2,x}^{(i)} p_{1,x}^{(i)} + F_{12} p_{2,x}^{(i)} p_{1,y}^{(i)} + F_{13} p_{2,x}^{(i)} p_{1,z}^{(i)} \\ + F_{21} p_{2,y}^{(i)} p_{1,x}^{(i)} + F_{22} p_{2,y}^{(i)} p_{1,y}^{(i)} + F_{23} p_{2,y}^{(i)} p_{1,z}^{(i)} \\ + F_{31} p_{2,z}^{(i)} p_{1,x}^{(i)} + F_{32} p_{2,z}^{(i)} p_{1,y}^{(i)} + F_{33} p_{2,z}^{(i)} p_{1,z}^{(i)} \\ = 0 \end{aligned} \tag{2.36}$$

$$\begin{pmatrix} p_{2,x}p_{1,x} & p_{2,x}p_{1,y} & p_{2,x}p_{1,z} & \cdots & p_{2,z}p_{1,z} \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ . \\ . \\ . \\ F_{33} \end{pmatrix} = 0 \tag{2.37}$$

Eq. 2.37 shows that a pair of corresponding point introduces a new constraint on the entries of F. Therefore, we need 8 correspondences to estimate F (up to a scale) in a linear way. Let's define the vector $f = (F_{11}, F_{12}, F_{13}, ..., F_{33})^T$ from the elements of the matrix F. By stacking the equations from the n available pairs of corresponding pixels (see Olsson (2013)), we construct a system ($Af = 0$) of linear equations:

$$\begin{pmatrix} p_{2,x}^{(1)}p_{1,x}^{(1)} & p_{2,x}^{(1)}p_{1,y}^{(1)} & p_{2,x}^{(1)}p_{1,z}^{(1)} & \cdots & p_{2,z}^{(1)}p_{1,z}^{(1)} \\ p_{2,x}^{(2)}p_{1,x}^{(2)} & p_{2,x}^{(2)}p_{1,y}^{(2)} & p_{2,x}^{(2)}p_{1,z}^{(2)} & \cdots & p_{2,z}^{(2)}p_{1,z}^{(2)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{2,x}^{(n)}p_{1,x}^{(n)} & p_{2,x}^{(n)}p_{1,y}^{(n)} & p_{2,x}^{(n)}p_{1,z}^{(n)} & \cdots & p_{2,z}^{(n)}p_{1,z}^{(n)} \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ . \\ . \\ . \\ F_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ . \\ . \\ . \\ 0 \end{pmatrix} \tag{2.38}$$

When $n > 7$ pairs of corresponding pixels are available, F can be estimated by solving the system of linear equations $Af = 0$. One can solve $\text{argmin}_f |Af|^2$ subject to $|f| = 0$. The coordinates for x (640 pixels) and y (480 pixels) are usually orders of magnitude larger than z ($z = 1$ in the homogeneous coordinates). For better numerical stability, it is therefore recommended to normalize the coordinates (see Hartley (1997)). Due to noise on the pixel locations, the estimated F_{init}, which solves the linear system, might not fulfill the constraint $Det(F_{init}) = 0$. Due to noise the smallest eigenvalue of the SVD of F_{init} can be very small but non-zero. Therefore, we calculate the final F from F_{init} by forcing $Det(F) = 0$. This can be done using the SVD (see Eq. 2.39) of F_{init} and setting the smallest eigen value to "0" (see Eq. 2.40). This guaranties that the corrected F is the closest (Frobenious norm) matrix to the initial F_{init} which fulfills the constraint $Det(F) = 0$ (see Olsson (2013)).

$$F_{init} = U diag(\sigma_1, \sigma_2, \sigma_3) V^T \tag{2.39}$$

The final Fundamental matrix F can be given by Eq. 2.40:

$$F = U diag(\sigma_1, \sigma_2, 0) V^T \tag{2.40}$$

Estimating F from 8 pixel correspondences gives an estimate which is very sensitive to wrong correspondences (outliers). In many applications, it is very cheap to detect

and match more than 8 corresponding pixels (hundreds or even thousands of correspondences) using image processing techniques. Intuitively one can think that with more correspondences we can find a way to get a better estimation of F. In the literature there are two major approaches for this: using a set of 8-points correspondences in a RANSAC scheme (see Fischler and Bolles (1981)). Or using least squares on all available correspondences to find the F which best fits the correspondences.

As can be seen in Eq. 2.34, the Fundamental matrix F encapsulates information about the intrinsic and extrinsic parameters. Since F can be estimated from the images alone, it can actually be used to estimate some of those parameters. In total, we have 22 DoFs to estimate the full calibration of the two cameras. K_1 (5 DoFs), K_2 (5 DoFs), $[R_1|C_1]$ (3+3 DoFs), $[R_2|C_2]$ (3+3 DoFs). But the Fundamental matrix F has only 7 DoFs. If no additional knowledge about the cameras or about the scene points or no additional assumptions are made then, the estimated parameter will be highly ambiguous. Examples of those ambiguities are the 3D reconstruction up to some homography transformations or even some projective transformations (see Hartley and Zisserman (2004)). We are, however, interested in "unique" Euclidean reconstructions on our experiments (see Wurm *et al.* (2010)).

2.3.2 The Essential matrix

Additional DoFs can be removed from the system ($p_2 F p_1 = 0$) if some prior knowledge is given. For example, if the cameras are individually offline calibrated such that K_1 and K_2 are known, then the Fundamental matrix takes a special form called the Essential matrix E. Let us consider q_1 and q_2 such that:

$$\begin{aligned} q_1 &= K_1^{-1} p_1 \\ q_2 &= K_2^{-1} p_2 \end{aligned} \tag{2.41}$$

By substituting Eq. 2.41 in Eq. 2.34. We get:

$$q_2^T . ([t]_x R q_1) = 0 \tag{2.42}$$

By defining the Essential matrix E as:

$$E = [t]_x R \tag{2.43}$$

The epipolar constraint can be written using E as given by Eq. 2.44. This is similar to its expression using the Fundamental matrix (see Eq. 2.35).

$$q_2 E q_1 = 0 \tag{2.44}$$

In this case, the unknown is the relative motion ($[R|t]$) between the cameras. The Essential matrix has 5 DoFs. The relative motion ($[R|t]$) can only be determined up to a scale from the Essential matrix E. This matrix is at the core of the 2D-2D correspondences motion estimation methods. Many works for visual odometry and SLAM rely on E and they differ only on the type of the features, the descriptors, the feature matching algorithms and on the techniques for increasing the robustness against outliers.

2.3.3 Extracting the motion from the Essential matrix

The original 8-point algorithm for motion estimation has been presented by Longuet-Higgins (1981). We will review the important aspects of this algorithm. First, the Essential matrix E is estimated using a similar method as the estimation of F (see page 22). Then, the motion parameters can be extracted by uniquely decomposing E into a matrix product $[t]_\times R$.
Longuet-Higgins (1981) proposed to perform the decomposition using the following 3×3 helper matrices: Z and W (see Eq. 2.45).

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} ; Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.45}$$

Having only 5 DoFs, it can be shown that the SVD decomposition of E has the following form (see Eq. 2.46).

$$E = U diag(1,1,0) V^T \tag{2.46}$$

Let us set S_1, S_2, R_1 and R_2 such that: ($S_1 = -UZU^T$, $R_1 = UW_TV^T$) and ($S_2 = UZU^T$, $R_2 = UWV^T$).
There are two possible solutions $E = S_1R_1$ and $E = S_2R_2$ for the decomposition of $E = [t]_\times R = SR$. In the following, we will show the verification that the above two factorizations are valid. We also verify that R_1 and R_2 are rotation matrices and verify that S_1 and S_2 are skew-symmetric matrices.
It is easy to check that $ZW = diag(1,1,0)$ and $ZW^T = -diag(1,1,0)$.
To verify that $E = S_1R_1$ we check

$$\begin{aligned} S_1R_1 =& -UZU^TUW_TV^T \\ =& -UZW_TV^T \\ =& -U(-diag(1,1,0))V^T \\ =& E \end{aligned} \tag{2.47}$$

We can verify that R_1 is a rotation matrix. We check that R_1 fulfills the two conditions $R_1^T R_1 = I_{3\times 3}$ and $Det(R_1) = 1$.

$$\begin{aligned} R_1^T R_1 =& (UW_T V^T)^T UW^T V^T \\ =& VWU^T UW^T V^T \\ =& I_{3x3} \end{aligned} \tag{2.48}$$

and:

$$\begin{aligned} Det(R_1) =& Det(UW^T V^T) \\ =& Det(U)Det(W^T)Det(V^T) \\ =& Det(W)Det(UV^T) \\ =& 1 \end{aligned} \tag{2.49}$$

We can verify that S_1 is a skew symmetric matrix by checking that $-S_1^T = S_1$.

$$\begin{aligned} -S_1^T =& (UZU^T)^T \\ =& UZ^T U^T \\ =& -UZU^T \\ =& S_1 \end{aligned} \tag{2.50}$$

Since t can only be extracted up to a scale any $t_\lambda = \lambda t$ where $\lambda \in \mathbb{R}^*$ is also valid. The vector product $t_\lambda \times t = 0$ can be written in matrix form as $[t_\lambda]_\times t = St = 0$. So, t is in the null-space of the matrix S. The third column u_3 of the matrix U gives a solution for t. It is worth to notice that the sign of λ is important as it determines if the 3D points will be reconstructed in front of the cameras or behind the cameras. We need to check both $t = u_3$ and $-t = -u_3$ as they are both valid solutions.
The result is four possible mathematical solutions (see Olsson (2013)) for the decomposition of E into $[t]_\times R$ where:

$$[R|t] = \begin{cases} [UWV^T \quad | \quad u_3], \quad for \quad \lambda = 1 \\ [UW^T V^T \quad | \quad u_3], \quad for \quad \lambda = 1 \\ [UWV^T \quad | -u_3], \quad for \quad \lambda = -1 \\ [UW^T V^T \quad | -u_3], \quad for \quad \lambda = -1 \end{cases} \tag{2.51}$$

Among the four solutions, only one is valid in practice and guarantees that all reconstructed 3D points are located in front of both cameras. This is illustrated in Fig. 2.7. In this example, the valid solution is the configuration on the top left image.

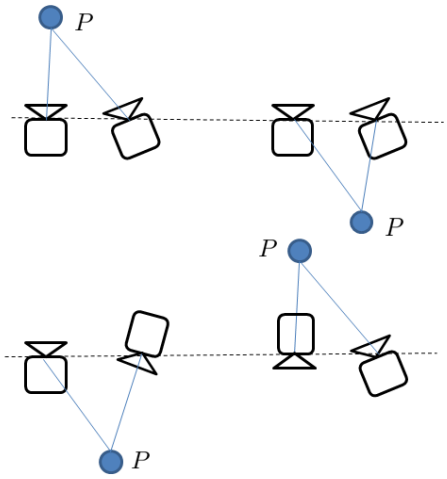


Figure 2.7: Illustration of the four mathematical solutions for extracting the motion $[R|t]$ from the Essential matrix E. Only one solution (top left) is valid in practice.

2.4 The PnP algorithm

We have seen in the previous section that 2D-2D correspondences can be used to estimate the camera pose. In this section, we review a conceptually different method for pose estimation: the perspective n-points (PnP) algorithm which relies on 3D-2D correspondences. Fig. 3.3 (on page 45) shows an illustration of 3D-2D, 2D-2D and 3D-3D correspondences. PnP can be used in the case there is prior knowledge about the geometry of the scene in form of a set of 3D points with known coordinates and known projections (pixel coordinates and associations) on the image. The minimal solution of PnP which provides a finite set of solution is achieved when $n = 3$ and called P3P algorithm. The origin of the P3P algorithm dates to the 17 century (see Grunert (1841)). Fischler and Bolles (1981) presented the RANSAC algorithm as well as a method for solving the P3P problem. They showed an application of their algorithm to estimate the camera pose using RANSAC from many P3P hypotheses. They pointed out that as many as four solutions can be found and by using a fourth 3D point (P4P), the best solution can be identified. It is worth to notice that Fischler and Bolles (1981) formulated the PnP problem as the problem of determining the lengths of line segments joining the camera pose and the 3D points given the relative locations of the 3D points and the angles to every pair of the 3D points. Fig. 2.8 shows an illustration of the P3P problem. In this illustration, the camera is located at position P. The camera sees the three 3D points A, B and C. The projections onto the image plane of A, B and C are the pixels a, b and c, respectively. Let us consider the tetrahedron which has the point P as the apex and has the triangle ABC as base. Let's define the variables PA, PB and PC as the lengths of line segments joining the camera center P and the 3D points A, B and C, respectively. Let's also define the angles $\alpha = \sphericalangle APB$, $\beta = \sphericalangle APC$ and $\gamma = \sphericalangle BPC$. The relative distances AB,

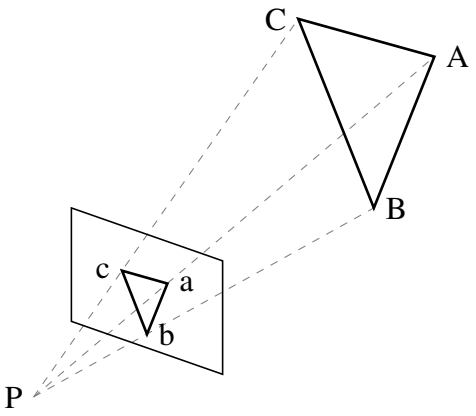


Figure 2.8: Illustration of the P3P problem.

AC and *BC* are given and the angles α, β and γ can be easily retrieved from the 3D-2D correspondences. The following system of equations (see Eq. 2.52) can be derived from the law of cosines.

$$\begin{aligned}(AB)^2 &= (PA)^2 + (PB)^2 - 2(PA)(PB)cos(\alpha)\\ (AC)^2 &= (PA)^2 + (PC)^2 - 2(PA)(PC)cos(\beta)\\ (BC)^2 &= (PB)^2 + (PC)^2 - 2(PB)(PC)cos(\gamma)\end{aligned} \tag{2.52}$$

Let's define x such that $(PB) = x * (PA)$. Fischler and Bolles (1981) reduced the system of equations in Eq. 2.52 into a bi-quadratic polynomial in one unknown x, where the unknown x represents the ratio of two legs of the tetrahedron PABC. Fischler and Bolles (1981) bi-quadratic equation in the unknown x has the form given in Eq. 2.53.

$$G_4x^4 + G_3x^3 + G_2x^2 + G_1x + G_0 = 0 \tag{2.53}$$

Where $G_i \in \mathbb{R}$ for $i = \{0,1,2,3,4\}$.
Eq. 2.53 can be solved either in closed form (see Dehn (1960)) or by iterative techniques (see Conte and de Boor (1980)). The camera orientation can be estimated when additionally the focal length of the camera and the principal point are known. We refer the interested reader to Fischler and Bolles (1981) for a detailed description of the PnP algorithm. A variant of the P3P algorithm which is faster to compute and more accurate is proposed by Masselli and Zell (2014). They designed a new geometric parametrization to solve the P3P problem.

2.5 Stereo matching

We will introduce in this section the background of the stereo correspondence problem. For static stereo, we will refer to the image I_1 as the left image I_l and the image I_2 as the right image I_r. For a reference pixel $p_l = (x_l, y_l)$ on the left image I_l, we seek to find the location of its corresponding pixel $p_r = (x_r, y_r)$ on the right (target) image I_r.

We distinguish between dense stereo matching and the sparse (feature-based) stereo matching. In dense stereo matching, we seek to estimate the correspondences for all pixels on the reference. The result of dense stereo matching is a disparity map. To deal efficiently with the large amount of pixels, usually simple feature vectors are used for computing the similarity measure for candidate matches. In chapter 4 (page 69), we will introduce our dense stereo matching algorithm (LS-ELAS). On the other hand, in the sparse stereo matching, we seek to establish correspondences for a small set of pixels on the reference image. For example, one usually extracts the features using a sophisticated feature or blob detector such as FAST or SIFT and wants to find the correspondences of those features.

2.5.1 Challenges and assumptions

Looking for correspondences can be quite challenging. Among these challenges are the half occlusion, rank order, fore-shortening effect, texture-less areas. We will elaborate on these challenges and show how the correspondence problem can be simplified based on the properties of the epipolar geometry and some assumptions.

Epipolar constraint. The epipolar geometry (see Eq. 2.35) constrains the correspondence search from 2D to 1D. This results in a huge reduction of the computational time. Looking for the corresponding pixel along the line on which it is located (the epipolar line) increases the robustness against false matches.

Photo-constancy. Most matching approaches rely on the photo-constancy assumption: the assumption that the light intensity reaching a pixel on the reference image is equal to the intensity reaching its corresponding pixel on the target image. In other words, the incident light on a surface of an object is assumed to follow the Lambertian reflectance model. This assumption can be violated in many cases but it remains a useful assumption.

Half-occlusions. For points which are only visible for one camera, looking for their correspondences in the other image (target image) is an ill-posed problem. This is illustrated in Fig. 2.9. The points on the segment AB are visible by the left but they are not visible for the right camera. The explicit detection of the half-occluded pixels is important in many applications and a number of approaches has been developed for dealing with occluded pixels (see Yang *et al.* (2009)). A simple and very effective approach is the Left-right consistency check. The idea behind this technique is to compute two disparity maps, each time one image is considered as the reference image and the other image as target. Then, check if the disparities agree.

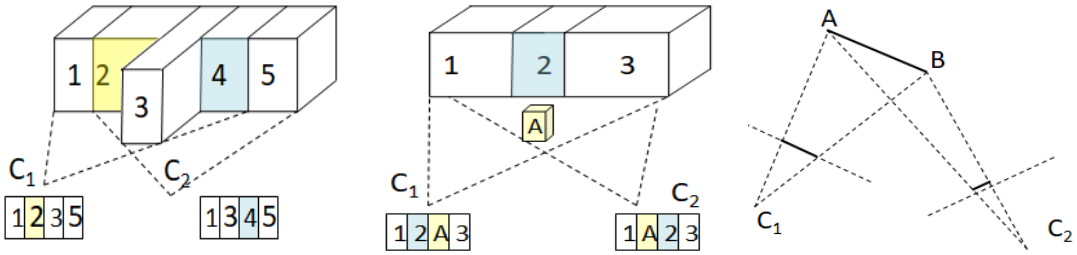


Figure 2.9: Illustration of some stereo matching challenges. Left: half occlusions. The number "4" is occluded for the camera C_1 and the number "2" is occluded for the camera C_2. Middle: a scenario where the ranking order is not uphold. The camera C_1 sees "2A" while the camera C_2 sees "A2". Right: foreshortening effect. The image of the segment AB is represented using fewer pixels on camera C_2 compared to camera C_1.

Rank order assumption. We assume that corresponding pixels have the same rank order in both images. In some cases, this cannot be true due to occlusions. Fig. 2.9 shows an illustration of a case where the order assumption does not hold.
Weak texture areas. To characterize a pixel on an image, one often needs to have some significant intensity changes on the pixel's surrounding. In the case of weak texture, the search can be ambiguous as many candidate matches might have similar matching scores. Moreover, the true corresponding pixel might have a lower score, of the similarity measure, than the neighboring candidate pixels.
Repetitive patterns. Another challenge of correspondence problem is the existence of repetitive patterns on the target image. Here, even if there is enough texture change, the search can still be ambiguous. As in the case of weak texture around the pixel, many candidate matches might have a similar matching scores.
Foreshortening effect. Since the scene is observed from two different viewpoints, some objects might appear shorter on one image compared to the other image. This case is illustrated in Fig. 2.9. On the left view, the scene object is represented by more pixels than on the right image. Even if there is no occlusion in this case, one may not find a one-to-one mapping for the correspondences. To simplify the correspondence search problem, we assume that there is at most one corresponding pixel.
Stereo rectification. To simplify the search of the correspondences the canonical configuration, is often used for dense stereo matching. In this configuration, the two identical cameras are perfectly aligned to be co-planar. The only motion between the two cameras is a translation along the $X-axis$ (the baseline B). The result is that the two image planes coincide (plane π) and the baseline is parallel to the image planes. The benefit of using the canonical configuration is that the epipolar lines are horizontal and have the same y coordinate. The epipolar line l_r, on the target image, corresponding to a reference point $p_l = (x_l, y_l)$ on the left image is a horizontal line with Y coordinate equal to y_l. The epipoles e_r and e_l are located at infinity since the image planes are parallel to the baseline.

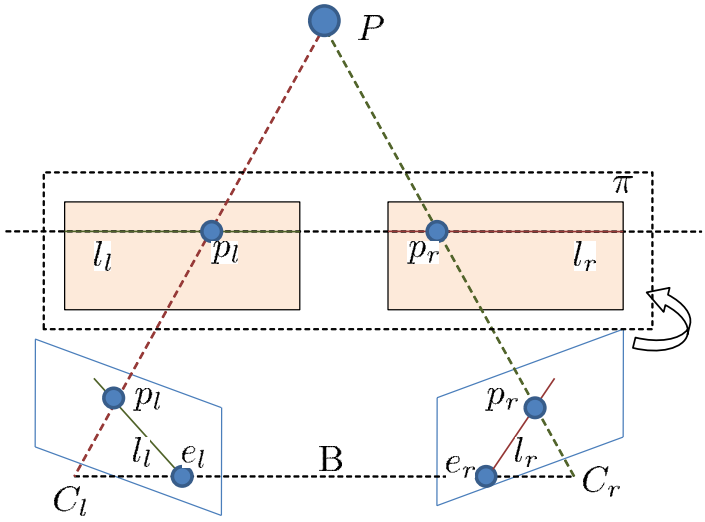


Figure 2.10: Illustration of the rectification process. The stereo pair is projected using the homographies onto a common virtual plane which is parallel to the baseline. On this virtual plane, only the coordinates along the $X-axis$ differs between corresponding pixels.

Since it is technically hard to set up a stereo pair in the canonical configuration geometry, we transform the images to simulate the canonical configuration. This process is called stereo rectification (see Zhang (2000)). It computes from the calibration parameters two 3×3 homography transformations that can be applied to rectify the stereo pair.

2.5.2 Depth from disparity

On a rectified stereo pair, the disparity d of a pixel $p_l = (x_l, y_l)$ refers to the apparent pixel difference or motion on the second image. The location of the corresponding point $p_r = (x_r, y_r)$ can be written as a function of the coordinates of p_l as follows: $p_r = (x_l + d, y_l)$. The disparity is proportional to the inverse depth. The disparity can be derived from the depth (based on similar triangles) and vice versa (see Eq. 2.54).

$$d = x_r - x_l = \frac{Bf}{Z}; Z = \frac{Bf}{d} \tag{2.54}$$

For a calibrated camera, the full coordinate (X,Y,Z) of a point P in 3D-space can be computed if its 2D projection p_l and the disparity d are known (see Eq. 2.55).

$$P = (X,Y,Z) = (X,Y,\frac{Bf}{d}) \tag{2.55}$$

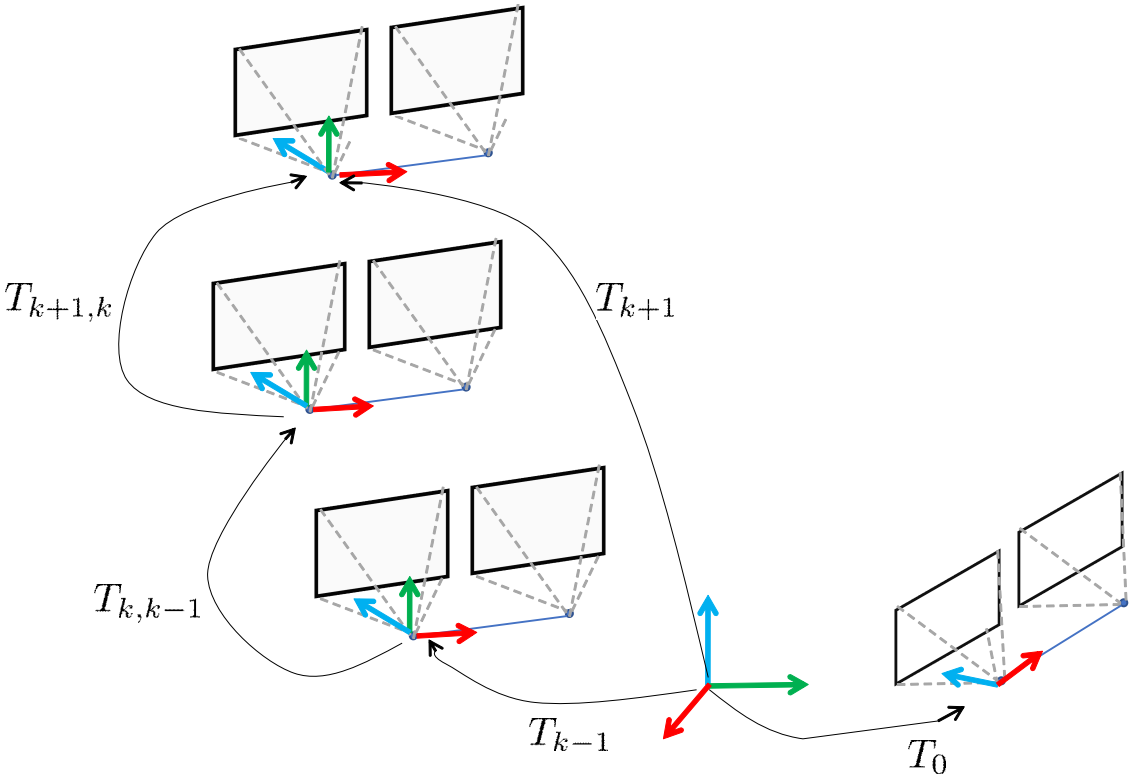


Figure 2.11: Stereo visual odometry. Concatenating the elementary relative transformations to recover the full trajectory.

2.6 Stereo visual odometry and SLAM

We have seen how to estimate the relative pose $T_{k,k-1}$ between two camera poses T_k and T_{k-1}. The absolute pose T_k at time k can be computed incrementally by concatenating the elementary relative transformations, in order to recover the full robot trajectory (see Fig. 2.11). Thanks to the homogeneous coordinates, the concatenation can be done by simple matrix multiplication as given by Eq. 2.56.

$$T_k = \begin{pmatrix} R_{3x3,k} & t_{3x1,k} \\ 0_{1x3} & 1 \end{pmatrix} = T_{k,k-1}T_{k-1,k-2}...T_{3,2}T_{2,1}T_{1,0}T_0 \tag{2.56}$$

Moravec (1980) has presented one of the first approaches to estimate the robot pose from camera only. The camera used in his work is mounted on a rail. The robot performs stop-and-go motion. At each stop, the camera moves along the rail to capture 9 images at equi-distant positions. The image set simulates a multi-baseline stereo camera. Most importantly, Moravec has proposed a pipeline for motion estimation which is still adopted today by many researchers.
Modern visual odometry algorithms perform local bundle adjustment (LBA see page 35)

to increase their performance.

In mobile robotics, the robot localization and the environment mapping are two important tasks. The localization refers to the task of estimating the robot pose on an a-priori given map. The mapping refers to the task of constructing a map of the robot environment using some measurements and some known robots poses. To navigate autonomously in unknown environments there is a need to simultaneously perform both tasks. Fortunately, there exists a way to achieve this goal: the Simultaneous Localization and Mapping (SLAM). SLAM can incrementally build a map of an environment and at the same time use this map to deduce the robot location within the map. SLAM has been successfully used on many different fields on mobile robotics wheeled robots, aerial robotics and underwater robotics.

2.6.1 Visual SLAM and SFM

Early work on SLAM was done using ground robots equipped with laser scanners. Currently, cameras are widely used for SLAM. This has led to establishing a bridge between the SLAM problem on robotics and the Structure from Motion (SFM) problem of computer vision. Both visual SLAM and SFM seek to estimate the camera motion by modeling the unknown robot environment. However, SFM focuses more on the estimation of accurate 3D models of the environment. Thus, SFM favors the use of batch optimization methods using all poses T_i and landmark X_j to solve the problem. SFM optimization methods often rely on bundle adjustment. On the other hand, SLAM focuses on the incremental update of the map. As the robot moves, the controller needs to know the current robot location. Thus, sequential filtering (EKF, Particle filter) approaches were favored over batch optimization techniques to solve the SLAM problem. The most accurate results are achieved by bundle adjustment. Strasdat *et al.* (2012) showed that per unit of computation, bundle adjustment is more efficient than filtering approaches. Recent keyframe-based visual SLAM algorithms are also based on bundle adjustment and they can run in real-time on mobile robots. The breakthrough was done by Klein in his work PTAM (see Klein and Murray (2007)). Klein used a dual core CPU to run two threads in parallel one for localization (tracking) and one for mapping. The tracking thread needs to achieve frame rate processing while the mapping thread only needs to finish processing before the next keyframe arrives. Decoupling the tracking from the (slow) mapping allowed achieving real-time performance. PTAM became a standard algorithm for visual navigation and researchers build many SLAM algorithms upon PTAM.

2.6.2 Solving the SLAM problem

Let us consider a robot moving in a scene and denote the robot poses $T_i \in SE3$. The robot observes the visual landmarks X_j (see Fig. 2.12). An observation z_{ij} denotes that the landmark $X_j \in \mathbb{R}^3$ is seen at robot pose T_i. Given a set of observations (z_{ij}), we seek to estimate the robot poses T_i and the landmark positions X_j. Since there are uncertainties in

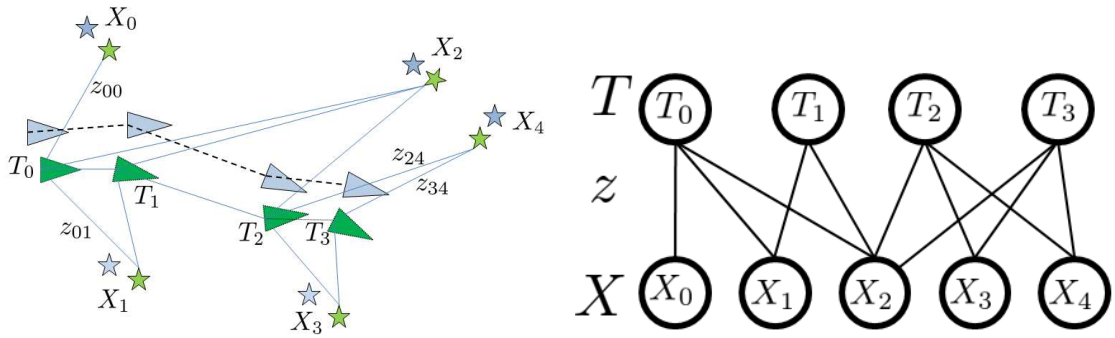


Figure 2.12: Left: An illustration of a SLAM problem with 4 robot poses $\{T_0, T_1, T_2, T_3\}$ and 5 landmarks $\{X_0, X_1, X_2, X_3, X_4\}$. Right: the graph representation of this SLAM problem using Markov Random Fields.

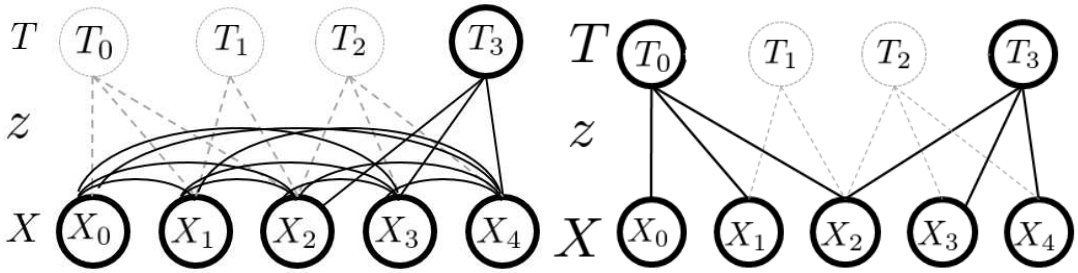


Figure 2.13: Left: illustration of the keyframe approach for SLAM. Right: illustration of the filtering approach for SLAM.

the measurements, probability theory has been adopted to formulate the SLAM problem. Probabilistic SLAM seeks to estimate the posterior probability of the map X_j and poses T_i conditioned on the measurements z_{ij}.
As the robot moves, the graph will grow by adding new poses T_i and measurements z_{ij} (see Fig. 2.12). Whenever a new part of the environment is explored for the first time, new landmarks nodes X_j will also be added to the graph. This introduces the challenge of how to efficiently deal with large scale SLAM problems. To get an estimate of the solution of the SLAM problem two main solutions are proposed: incremental filtering and keyframe based bundle adjustment.

Filtering approaches

The filtering approaches are based on marginalizing out past poses and summarizing the measurements over time. One assumes that for estimating the state (T_i, X_j) at time k one only needs to know the state at time $k-1$ and the results of the measurements at time k (Markov property). Previous nodes can then be deleted from the graph. The consequence

is a denser graph as new links between pairs of the variables X_j need to be inserted into the graph. For eliminating the node T_{k-1} we need to insert new links between each pair (X_{j1}, X_{j2}) of landmarks where X_{j1} and X_{j2} are both seen from the pose T_{k-1}. The marginalization of old poses for filtering approaches introduced heavily interconnected landmark nodes. This introduces poor scalability of the SLAM system. An illustration of the resulting graph is shown in Fig. 2.13.

Keyframe based bundle adjustment

This background section is based on the work of Pollefeys *et al.* (2014).

Bundle adjustment is the optimal solution to the visual SLAM problem as it solves for the maximum likelihood solution by taking into account all measurements from all camera poses. The optimization usually minimizes the photometric re-projection error (see Eq. 2.57) to estimate the optimized structure and motion $[R_i|t_i]_{1:n}$ from an initial estimate.

$$z_{ij} = x_{ij} - \pi(R_i X_j + t_i) \tag{2.57}$$

Robust cost functions (iteratively re-weighted least squares) are usually used to account for outliers in the measurements. This optimization problem can be solved using robust non-linear least squares as discussed in section 2.1.1 (page 9). Eq. 2.58 gives the mathematical formula for the bundle adjustment. We assume that the first camera pose is the reference coordinate system and we exclude it from the optimization.

$$(T_{2:n}, X_{1:m})^* = \arg\min_{T_{2:n}, X_{1:m}} \sum_{i=1}^{n} \sum_{j=1}^{m} \rho\left(\|x_{ij} - \pi(R_i X_j + t_i)\|_{\Omega_{ij}}\right) \tag{2.58}$$

Where ρ is a robust cost function such as the Huber cost function.
Most SLAM algorithms rely on the efficient Levenberg-Marquardt algorithm and its variants for solving the bundle adjustment problem in Eq. 2.58. In the case of the original Levenberg-Marquardt algorithm, the bundle adjustment problem corresponds to solving the normal equations given in Eq. 2.9 (viz. $(J^\intercal J + \lambda I)\delta x^* = -J^\intercal e(\breve{x})$ see page 11).
The keyframe based bundle adjustment for feature-based visual SLAM approaches are based on three kinds of sparsity, which are as follows:

1. Sparsity of the graph because of discarding some nodes. In fact, only an approximation of the bundle adjustment problem is solved. The original graph is approximated by a sparse graph using a sparsification process.

2. Sparsity of the graph because of its bipartite nature which allows the usage of the Schur complement trick (see Emilie (1968)). There are no edges between nodes of the same type. Egdes (measurements) are only relating poses with landmarks.

3. Sparsity of the graph because usually not all points are visible in every camera pose. This allows the usage of sparse matrix manipulation. The Jacobian (J) has only a few non-zero entries. A measurement adds two rows to the Jacobian matrix. These two rows are everywhere equal to zero except at the columns corresponding to the camera pose at which the measurement is made and the columns corresponding to the landmark. The matrix $J^T J$ has also a sparse block structure.

Naive implementations of bundle adjustment based SLAM which do not exploit these sparsity properties might become computationally very expensive and memory demanding as the graph grows. In the following, we will describe the afore-mentioned three levels of sparsity.
The sparsification of the graph by simply discarding some pose nodes (see Fig. 2.13) is a technique for reducing the computational cost. In many cases, it is not needed to take into account all available poses and landmarks. A strategy to carefully discard nodes from the graph has to be designed. Typically, an incoming frame becomes a keyframe node on the graph only if it includes information about new parts of the scene. Whenever a new keyframe is created the map has to be updated. The map update optimization is performed only on the small subset of the nodes which are retained.
Egdes (measurements) are only relating camera pose nodes with landmark nodes. Thus, the Jacobian matrix can be split into two parts as follow:

$$J = \begin{pmatrix} J_c & J_s \end{pmatrix} \tag{2.59}$$

Where J_c is the Jacobian with respect to the camera poses and J_s is the Jacobian with respect to the 3D landmarks. The index "c" refers to the "camera poses" and the index "s" refers to the "scene geometry" (the 3D landmarks). As a result, the matrix $J^T J$ has a block structure composed of 4 blocks as given in Eq. 2.60. Each of the two blocks $J_c^T J_c$ and $J_s^T J_s$ depend only on one type of parameters.

$$J^T J = \begin{bmatrix} J_c^T J_c & J_c^T J_s \\ J_s^T J_c & J_s^T J_s \end{bmatrix} = \begin{bmatrix} H_s & H_{sc} \\ H_{sc}^T & H_c \end{bmatrix} \tag{2.60}$$

$$\begin{bmatrix} H_s & H_{sc} \\ H_{sc}^T & H_c \end{bmatrix} \begin{bmatrix} \delta_s \\ \delta_c \end{bmatrix} = \begin{bmatrix} b_s \\ b_c \end{bmatrix} \tag{2.61}$$

This partitioning of the matrix $J^T J$ allows using the Schur Complement trick (see Emilie (1968)). The idea is that we split the large system into two small systems. First, we solve for the camera parameters (δ_c). Then, we solve for the structure parameters (δ_s) using substitution of δ_c. While it is possible to start by solving for the structure parameters, there are usually less camera parameters than structure parameters and thus more efficiency can be gained by starting with the camera parameters.

$$\begin{bmatrix} I & H_{sc} \\ -H_{sc}^T H_s^{-1} & I \end{bmatrix} \tag{2.62}$$

By multiplying Eq. 2.61 by Eq. 2.62, we obtain Eq. 2.63.

$$\begin{bmatrix} H_s & H_{sc} \\ 0 & H_c - H_{sc}^T H_s^{-1} H_{sc} \end{bmatrix} \begin{bmatrix} \delta_s \\ \delta_c \end{bmatrix} = \begin{bmatrix} b_s \\ b_c - b_s H_{sc}^T H_s^{-1} \end{bmatrix} \tag{2.63}$$

The matrix $H_c - H_{sc}^T H_s^{-1} H_{sc}$ is called the Schur complement of the matrix $J^T J$. Now, solving for the camera parameters δ_c can be done by solving $(H_c - H_{sc}^T H_s^{-1} H_{sc})\delta_c = b_c - H_{sc}^T H_s^{-1} b_s$. The matrix H_s^{-1} is a block diagonal matrix and thus it is easy to invert, which facilitates the computation of δ_c. Solving for the structure parameters δ_s can be done by backward substitution. δ_c and δ_s are the components of the solution of the bundle adjustment problem (Eq. 2.58).

Chapter 3

Hybrid SLAM by combining sparse features with direct image alignment

In this chapter, we address the problem of localization and mapping (SLAM) in 3D using a stereo camera. This algorithm is used in chapter 5 as part of a large system for autonomous outdoor obstacle avoidance using the MAV. Our SLAM algorithm is an extension of the popular ORB-SLAM2 algorithm (see Mur-Artal and Tardós (2016)). Large parts of this work have been pre-published in Ait Jellal and Zell (2017).

3.1 Introduction

ORB-SLAM2 relies on sparse ORB features. We added to ORB-SLAM2 a refinement step which relies on (semi-dense) direct image alignment. Our algorithm combines feature based SLAM and direct image alignment SLAM, we therefore call it hybrid stereo SLAM. The choice of extending ORB-SLAM2 is justified by its high accuracy, its efficiency, and its capability to map large scale environments.
Depth images are needed for our refinement step. We present in chapter 4 LS-ELAS, our stereo matching algorithm, which we use for depth estimation. Depth maps estimation is relatively computationally expensive. So, we perform dense stereo matching only at the keyframes of our hybrid SLAM. Accordingly, the refinement of the pose of a given frame is done with respect to its reference keyframe. For easy reading of this thesis, we introduce the SLAM algorithm before the stereo matching algorithm.

3.2 Motivation

Our motivation to combine a feature-based method and a direct image alignment method is twofold: first, in some aspects feature-based methods and direct methods are complementary. So, by combining them they compensate for each other's drawbacks. For example, when the stereo camera has moved over large distances between two frames the direct methods might diverge. This is because direct methods usually need a good initial guess which in case of large camera movement may not be satisfied. In the best case,

they might need considerably more iterations to converge. This might make real-time operation difficult to achieve. Using a coarse-to-fine approach and/or a motion model might help in some cases. On the other hand, the feature-based approach can deal very efficiently with large camera movements. This can be attributed to the development of descriptors which are invariant to a range of geometric (i.e. view-point) and photo-metric changes. Having to deal with very sparse features, we can afford to enlarge the search domain for correspondences, while still keeping real-time capabilities. The benefit of using direct methods rather than feature based methods is in cases of degraded image quality such as camera defocus and motion blur. In these cases, feature extraction might fail, causing tracking loss while the image alignment would give reasonably good motion estimation.
Second, we want to use additional information from the images and not only abstract them to a set of features. We also want to benefit from the relatively computationally expensive dense stereo matching not only for building a volumetric (chapter 5) map but also for refining the tracking poses.

3.3 Related work

Most of the visual-SLAM-based systems which have been developed for autonomous MAVs, and for mobile robotics in general, are feature-based. The PTAM framework has been the base of many efficient visual SLAM algorithms for MAVs. Scherer *et al.* (2012b) extended PTAM by using the inverse depth using an RGB-D camera. PTAM has the drawback of being limited to a small environment. Mur-Artal and Tardós (2016) proposed ORB-SLAM2, an efficient feature-based SLAM for large scale environments. More details about ORB-SLAM2 can be found in the following section. Labbé and Michaud (2018) presented a frame-to-map approach for motion estimation. Their SLAM algorithm is an extension of their appearance-based loop closure approach (RTAB-Map) which includes a powerful memory management tool.
Badino and Kanade (2011) presented a very efficient feature-based stereo visual odometry algorithm which tracks Harris features (Harris and Stephens (1988)) from frame to frame, using a Kanade–Lucas–Tomasi tracker (Tomasi and Kanade (1991)). For dealing with outliers, they use the robust iterative least squares. Another example of efficient stereo visual odometry is the work of Geiger *et al.* (2011b). They proposed a simple descriptor which is designed for easy vectorization using SSE2 instructions. For robustness against outliers, they proposed to track the features on the four images (two stereo pairs) in a loop and they employed a minimal P3P solver in a RANSAC scheme. Buczko and Willert (2016) propose to decouple the rotation from the translation when estimating the motion. The results from their algorithm on the KITTI odometry benchmark are remarkable, given that they propose a visual odometry only algorithm and no SLAM.
Direct methods estimate the motion parameters directly from the images without prior feature extraction and matching steps. The goal is to compute the deformation between

the reference image and the current image such that a measure of dissimilarity is minimized. Lucas and Kanade (1981) presented the first direct image alignment algorithm for optical flow and stereo. The Lucas-Kanade algorithm is a direct application of non-linear least squares. The Lucas-Kanade algorithm and its variants have been applied to various parametric image alignment applications such as tracking of rigid and articulated objects (Black and Jepson (1998)) and mosaic construction (Shum and Szeliski (2000)).
Direct methods for visual odometry and visual SLAM are traditionally known for being computationally intensive. Nevertheless, there are some efficient dense SLAM approaches. Dense tracking and mapping in real-time (DTAM, see Newcombe *et al.* (2011)) can achieve real-time performance on GPU hardware. The sensor used for DTAM was an RGB-D camera. Large-Scale Direct SLAM with Stereo Cameras (LSD-SLAM, see Engel *et al.* (2015), Engel *et al.* (2014)) is a more efficient example which achieves real-time performance using a CPU. LSD-SLAM estimates the inverse depth for the keyframes and uses this inverse depth to estimate the camera pose by aligning the current frame with respect to the reference keyframe. LSD-SLAM uses the forward compositional Simon and Matthews (2004) variant of the Lucas Kanade algorithms (see Lucas and Kanade (1981) and Simon and Matthews (2004)). In our hybrid SLAM, we use the more efficient inverse compositional alignment algorithm of Baker *et al.* (2001), which allows the pre-computation of the Jacobian (and Hessian). Forster *et al.* (2014) introduced sparse direct image alignment. They use the inverse compositional algorithm to track a set of sparse features by means of image alignment. They extend their approach for multiple cameras in the recent work Forster *et al.* (2016). Unlike Forster *et al.* (2014) and Forster *et al.* (2016) which is a visual odometry algorithm, our algorithm is a full SLAM algorithm. Fang and Scherer (2014) carried out an experimental study of many visual odometry algorithms (using RGB-D cameras) where they emphasized on the trade-off among accuracy, robustness, and computation speed. Klose *et al.* (2013) presented a survey of direct image alignment algorithms, which are based on the compositional motion update. Most direct visual odometry algorithms use the intensity differences to measure the dissimilarity between the warped image and input image. Alismail *et al.* (2016a) proposed to use a low complexity dense binary descriptor (bit-planes) which they show to be more robust against illumination changes. They used a challenging dataset recorded using an underground mining robot.

3.4 ORB-SLAM2: Parallel tracking, mapping and loop closing

The feature based part of our hybrid visual stereo SLAM system uses the popular ORB-SLAM2 (Mur-Artal and Tardós (2016)) algorithm. The ORB-SLAM2 algorithm splits the tracking from the local mapping using two separate CPU threads. This architecture was initially proposed by Klein and Murray (2007) in their popular parallel tracking and

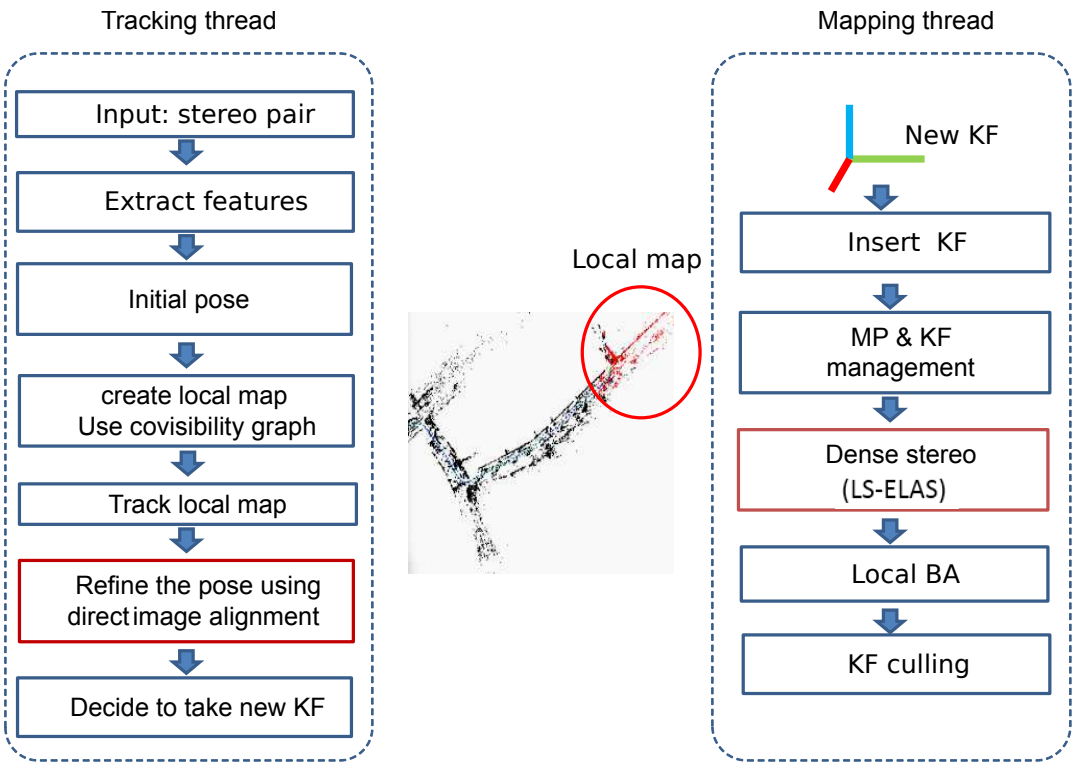


Figure 3.1: A diagram illustrating the tracking and mapping threads of our hybrid stereo SLAM. Our algorithm extends the ORB-SLAM2 algorithm. The blue boxes are from ORB-SLAM2 and the red boxes are our modification to ORB-SLAM2.

mapping (PTAM) algorithm. The PTAM algorithm was then successfully used by many mobile robotics researchers, like Scherer and Zell (2013), Scherer *et al.* (2012c). The ORB-SLAM2 algorithm starts by extracting a (sparse) set of ORB features (Rublee *et al.* (2011)) on both images of the current stereo frame. The features are then matched on the previous stereo frame to establish 2D-2D correspondences. Using the map we get 3D-2D correspondences. A PnP solver is then used to optimize the initial pose. Using the co-visibility graph (a graph which connects keyframes which share enough map points), a local map is extracted and the pose gets refined by using more map points. The local map contains the keyframes and the map points observed by these local keyframes. The final pose we get from ORB-SLAM2 is estimated using only sparse features. Valuable information might be missed by abstracting the images to a set of ORB features. We propose to further refine the pose by using direct image alignment and use more data from the image. We do not use all of the pixels but only those which have valid depth (from dense stereo matching) and have an image gradient larger than a given threshold.

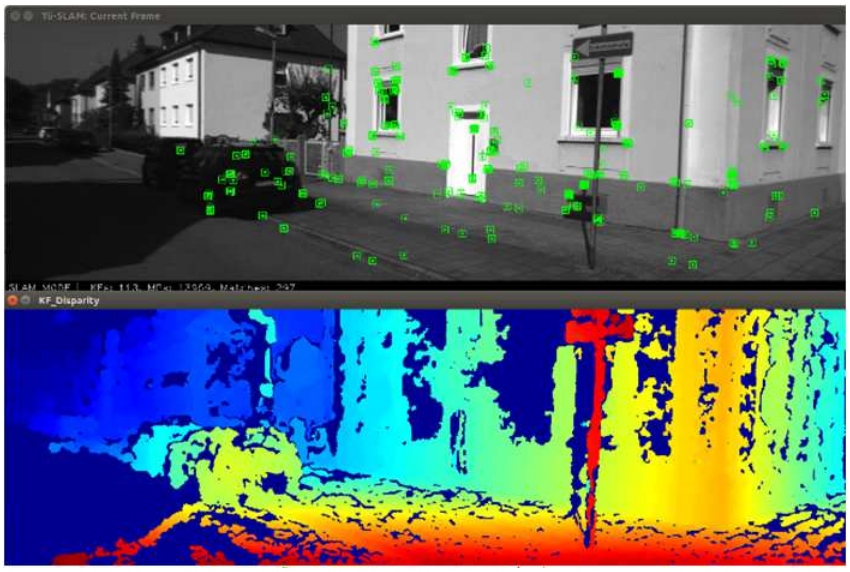


Figure 3.2: The pixels that can contribute to the computation of the motion. Top: sparse ORB-features used on the feature-based method. Bottom: disparity map computed for the reference keyframe used on the direct method. If needed, all pixels with valid disparity can contribute to the estimation of the motion. More accuracy and robustness can be obtained by virtue of measurements redundancy.

3.5 Hybrid stereo SLAM

We propose a hybrid visual stereo SLAM algorithm which combines a feature-based method and a direct image alignment method. As discussed in the introduction, the feature-based approaches are generally fast and they can handle large camera movements. We make use of this characteristic and design a hybrid visual stereo SLAM which uses the motion estimation from a feature-based visual SLAM as an initial guess for a direct image alignment method refinement step. Our hybrid stereo SLAM is based on the popular ORB-SLAM2 algorithm (see Mur-Artal and Tardós (2016)). We modify the tracking thread such that we refine the poses using direct image alignment and we modify the mapping thread such that we compute dense binocular stereo matching at keyframes. Fig. 3.1 shows the different components of the tracking thread and mapping thread of our algorithm. The loop closure thread is not shown.

3.5.1 Pose initialization using ORB features

We chose to initialize the pose estimate using a feature-based method. The feature-based methods deals effectively with large motion. This can be attributed to the development of descriptors which are invariant to a range of geometric (i.e. view-point) and photometric changes. This claim can be supported by many works such as Agarwal *et al.*

(2009), which used SIFT features to perform Structure-from-Motion on 150 thousands of unorganized images from Flickr.

Minimizing Euclidean distances

Our system uses ORB-SLAM2 to estimate ξ_f^*, the pose of the current frame based on features. Any other feature-based stereo SLAM can be used. We compute T_f by minimizing the re-projection error (see Eq. 3.2) between predicted pixel locations and the observed pixels locations (see Eq. 3.1).

$$e_f(x_i) = x_i - \pi(RX_i + t) \tag{3.1}$$

$$T_f^* = \underset{T_f}{\operatorname{argmin}} \sum_{i \in \aleph} \rho \left(\| x_i - \pi(RX_i + t) \|_{\Omega_i} \right) \tag{3.2}$$

Where Ω_i is a weighting (inverse covariance matrix) associated with the scale at which the feature was extracted. And ρ is the Huber robust cost function. We note that we minimize distances (pixel locations) on the image plane. We illustrate this case in Fig 3.3.

Initial pose of the direct alignment

The final pose ξ_f^* we estimate by means of the feature-based motion estimation is used as initial guess for our direct image alignment motion refinement. This is described in the following equation (see Eq. 3.3):

$$\breve{\xi}_d = \xi_f^* \tag{3.3}$$

3.5.2 Refining the pose using direct image alignment

Image warp

The 3D warp $W(x_i, \xi_d)$ which maps a pixel $x_i \in I_{ref}^l$ in the reference keyframe into its corresponding pixel in the image I_{cur}^l using the pinhole camera projection model π is given by:

$$\begin{aligned} W &: \mathbb{R}^2 \times \mathbb{R}^6 \to \mathbb{R}^2 \\ &(x, \xi_d) \to W(x, \xi_d) = \pi(\pi^{-1}(x, Z), g(T(\xi_d))) \end{aligned} \tag{3.4}$$

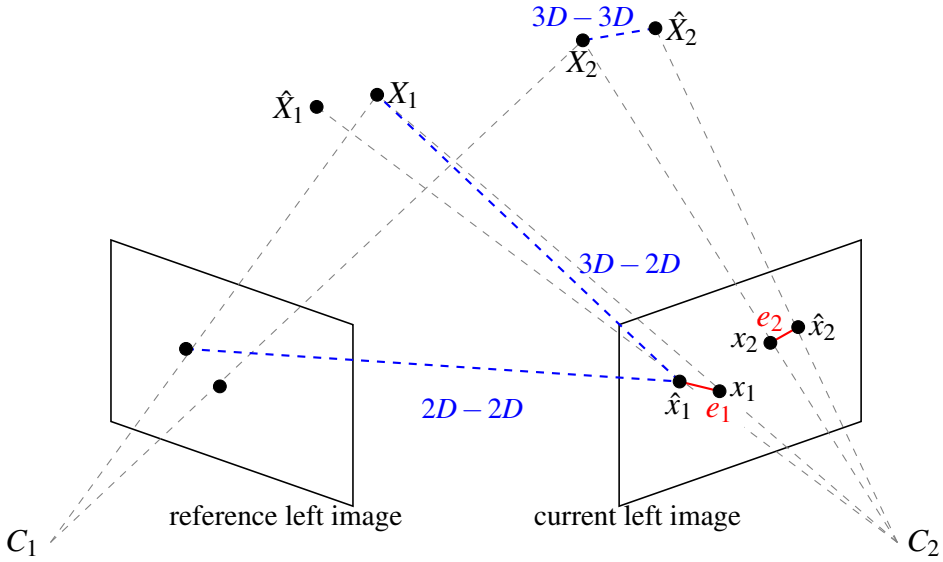


Figure 3.3: Feature-based motion estimation: we aim to find the transform (pose of the current camera) for which the sum of all re-projection distances e_i (shown in red) is minimal. 3D-2D correspondences for motion estimation are in general more accurate than 3D-3D correspondences. x_1 denotes the true projection of the true 3D point X_1. The pixel $\hat{x}_1$ denotes the detected/estimated corresponding pixel. $\hat{X}_1$ denotes the reconstructed 3D point based on the estimated correspondence.

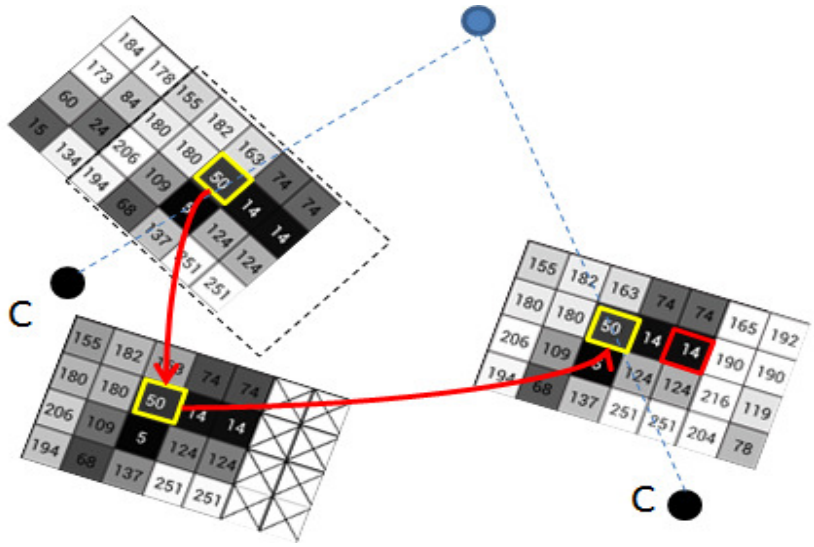


Figure 3.4: Direct image alignment motion estimation: illustration of the inverse compositional algorithm. Here, we minimize the photometric error. The color (gray values) of the squares encodes the intensity value [0-255]. The search for the warp increment (image bottom left) is done in the reference image (top left) and then this warp is inverted and composed (see Eq. 3.10) with the current warp of the target image (right image). The result is a direct warp from the template image to the target image. The goal is to find the warp parameters (motion parameters) which minimize the dissimilarity between the warped template image and the input image.

where

$$\pi\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x\frac{X}{Z} + c_x \\ f_y\frac{Y}{Z} + c_y \\ f_x\frac{X-b}{Z} + c_x \end{bmatrix} \tag{3.5}$$

and its inverse maps a 2D pixel $x = (u, v)$, with known depth Z, to its corresponding 3D point $p = (X, Y, Z)$:

$$\pi^{-1}\left(\begin{bmatrix} u \\ v \\ Z \end{bmatrix}\right) = \begin{bmatrix} \frac{u-c_x}{f_x}Z \\ \frac{v-c_y}{f_y}Z \\ Z \end{bmatrix} \tag{3.6}$$

Minimizing the intensity differences

We compute T_d by minimizing the re-projection photometric error between the predicted intensities of the pixels and the observed intensities. We illustrate this case in Fig 3.4. We work with 8-bit gray-scale images with intensities in the interval [0-255]. The error is defined as follows:

$$e_d(x_i) = I^l_{ref}(W(x_i, \delta\xi_d)) - I^l_{cur}(W(x_i, \xi_d)) \tag{3.7}$$

We optimize using pixels $x_i \in D_{ref}$ where D_{ref} is the set of pixels $x_i \in I^l_{ref}$ such that x_i has a valid depth and the magnitude of the intensity gradient at x_i is larger than a given threshold. In practice, we use about 15% of the image pixels. Note that the domain D_{ref} here is defined in I^l_{ref} for the inverse compositional image alignment. For the forward additive algorithm and the compositional forward algorithm, the domain is defined on the current (input) image (I^l_{cur}).
We iteratively optimize the following energy function to align I^l_{cur} with I^l_{ref}:

$$\xi^*_d = \underset{\xi_d}{\operatorname{argmin}} \sum_{x \in D_{ref}} \|I^l_{ref}(W(x_i, \delta\xi_d)) - I^l_{cur}(W(x_i, \xi_d))\|^2 \tag{3.8}$$

$$= \underset{\xi_d}{\operatorname{argmin}} E_d(\delta\xi_d) \tag{3.9}$$

Compositional motion update

Since the increment $\delta\xi_d$ is computed for the reference keyframe image (the template image not the target image) we need to invert it and and compose it with the current parameter estimate. The warp update at iteration $k+1$ is given by Eq. 3.10.

$$T^{k+1}_d = T^k_d * exp(-\delta\xi^*_d) \tag{3.10}$$

Linearization using Taylor expansion

In the iterative optimization process, we linearize around $\delta\xi = 0$ at every iteration using Eq. 2.4 applied to the cost function in Eq. 3.8.

$$E_d \approx \sum_{x \in D_{ref}} \| I^l_{ref}(W(x,0)) - I^l_{cur}(W(x,\xi_d)) + J_d \delta\xi_d \|^2 \tag{3.11}$$

For the inverse compositional direct alignment step of our algorithm, the Hessian and its inverse can be pre-computed (for all iterations). Thus, solving the linear system in Eq. 2.6 becomes trivial. The parameter update is also different since it involves inverted composition rather than forward additive increments (See Eq. 3.10). The pre-computation of the Jacobian and Hessian assumes that we are not using M-estimator robust functions. Using a robust function involves the use of iterative re-weighted least squares and the Jacobian and the Hessian needs to be recomputed at each iteration. We provide the option to use the Huber and Tukey robust functions.

Computing the Jacobians

The derivation of the Jacobians for our inverse compositional algorithm is similar to the forward compositional algorithm in Kerl (2012). However, the Jacobians in our case are computed for the I^l_{ref} image and not for I_{cur} as in Kerl (2012). For the sake of completeness and clarity, we remind the derivation of the Jacobians for the inverse compositional algorithm. We use the same notation as in Kerl (2012).

$$J_d(x,\xi_d) = \frac{\partial E_d}{\partial \xi_d} \tag{3.12}$$

The Jacobian can be computed using the chain rule.

$$J_d(x,\xi_d) = J_I J_\pi J_g J_T \tag{3.13}$$

$$\begin{aligned} = & \frac{\partial I^l_{ref}(W(x,\delta\xi_d))}{\partial \pi}\Big|_{x=\pi(g(p_i,T(0))=x_i} \cdot \\ & \frac{\partial \pi(p)}{\partial g}\Big|_{p=g(p_i,T(0))=p_i} \cdot \\ & \frac{\partial g(p,T)}{\partial T}\Big|_{\substack{T=T(0)=I^l_{cur}, \\ p=p_i}} \cdot \\ & \frac{\partial T(\xi_d)}{\partial \xi_d}\Big|_{\xi_d=0} \end{aligned} \tag{3.14}$$

The individual Jacobians in Eq. 3.12 can be derived as follow.
The intensity Jacobian J_I is evaluated at $x = \pi(g(p_i, T(0))$, where $T(0) = I_{4x4}$.

$$J_I = \frac{\partial I^l_{ref}(W(x, \delta\xi_d))}{\partial \pi}\Big|_{x=\pi(g(p_i,T(0))=x_i} \tag{3.15}$$

$$= (\nabla I^l_{ref,u} \nabla I^l_{ref,v}) \tag{3.16}$$

The camera projection Jacobian is:

$$J_\pi = \frac{\partial \pi(p)}{\partial g}\Big|_{p=g(p_i,T(0))=p_i} \tag{3.17}$$

$$= \begin{bmatrix} f_x \frac{1}{z} & 0 & -f_x \frac{x}{z^2} \\ 0 & f_y \frac{1}{z} & -f_y \frac{y}{z^2} \end{bmatrix} \tag{3.18}$$

The Jacobian of the function g is:

$$J_g = \frac{\partial g(p,T)}{\partial T}\Big|_{\substack{T=T(0)=I^l_{cur}, \\ p=p_i}}$$

$$= \begin{bmatrix} x.I_{3\times3} & y.I_{3\times3} & z.I_{3\times3} & I_{3\times3} \end{bmatrix} \tag{3.19}$$

The Jacobian of T is:

$$J_T = \frac{\partial T(\xi_d)}{\partial \xi_d}\Big|_{\xi_d=0} \tag{3.20}$$

$$= \begin{bmatrix} & 0 & 0 & 0 \\ 0_{3\times3} & 0 & 0 & 1 \\ & 0 & -1 & 0 \\ & 0 & 0 & -1 \\ 0_{3\times3} & 0 & 0 & 0 \\ & 1 & 0 & 0 \\ & 0 & 1 & 0 \\ 0_{3\times3} & -1 & 0 & 0 \\ & 0 & 0 & 0 \\ \\ I_{3\times3} & & 0_{3\times3} & \end{bmatrix} \tag{3.21}$$

Finally, we get the expression for our per-pixel Jacobian:

$$J_d = J_I J_\pi J_g J_T \tag{3.22}$$

$$= (\nabla I^l_{ref,u} f_x, \nabla I^l_{ref,v} f_y) . \begin{bmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} & -\frac{xy}{z^2} & (1+\frac{x^2}{z^2}) & -\frac{y}{z} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} & -(1+\frac{y^2}{z^2}) & \frac{xy}{z^2} & \frac{x}{z} \end{bmatrix} \tag{3.23}$$

Note that the per-pixel Jacobian J_d is a 6-dimensional vector. By checking the dimensions of the different matrices which compose J_d, we get:

$$\underbrace{J_d}_{1\times 6} = \underbrace{J_I}_{1\times 2} . \underbrace{J_\pi}_{2\times 3} . \underbrace{J_g}_{3\times 12} . \underbrace{J_T}_{12\times 6} \tag{3.24}$$

3.5.3 Loop closure thread

We rely on the feature-based bundle adjustment from ORB-SLAM2 for the back-end of our SLAM algorithm. Photometric bundle adjustment (see Alismail *et al.* (2016b)), which jointly optimizes the camera trajectory and scene geometry, is unfortunately still not suitable for real-time application on our hardware. Thus, a SLAM which is based only on image alignment (and no features) is not developed in the scope of this dissertation. Engel *et al.* (2015) developped a SLAM algorithm, which is based only on image alignment. However, their SLAM back-end optimizes only the motion and does not optimize the structure. They perform pose graph optimization instead of jointly optimzing the structure and the motion. We use the SLAM back-end from ORB-SLAM2. The loops are detected using bag of binary words. The loop correction is done using the efficient on-manifold general optimization framework g2o (see Kuemmerle *et al.* (2011)). Whenever a new keyframe is created, the loop closure thread searches for potential loops. If a loop is detected and validated then a process including two steps for loop correction is performed. The first step is the optimization of a pose graph called the essential graph. Then, bundle adjustment for jointly optimizing the structure and the motion is performed to compute the optimized camera trajectory and scene geometry.

3.5.4 Autonomous quadcopter flights

We implemented our SLAM algorithm on our experimental quadcopter for performing outdoor autonomous flights. The experimental setup is described in page 107. This is a part of the large system for outdoor collision free autonomous navigation which is described in chapter 5. We conducted various experiments in an outdoor environment which has mostly vegetation, asphalt and buildings. In this environment, the algorithm detects enough ORB features for tracking the robot pose. We had to replace the camera lenses with lenses which have IR-cut filter to enhance the image quality on vegetation.

We commanded the quadcopter to follow a path consisting of predefined way-points. Since we were not using any external positioning system (e.g. GPS), we could not record reference measurements (ground-truth data). Therefore, we evaluate our algorithm on a public outdoor stereo dataset (see next section). The experiments show that the algorithm could effectively be used on-board in real-time for outdoor autonomous flights. Fig. 3.5 shows the estimated trajectory from an outdoor experiment using our quadcopter research robot.

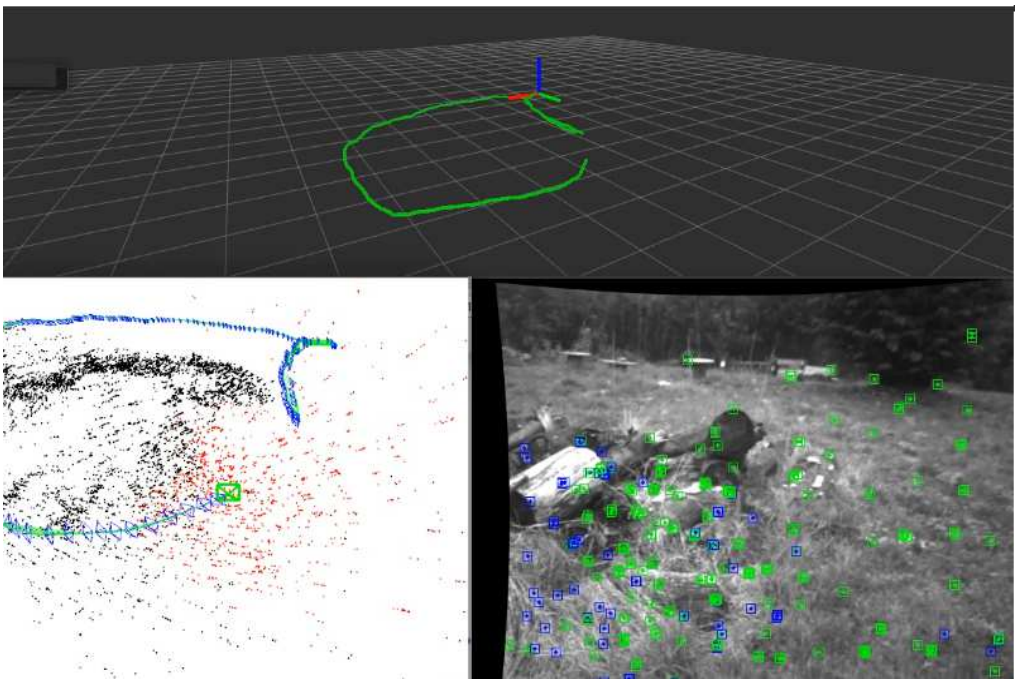


Figure 3.5: Outdoor experiment. First row: robot trajectory. Second row left: robot trajectory, keyframes and the map points (the landmarks). Second row right: an image of the scene showing the tracked ORB features.

3.6 Evaluation using the KITTI odometry Dataset

3.6.1 Evaluation criteria

We have targeted an objective evaluation of our hybrid stereo SLAM algorithm. We identified many requirements, which we consider they contribute to achieve an objective evaluation. In the following, we summarize the most relevant requirements:

1. The algorithm needs to be compared to other algorithms based on the same input data and the same metrics.
2. Quantitative results of the performance of the algorithm shall be reported by the algorithm developers. The other algorithms should be tuned by their developers.
3. The performance measures should be robust. In particular, the measure should not be based on the absolute reference.
4. The algorithm needs to run using the same parameters and should not not be tuned for each sequence individually.
5. The dataset for the evaluation should be challenging. The environment should be large, diverse, contain loops and illumination changes.
6. As we are interested in using our quadcopter in outdoor environment (see chapter 5 page 105), we are mainly interested in outdoor datasets.
7. It is preferred to evaluate the algorithms on some test data (for which there is no available public ground truth data).

We consider that the evaluation by a third party on a publically available odometry benchmark is an effective way to achieve an objective evaluation. We made a benchmark of available public stereo odometry datasets. Table 3.1 lists a comparison of three relevant public stereo odometry datasets. We chose to evaluate our algorithm on the KITTI

Table 3.1: Comparaison of relevant publicaly available visual SLAM datasets

Dataset	Cameras	Statistics	Ground thruth	Env.	ranking
KITTI	1392x512 @10 Hz	39.2 km 41 k frames 22 sequences	Accuracy 10cm OXTS RT 3003 10 Hz	Outdoor	Yes
EuRoC	752x480 pixels @20 Hz	0.9 km 11 sequences	Accuracy 1mm Motion capture (Vicon) 200Hz	Indoor	No
New College	512x384 pixels @20 Hz	2.2 km 51 k frames 9 sequences	GPS CSI Seres 5 Hz	Outdoor	No

odometry benchmark (see Geiger *et al.* (2012)) for two reasons. First, it includes a robust performance measure and maintains a ranking list of other algorithms. This is very important, as it allows comparing our algorithm with other algorithms for which no open-source code is available. Second, the dataset is recorded in an outdoor environment where RGB-D cameras won't work.

3.6.2 The KITTI odometry dataset

The platform used for recording the data is a car driving in the city of Karlsruhe in Germany. The platform was equipped with many sensors including a forward-looking gray-scale stereo rig mounted on the top of the platform. The cameras have a frame rate of 10 Hz and a resolution of 1392×512 pixels. The recorded trajectories have a total length of about 39.2 *km* divided into 22 challenging sequences. The distance accuracy of the ground truth is 2 *cm*. Some sequences include multiple loops and dynamic objects (moving vehicles and moving pedestrians). The existence of loops on the trajectories is of particular importance to evaluate full SLAM algorithms. Ground truth trajectories are provided for the first 11 training sequences. The ground truth trajectories have an accuracy of 10 *cm*. A GPS system of type OXTS RT 3003 operating at 10 Hz was used to obtain the ground truth trajectories. The sequences 11 to 22 are used as test data sets.

3.6.3 Error measures

In this section, the following notations are used:

- $C_i \in SE(3)$: camera/robot pose at time i estimated by the SLAM algorithm.
- $C_i^{gt} \in SE(3)$: the ground truth pose at time i.
- i : index which represents the time (i.e frame number).
- N : number of camera poses.
- $\oplus$: standard motion composition operator.
- $\ominus$: inverse motion composition operator.
- $\delta_{i,j} \in SE(3)$: relative transformation between poses C_i and C_j.

$\delta_{i,j}$ moves the camera from C_i to $C_j = C_i \oplus \delta_{i,j}$. An intuitive error metric to compare the performance of SLAM algorithms can be achieved by summing up all the individual errors between the estimated poses and their corresponding ground truth poses. This can be formulated using Eq. 3.25.

$$E(C) = \sum_i (C_i \ominus C_i^{gt})^2 \tag{3.25}$$

This simple metric is based on the comparison of the estimated poses to the absolute poses of the ground truth. It has been shown that this method is sub-optimal (see

Kümmerle *et al.* (2009)). It relies on absolute references which make it dependent on when an error has been occurred. An error which occurs at the beginning of the trajectory will be counted multiple times. Fig. 3.6 illustrates the difference between the absolute and relative error metrics. In this figure the camera moves in 1D with constant motion increment δ. We compare the results of two algorithms 1 and 2. Both algorithms provide perfect estimate for all but one transition. The algorithm 1 made the error at the end of the trajectory (last transition). The algorithm 2 made an error of the same amount as algorithm 1 but this time at the beginning of the trajectory. This introduced a shift of all consecutive poses. Based on the metric given in Eq. 3.25, the algorithm 1 is evaluated as more accurate than the algorithm 2. The total error for algorithm 1 is ε. For algorithm 2, the total error is 5ε: the error ε occurring at the first transition is counted multiple times (on all consecutive transitions). A fair metric would provide the same score for both algorithms.
Kümmerle *et al.* (2009) addressed this problem and proposed a metric which depends on the relative transformations between the poses instead of the absolute transformations.

$$E(\delta) = \frac{1}{N}\sum_{i,j}(trans(\delta_{i,j} \ominus \delta_{i,j}^{gt})^2 + rot(\delta_{i,j} \ominus \delta_{i,j}^{gt})^2) \tag{3.26}$$

Inspired by the work of Kümmerle *et al.* (2009), Geiger *et al.* (2012) used (in the KITTI benchmark) a similar metric for evaluating the performance of SLAM algorithms. However, Geiger *et al.* (2012) split the rotational error from the translational error. The metric for the quantifying translational error is given by Eq. 3.27:

$$E_{trans}(\delta) = \frac{1}{N}\sum_{i,j}||trans(\delta_{i,j} \ominus \delta_{i,j}^{gt})||_2 \tag{3.27}$$

Similarly, the metric for the quantifying the rotational error is given by Eq. 3.28:

$$E_{rot}(\delta) = \frac{1}{N}\sum_{i,j}\sphericalangle(\delta_{i,j} \ominus \delta_{i,j}^{gt}) \tag{3.28}$$

Where $\sphericalangle$ is the rotation angle.
The KITTI odometry benchmark computes the translational and the rotational errors for all possible subsequences of length $\{100,200,...,800\}$ meters. The compared SLAM algorithms are ranked according to the average of the translational and the rotational errors. The translational error is measured in percent and the rotational error is measured in degrees per meter.
The figures showing the trajectories are best viewed in the electronic format of this dissertation. In the electronic format of this dissertation, the trajectory figures are created using vector graphics to make sure that the reader sees sharp plots when zooming in the figures.

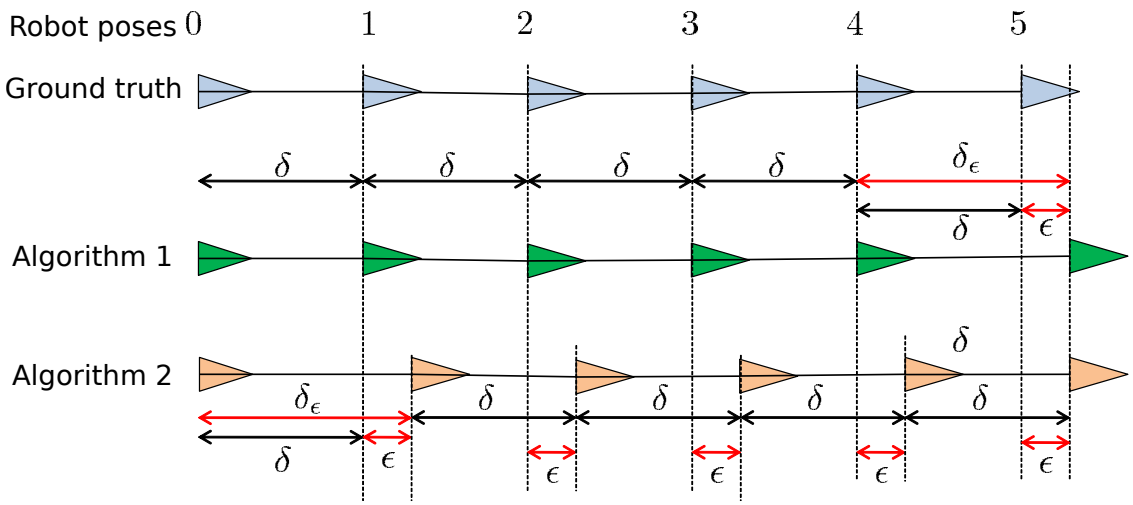


Figure 3.6: An illustration of a case where the metric given by Eq. 3.25 will fail. The two SLAM algorithms did both an error of the same amount but in different places on the trajectory. According to Eq. 3.25, algorithm 1 induces a smaller trajectory error (ε) than the trajectory error of algorithm 2 (5ε). The robust metrics error given by 3.26 and 3.27 yield identical scores for the two algorithms

3.6.4 Trajectories with no loops

In trajectories with no loops, our algorithm achieves a considerable performance improvement compared to ORB-SLAM2. Our method accumulates less drift. The results for sequence 12 of the KITTI odometry benchmark are shown in Fig. 3.7 and in Fig. 3.8. Our algorithm achieves the third best rotational error after Rot-ROCC and RTAB. There is a difference of the rotational error of $\sim 0.0005\,deg/m$ between our algorithm and ORB-SLAM2. The average rotational error on sub-sequences of $100\,m$ length is $0.2\,deg$ (i.e. $0.002 * 100$) for our algorithm and $0.25\,deg$ (i.e. $0.0025 * 100$) for ORB-SLAM2. On sub-sequences of $800m$ length, the average rotational error increases to $1.12\,deg$ (i.e. $0.0014 * 800$) for our algorithm and $1.47\,deg$ (i.e. $0.0018 * 800$) for ORB-SLAM2. The average translational error of ORB-SLAM2 is slightly (up to $0.05\,\%$) smaller than the one of our algorithm. The average translational error on sub-sequences of $600\,m$ length is $0.77\,\%$ for our algorithm and $0.72\,\%$ for ORB-SLAM2.

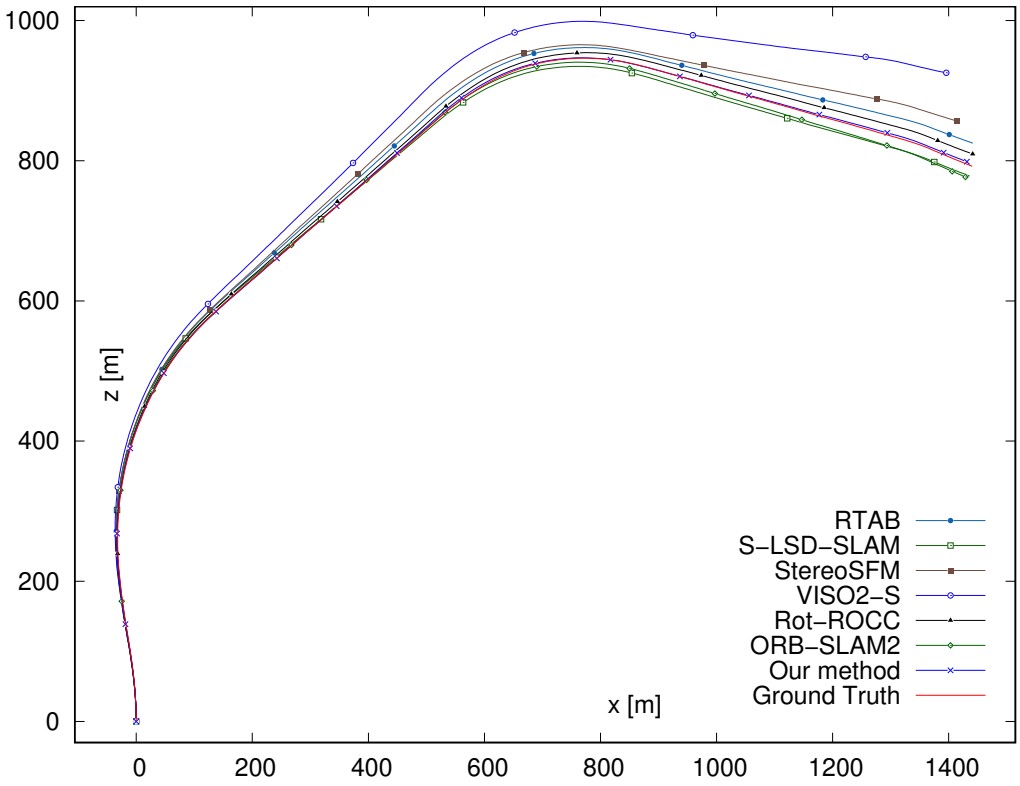


Figure 3.7: Results from the sequence 12 of the KITTI odometry benchmark. This trajectory has no loops. The accuracy of our algorithm is significantly better than the accuracy of ORB-SLAM2.

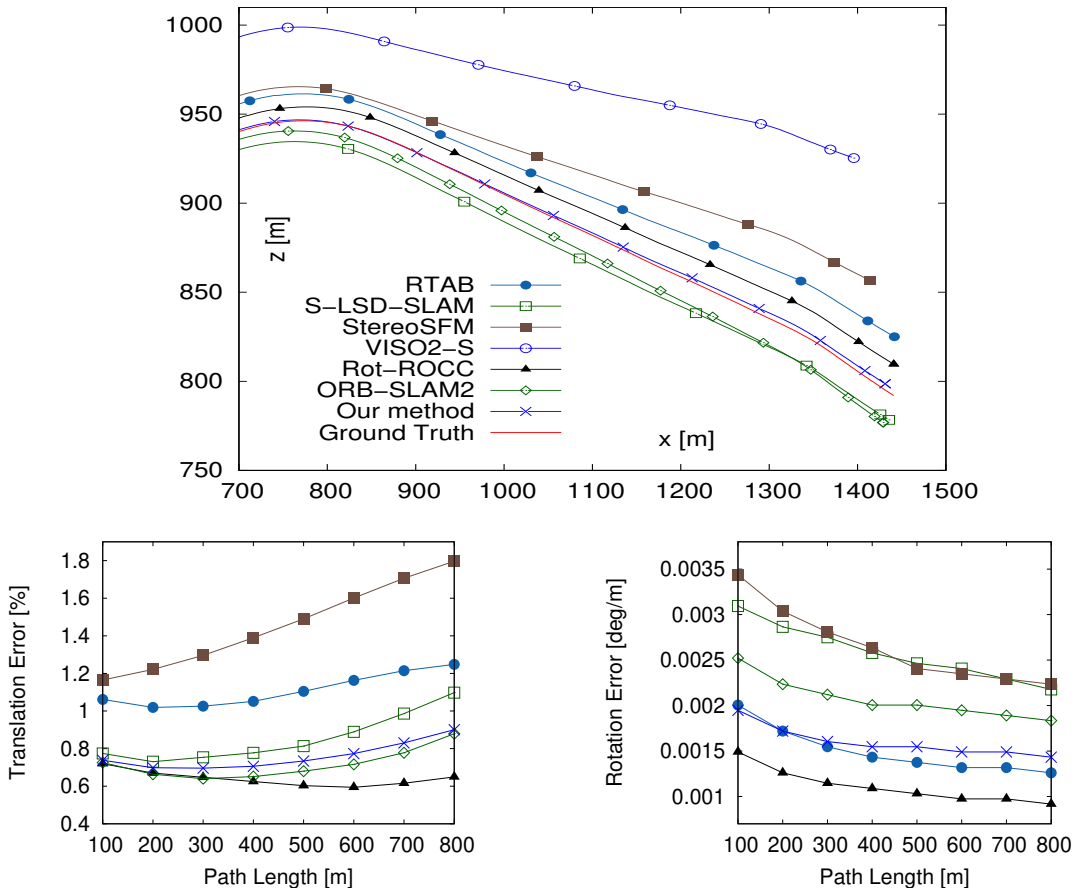


Figure 3.8: Results from the sequence 12 of the KITTI odometry benchmark. Top: a closer look at the last part of the full trajectory on figure 3.7. Bottom left: the relative translational error. Bottom right: the relative rotational error.

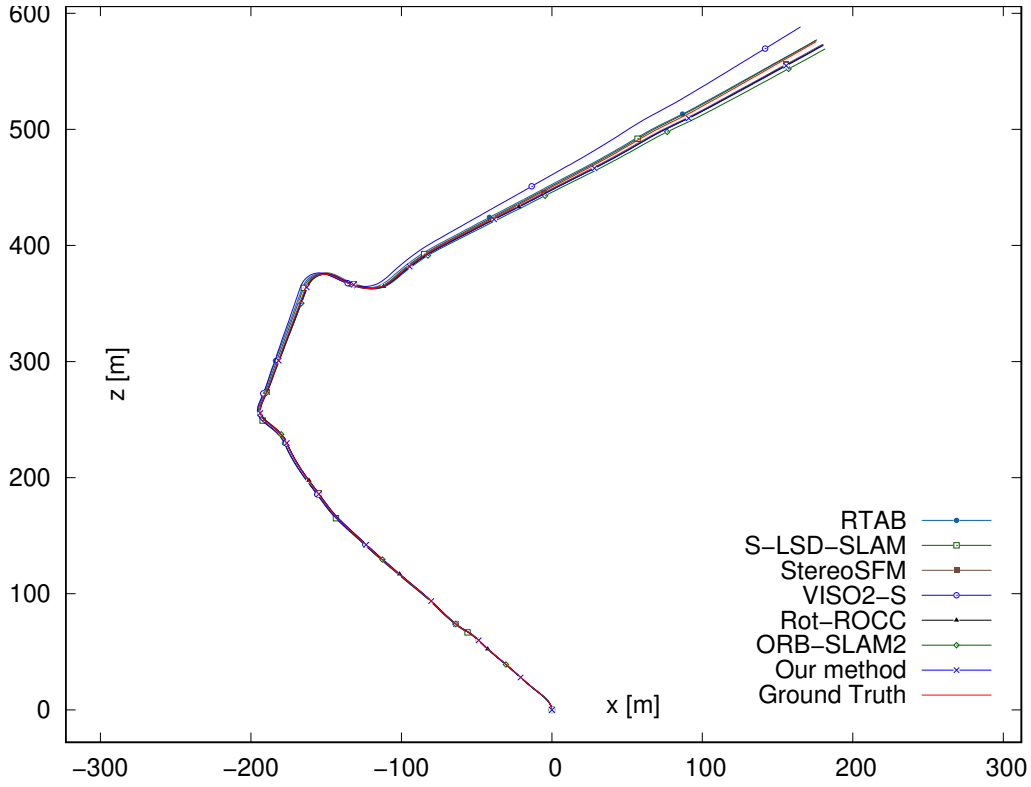


Figure 3.9: Results of the test sequence 11 from the KITTI odometry benchmark. The trajectory has no loops. Similar to the results from the sequence 12 of the the KITTI odometry benchmark, the accuracy of our algorithm is significantly better than the accuracy of ORB-SLAM2.

In the sequence 11 of the KITTI odometry benchmark, our algorithm performs better than ORB-SLAM2. RTAB achieves the best performance. The difference between the performance of RTAB, StereoSFM, Rot-ROCC and our algorithm is marginal. We also want to emphasis here that StereoSFM and our algorithm are faster than RTAB and Rot-ROCC. Running time results will be shown on the last part of the evaluation section (page 64). The results for sequence 11 are shown on Fig. 3.9 and Fig. 3.10. Our algorithm achieves the third best average rotational error and the best average translational error on the sub-sequences of $100\,m$, $200\,m$ and $300\,m$ lengths. While the average translational error from our algorithm is almost constant ($\sim 0.5\%$), the results from ORB-SLAM2 show an increase of the average translational error when the path length increases. ORB-SLAM2 has an average translational error of $0.60\,\%$ for path length equal to $300\,m$ and $1.13\,\%$ for path length equal to $700\,m$. Unlike the results obtained from the sequence 12 of the KITTI odometry benchmark, we see on the results from the sequence 11 that our algorithm has both a smaller translational and a smaller rotational error for all the path lengths compared to ORB-SLAM2. VISO-2, which is a frame-to-frame visual odo-

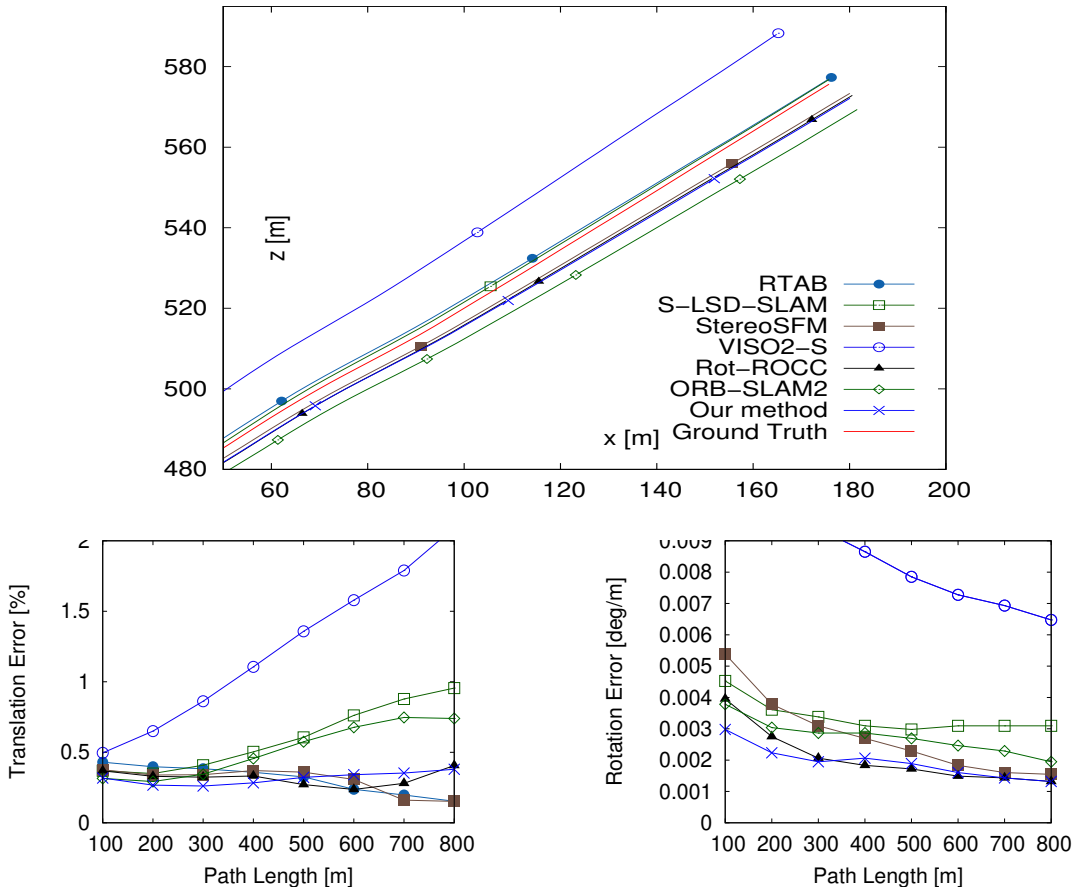


Figure 3.10: Results from the sequence 11 of the KITTI odometry benchmark. Top: a closer look at the last part of the full trajectory on figure 3.9. Bottom left: the relative translational error. Bottom right: the relative rotational error.

metry with no bundle adjustment optimization, has the largest translational and rotational errors. This applies for all path lengths.

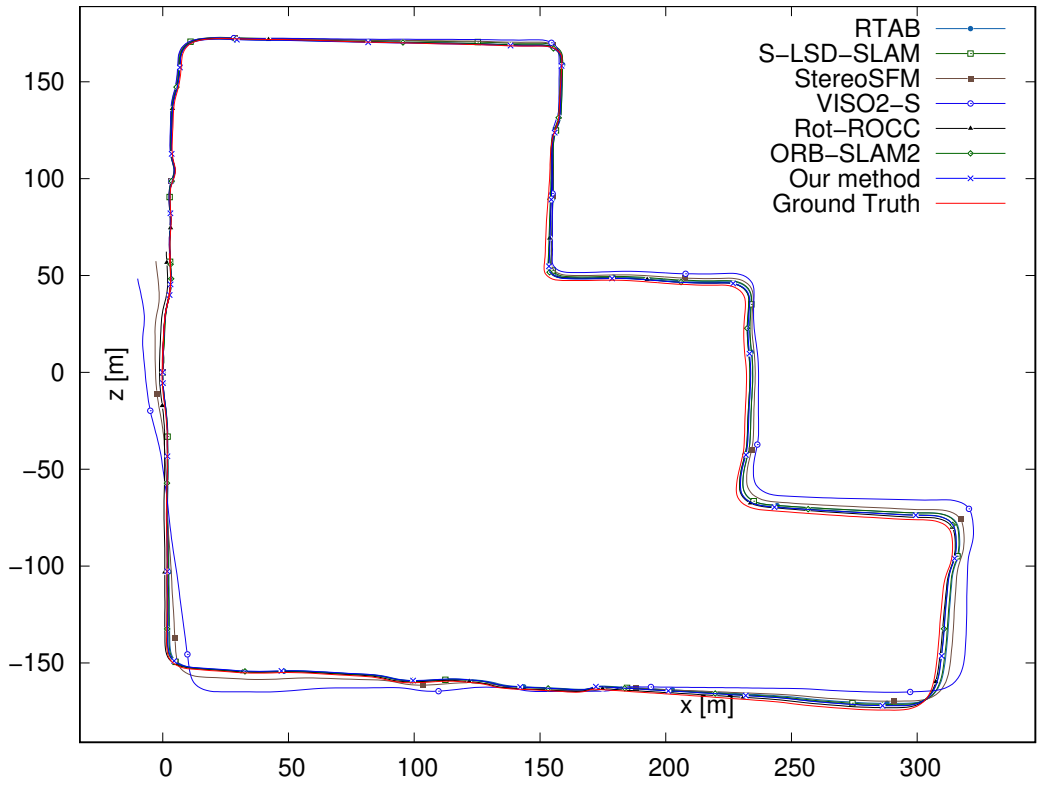


Figure 3.11: Results of the test sequence 15 from the KITTI odometry benchmark. The trajectory has one loop.

3.6.5 Behavior on loopy trajectories

Trajectories with a single loop

Closing loops helps to correct the trajectory. One should be careful not to find (and close) false loops. This would introduce significant errors on the trajectory and it would corrupt the map. On our experiments we did not experience any false loop.
If the robot trajectory includes loops, the benefit of a full SLAM system (with SLAM back-end) becomes obvious. As expected, the SLAM algorithms have a better performance than visual odometry (without SLAM back-end) algorithms such as VISO-2 (see Fig. 3.11(a)). One can see that loop closures help ORB-SLAM2 to achieve a performance comparable to our algorithm. This finding can be consolidated when examining trajectories with multiple loops (see Fig. 3.14). ORB-SLAM2 has a powerfull SLAM back-end. This was also a remark from Jakob *et al.* (2016), who presented a SLAM front-end based on direct sparse visual odometry. Their SLAM front-end outperforms the front-end of ORB-SLAM2 in accuracy. But when activating the SLAM back-end, ORB-SLAM2 outperforms their approach.

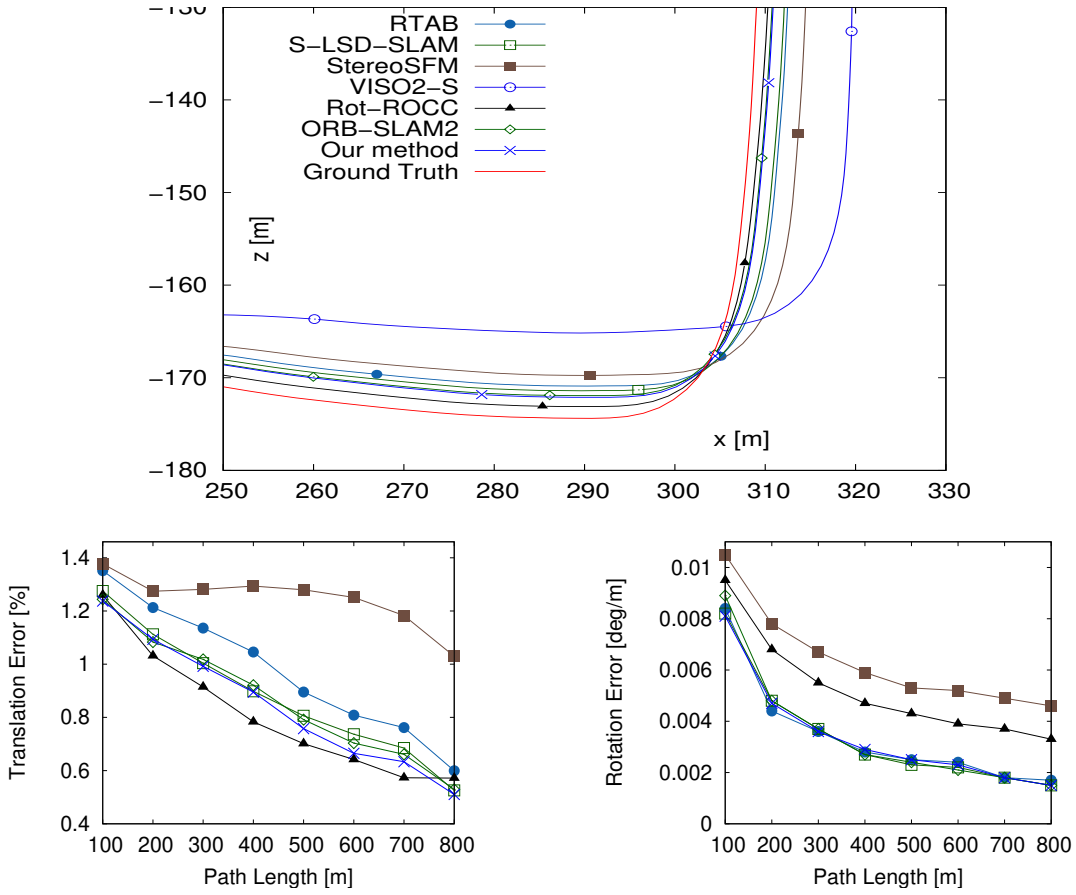


Figure 3.12: Results of the test sequence 15 from the KITTI Odometry benchmark. Top: a closer look at the trajectory on figure 3.13. Bottom left: the relative translational error. Bottom right: the relative rotational error.

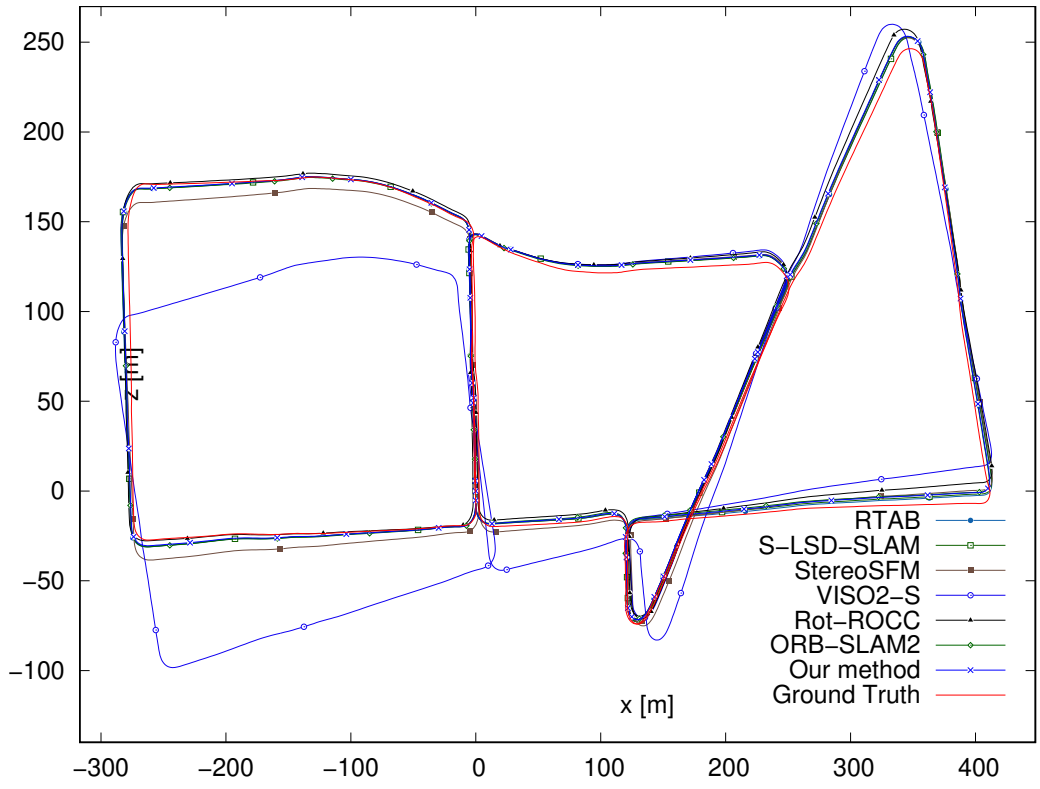


Figure 3.13: Results from the sequence 13 of the KITTI odometry benchmark. This sequence has many loops. Our algorithm and ORB-SLAM2 have almost the same accuracy. As expected, the full SLAM algorithms perform better than visual odometry (without SLAM back-end) algorithms.

Trajectories with multiple loops

It is worth to notice that when there are multiple loops on the trajectory, ORB-SLAM2 benefits from the loop closures more than our hybrid SLAM does. This can be attributed to the fact that we are using the feature-based SLAM back-end from ORB-SLAM2, which is optimized to fit with ORB-SLAM2. As discussed on Sec. 3.5.3 (page 50), performing a back-end which is based on direct image alignment is computational expensive and not suitable for real-time systems. The difference on the performance between our hybrid SLAM and ORB-SLAM2 becomes marginal in these cases. This behavior has been seen on trajectories with one single loop and it becomes even more obvious when there are multiple loops on the trajectory. When there are dynamic objects on the scenes (moving vehicles), ORB-SLAM2 performs better than our algorithm.

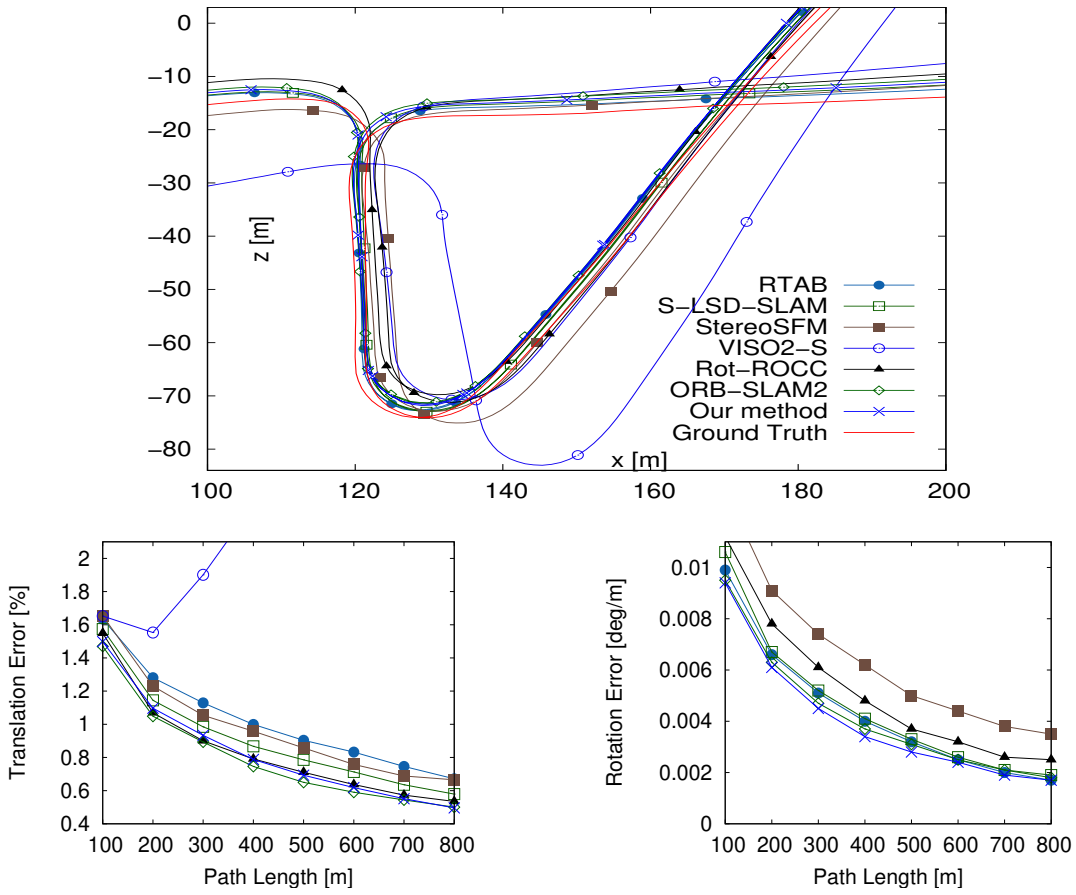


Figure 3.14: Results from the KITTI odometry benchmark sequence 13. The sequence has many loops. There is no significant difference between the performance of our algorithm compared to ORB-SLAM2.

RTAB	1 core @ 2.5 Ghz	0.10 s
S-LSD-SLAM	1 core @ 3.5 Ghz	0.07 s
StereoSFM	**2 cores @ 2.5 Ghz**	**0.02 s**
VISIO2-S	1 core @ 2.5 Ghz	0.05 s
Rot-ROCC	2 cores @ 2.0 Ghz	0.25 s
ORB-SLAM2	2 cores @ 3.5 Ghz	0.06 s
Our method	2 cores @ 3.5 Ghz	0.08 s

Table 3.2: Comparison of the average running time for processing a stereo frame. The times are reported as submitted by the algorithms developers to the KITTI benchmark. All algorithms are implemented using C/C++. The image resolution is (1392×512).

3.6.6 Running time evaluation

Table 3.2 shows the running times from the KITTI benchmark of the different algorithms which we selected for the comparison in the previous section. We report the times as submitted by the algorithms developers. Since different CPUs are used, it is not straighforward to rank the algorithms based on their running time. But given the CPU frequency and the number of cores used for computation, one can get some insights about the running time of the selected algorithms. We remind that the running time depends on other factors such as the cache size. The refinement step of our algorithm requires about 25 % of the (tracking) processing time. While Rot-ROCC (see Buczko and Willert (2016)) is more accurate than ORB-SLAM2 and our method, this comes at the cost of being up to three times computationally more expensive than our method.

3.7 Qualitative evaluation

The figures below (Fig. 3.15 and Fig. 3.16) show some qualitative results of our algorithm. The images are from the sequence 00 of the KITTI odometry benchmark. The point clouds at every fifth keyframe are drawn using a random color. The point clouds seem to be well aligned. The camera poses are represented with a rectangle of size $1.0 \times 0.8\,m^2$.

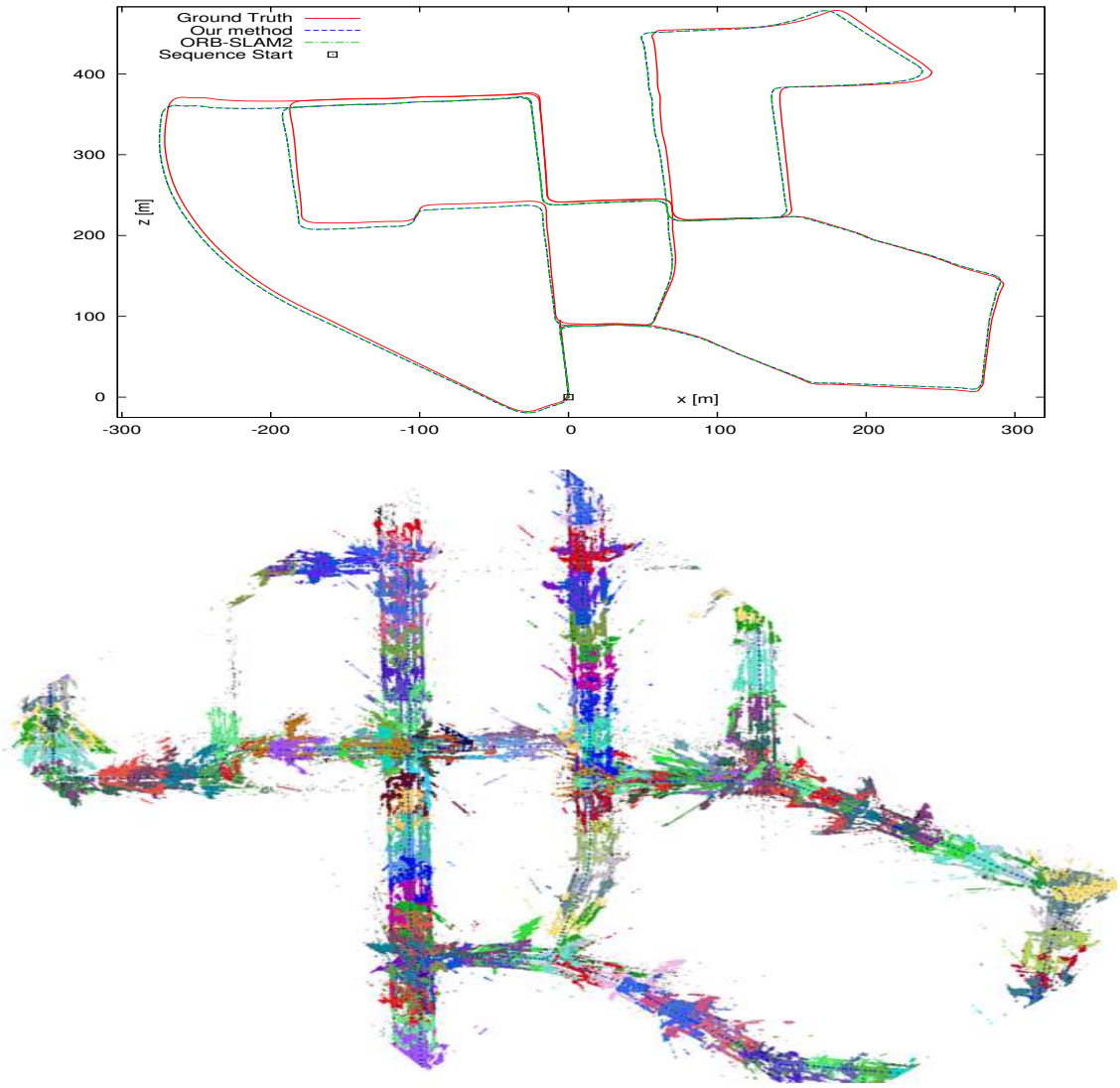


Figure 3.15: Our algorithm can map large environments. Point clouds at every fifth keyframe are shown using a random color. These point clouds will be used to create a consistent global volumetric occupancy map in chapter 5.

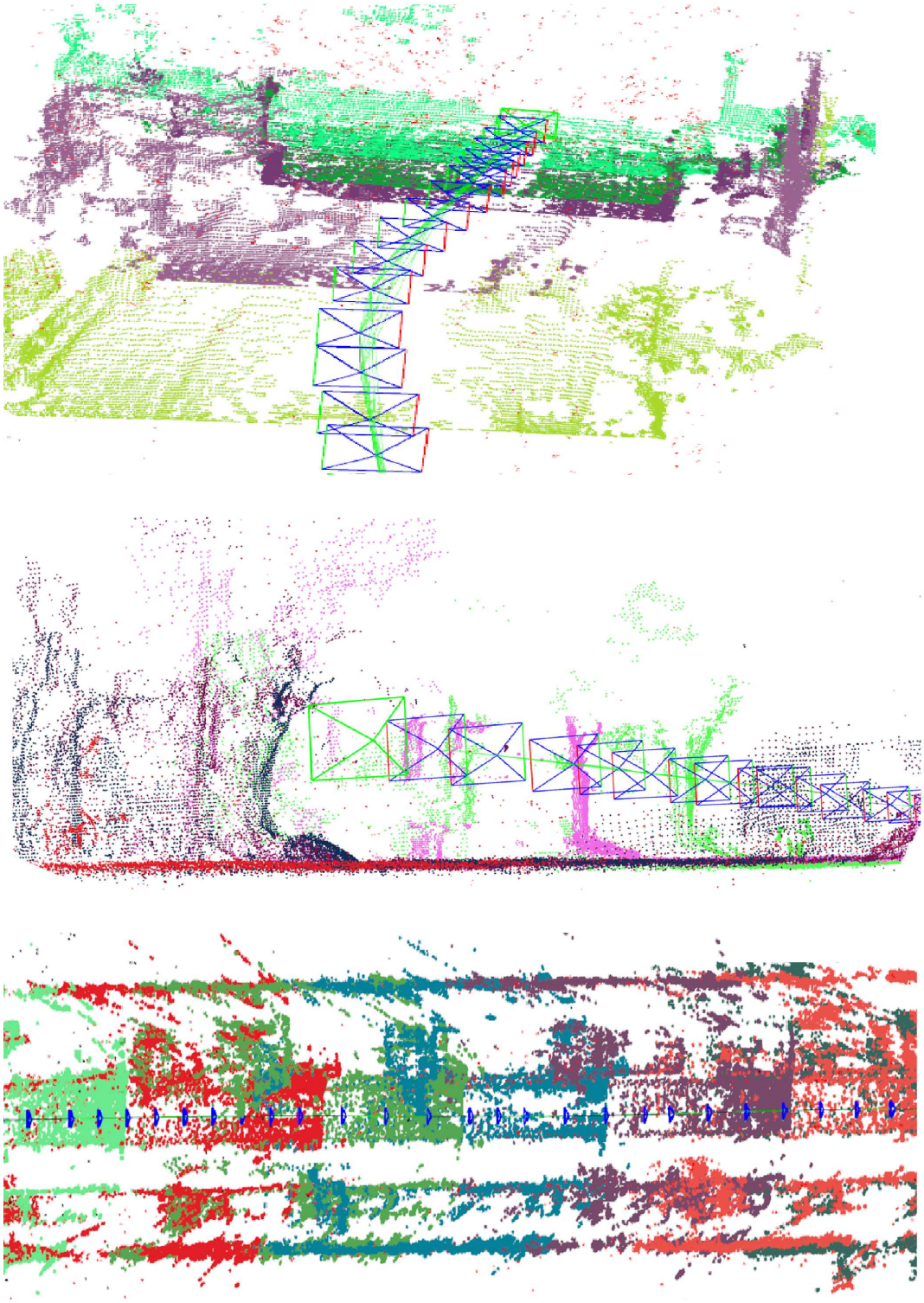


Figure 3.16: Qualitative results. Each point cloud is drawn using a random color. The point clouds are well aligned. The image in the middle shows the road plane. The bottom image shows the bird-view.

3.8 Conclusion

In this chapter, we developed a hybrid stereo SLAM algorithm by extending the popular ORB-SLAM2 algorithm. The ORB-SLAM2 is feature-based and abstracts the input images to a set of ORB features. We proposed to start the tracking thread by using ORB features as in ORB-SLAM2 and to refine the pose using direct image alignment. The algorithm aligns the current left image with the left image of the reference keyframe and updates the pose estimate accordingly. In this refinement step, we start with the feature-based method because it can effectively and efficiently handle large camera displacements. It is efficient because it uses considerably less data (the set of ORB features). It is effective because it uses feature descriptors which are robust against viewpoint changes. The drawback of the feature-based approach is the possible loss of useful details by the abstraction. This drawback can be mitigated by our refinement step. We mean with useful details here, the details that can allow for a more accurate pose estimation. For efficiency, the image alignment is done backward using the iterative inverse compositional algorithm. The Jacobians need to be computed solely in the first iteration. Thus, they can be reused in the consecutive iterations. For efficiency and robustness reasons, we used only the pixels with significant texture changes. To perform the direct alignment we need the depth map at the reference keyframe. This depth map is estimated from stereo matching (see chapter 4). We note that we have developed LS-ELAS before the hybrid SLAM algorithm.

We have evaluated our algorithm using the KITTI odometry benchmark. We selected 6 relevant visual odometry/SLAM algorithms to compare to our algorithm. In trajectories with no loops and no dynamic objects, our algorithm has shown significant improvement of the accuracy of the trajectory estimate. On a sequence of stereo images which contains 1061 stereo pairs with no trajectory loops (the sequence 12 from the KITTI odometry benchmark), our algorithm achieved the third best rotational error. There was an almost constant difference in the rotational error of $\sim 0.0005\,deg/m$ between our algorithm and ORB-SLAM2. The rotational error of our algorithm varies between $0.2\,deg$ (path length=$100m$) and $1.12\,deg$ (path length=$800m$). The rotational error of ORB-SLAM2 varies between $0.25\,deg$ and $1.47\,deg$. The translational error of ORB-SLAM2 is slightly (up to 0.05 %) smaller than the one of our algorithm. On paths of $600m$ length, our algorithm has a translational error of 0.77 % while ORB-SLAM2 has a translational error of 0.72 %.
The results from a second sequence of 921 images (the sequence 11 from the KITTI odometry benchmark) with no loops show that our algorithm has both a smaller translational error as well as smaller rotational error for all the path lengths (100,200,..,800) compared to ORB-SLAM2. Our algorithm has an almost constant translational error of $\sim 0.5\%$ while the translational error of ORB-SLAM2 varies between 0.60 % and 1.13 %. Our results confirm that which is a frame-to-frame visual odometry with no bundle adjustment optimization, has the largest translational and rotational errors. This applies for all path lengths.

For trajectories with loops, no significant improvement was made by our refinement step to the ORB-SLAM2 algorithm. This behavior has been seen on trajectories with one single loop and it becomes even more obvious when there are multiple loops on the trajectory.

Chapter 4

Line Segment based Efficient Large Scale Stereo Matching

In the previous chapter, we have discussed the use the of direct image alignment to refine feature based SLAM. Depth images were needed at keyframes to perform this alignment. We estimate depth maps from dense stereo disparity maps. The depth maps serve for a second purpose on our quadcopter MAV: the perception of the robot environment. Using the forward-facing stereo camera, the robot can estimate the geometry of the visible parts of scene in front of it. In the next chapter, we fuse the depth maps from different viewpoints to build a consistent volumetric map of the environment. As our goal is to achieve meaningful fully autonomous flights such as planning and following collision-free trajectories, the dense stereo algorithm needs to be computationally efficient. It should meet the real-time requirements while running alongside other algorithms such as path planning. In this chapter, we introduce our dense stereo matching algorithm (LS-ELAS), which we use for estimating depth maps.

Large parts of this work have been pre-published in Ait Jellal *et al.* (2017).

4.1 Introduction

Depth estimation is an important task in mobile robotics. It is essential for many tasks such as 3D reconstruction, view synthesis, obstacle avoidance, navigation, recognition and object grasping. The introduction of RGB-D sensors[1] with their sufficiently accurate depth maps and their real-time capability on dedicated hardware has accelerated the research on indoor mobile robotics. Since these Kinect-style RGB-D sensors are based on the projection and capture of structured infra-red light, they are not designed for outdoor usage with substantial sunlight. There is a need for more adequate sensors. Stereo cameras are the most adequate depth estimation sensors to be used for a wide range of outdoor and indoor applications. Stereo cameras provide data at high frame rates, they are lightweight, passive, energy efficient and customizable. This makes the hardware side of a stereo system very convenient. The software side is still problematic because

[1] e.g. Microsoft Kinect, Asus Xtion Pro or Intel RealSense

of the trade-off between accuracy and computational efficiency. In this work, we propose an efficient and accurate dense stereo algorithm which can facilitate the migration of systems initially designed to work in indoor environments using Kinect-style RGB-D sensors, to operate outdoors.

We developed a dense stereo matching algorithm based on the popular ELAS algorithm to estimate depth from stereo vision. We present four contributions in our work. (1) An efficient method for edge extraction, which provides the connectivity of the edge components as well. (2) A new method based on edge features for computing the support matches. This method is selective and does find support points which are more informative and allows preserving the depth discontinuity. It is more efficient as well, since the candidate support points that we use are more likely to have robust correspondences. (3) A new prior for the Bayesian inference approach proposed by ELAS. Our prior better represents the probability distribution of the disparity given the set of support points, the set of line segments and the observations. (4) An adaptive method for sampling support points along the edge segments.

4.2 Motivation

The depth estimation from stereo vision suffers from the problem of quadratic increase of the depth measurement error with respect to the measured depth. For a given depth measurement Z, if the depth error is assumed to be a Gaussian distribution, then the standard deviation of the measurement error can be given by Eq. 4.1 (see Chang and Chatterjee (1992)).

$$\sigma_Z = \frac{\sigma_d}{fB} Z^2 \tag{4.1}$$

Where f is the focal length, B is the baseline distance and σ_d is the disparity standard deviation.

In particular, distant points have high uncertainty on their depth estimate. Since the reconstructed X, Y coordinates depend on the depth Z (see page 31), they are also affected by the uncertainty of Z. It is worth to notice that other range sensors such as the Kinect 1.0 RGB-D camera also suffer from the problem of increasing range measurement error with respect to the measured depth. While laser scanners do not suffer from this problem and they provide more accurate depth estimates, they have in general large weight and have high power consumption, making then not suitable for a quadcopter MAV. Since we use stereo vision on our MAV quadcopter, we need to mitigate the afore-mentioned problem. One option would be to increase the baseline (see Fig. 4.2). However, this comes with the advantage of not recovering depth for nearby objects. Furthermore, increasing the baseline is strongly constrained for a quadcopter MAV. A more elegant way to mitigate the problem is by increasing the image resolution and exploring large disparity search intervals (see Fig. 4.2). Considering large disparity search intervals might be very inefficient. In this case, even stereo algorithms with linear-time ($\mathcal{O}(d)$) complexity

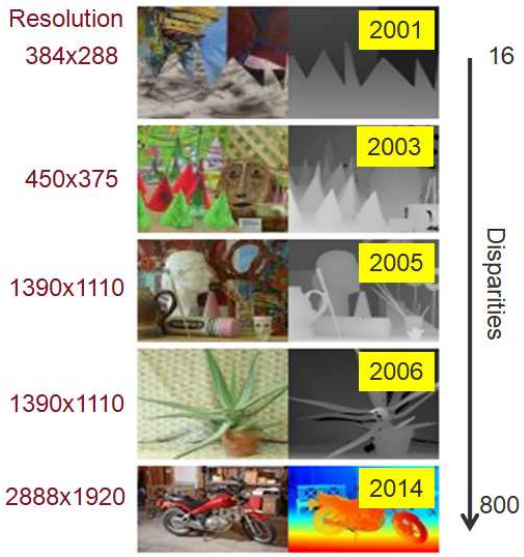


Figure 4.1: Evolution over time of the Middlebury stereo benchmark datasets. The image resolutions as well as the disparity search intervals have significantly increased over time.

(such as block matching) become inefficient. We note that we analyze the complexity only with respect to the disparity d and ignore the number of the pixels $(width \times height)$ of the image. The global stereo matching in this case becomes clearly unfeasible for real-time applications on standard hardware. Currently, most real-time dense stereo matching algorithms, which meet real-time requirements for mobile robotic applications, are from the category of local stereo matching algorithm with linear-time complexity with respect to the disparity. When dealing with large disparity search intervals, the disparity computation in linear-time ($\mathcal{O}(d)$) is still expensive to be estimated in real-time. Exceptions to this rule are algorithms which use highly parallel computing units such as GPUs and FPGAs. This implies that the constant-time and near-constant-time algorithms are the best choice to deal with large disparity search intervals and high resolution images. Independently of the size of the disparity search interval, the constant-time algorithm for stereo matching constrains the search to a small fixed size subset of candidate disparities. For example, instead of exploring the entire disparity search interval for a stereo pair (Fig. 4.1) of up to 800 candidate disparities, the search can be restricted to 5 disparity candidates. Table 4.3 on page 90 shows the resolution and the number of possible disparities of the newest Middlebury stereo dataset (version 3, 2014).

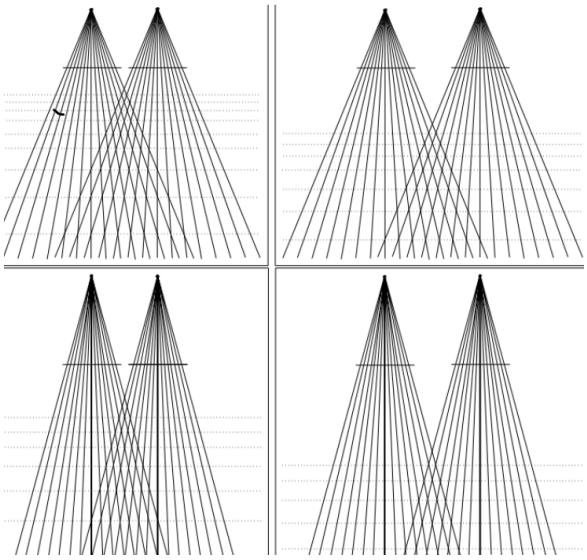


Figure 4.2: Two methods for increasing the depth accuracy for distant points. Top left: original stereo camera. Top right: increasing the baseline. Bottom left: increasing the image resolution. Bottom right: increasing both the baseline and the image resolution. This figure is copied from Moons *et al.* (2010).

4.3 Related work

Many approaches have been proposed to solve the dense stereo matching problem in the last 40 years. Scharstein and Szeliski (2002) showed that most of these approaches follow a common pipeline. This pipeline has the following four steps:

1. Computation of the matching cost using a similarity measure function. Usually some pre-processing of the images is performed prior to the computation of the matching costs. Removing the noise from the images using filtering techniques often improves the results. Examples of cost functions are: the sum of absolute differences (SAD), the sum of squared differences (SSD), the zero-mean SSD (ZSSD), the mutual information (MI) and the Hamming distance.
2. Aggregation of the matching costs, which are computed on the first step, over a given neighborhood. A simple aggregation of the matching costs could be performed by computing the SAD on a square around the candidate pixel location.
3. Optimization of the matching results. This step selects the best matching candidate from the list of candidate matching pixels on the target image. According to how the optimization of the matching cost is performed, one can distinguish between two types of stereo matching algorithms: the local stereo matching algorithms and the global stereo matching algorithms. On global stereo matching all of the pixels on the image contribute for the computation of the corresponding pixel.

4. Refining the disparity maps. In this step, wrong matches can be identified, filtered and sometimes corrected. The left-right consistency check is a widely used technique for refining the disparity maps and can provide robustness against occlusion.

Local stereo matching algorithms
In local stereo matching algorithms (e.g. Yoon and Kweon (2006), Einecke and Eggert (2010) Ma *et al.* (2013), Kanade and Okutomi (1994), Ait Jellal and Zell (2015)), the disparities are computed for each pixel individually, with the cost aggregated over a local correlation window. The choice of the size and the shape of the correlation window is both crucial and challenging. A correlation window with a large size can lead to edge blurring at the depth discontinuities. A correlation window with a small size, on the other hand, can significantly decrease the signal to noise ratio. This problem increases the ambiguity of the candidate disparities and as a result it increases the amount of outliers. A customized shape and size of the correlation window (Kanade and Okutomi (1994)) for each pixel individually comes at the cost of increasing the computational time. Hirschmüller *et al.* (2002) proposed to use multiple correlation windows and a border correction method. In this case it is not surprising that some adaptive window local stereo matching algorithms are even slower than some efficient global stereo matching algorithms. Bleyer *et al.* (2011) proposed slanted support windows. They compute 3D planes at the level of pixels and project the support region onto the local 3D plane. Einecke and Eggert (2010) proposed a two-stage correlation method for computing the scores. This method shows less sensitivity to high contrast outliers.

Global stereo matching algorithms
In global stereo matching (e.g. Mozerov and van de Weijer (2015) Boykov *et al.* (2001) Yang *et al.* (2010) Klaus *et al.* (2006)), the whole pixels of the image contribute to the computation of the disparity of a given pixel. In addition to the data term the energy function does include a smoothness term, which penalizes the disparities, which do not agree with the disparities of the neighbors. The complexity of these algorithms is usually $\mathcal{O}(d^c)$, where d is the disparity range and c is the size of the maximum clique potential. c is usually set to $c = 2$ for speeding-up the global stereo algorithms. Nevertheless, global stereo algorithm remain very slow for mobile robotics applications. An efficient global stereo algorithm is proposed by Felzenszwalb and Huttenlocher (2006). They come up with a method for updating the messages in linear-time $\mathcal{O}(d)$ for the three models of smoothness term: the potts model, the linear model and the quadratic model. However, the huge memory requirement needed for storing the messages and the fact that there is no guarantee for convergence for loopy graphs are two major drawbacks of this method. Another efficient algorithm is the semi-global stereo matching (SGM) from Hirschmüller (2008). In SGM, the costs are first computed using local stereo matching in linear-time $\mathcal{O}(d)$. Then, a (global) smoothness step is applied for updating the costs by penalizing the disparities that disagree with those of the neighbors. It considers interactions that are potentially very long range along image rows, columns and diagonals to minimize a

cost function. The semi-global characteristic is due to the fact that only a small subset of possible interaction paths is considered.

Constant-time and near-constant-time algorithms
The classification of stereo matching algorithms into local and global is based on the way the disparity optimization is done (step 4 of the stereo pipeline, see page 72). Since we are interested in real-time algorithms, we focus on the classification based on the computational time complexity. Most local stereo algorithms have linear-time with respect to the disparity $\mathcal{O}(d)$ and most global stereo algorithms have a quadratic-time with respect to the disparity $\mathcal{O}(d^2)$. Algorithms with higher order complexities are usually impractical for robotics applications. Stereo matching algorithms, which smartly restrict the search domain to a small interval, such as Sinha *et al.* (2014), Bleyer *et al.* (2011) and Geiger *et al.* (2011a), are becoming more popular. This category of algorithms deals efficiently with large scale images. While Sinha *et al.* (2014) involves a global optimization stage, ELAS (Geiger *et al.* (2011a)) is a near-constant-time local stereo matching algorithm.

The ELAS algorithm from Geiger *et al.* (2011a)
ELAS deals efficiently with large scale images. For computing the disparities, ELAS explores the entire disparity search interval only for a very small subset of the pixels. For all other pixels the search is restricted to a small fixed window of candidate disparities. ELAS starts by finding robust matches for some pixels called support points and generates from them a 2D mesh by Delaunay triangulation. The support points are the triangle corners. The matching of these triangle corners is done in linear-time ($\mathcal{O}(d)$). Afterwards the algorithm uses Bayesian inference with the assumption that the disparity of a pixel is independent of all other pixels on the reference image, given the disparities of the triangle corners it belongs to. For the pixels inside the triangles the matching is done in constant-time ($\mathcal{O}(1)$).

4.4 The LS-ELAS Algorithm

In this section, we describe in detail LS-ELAS, which is our dense stereo matching algorithm. LS-ELAS stands for line segment based efficient large scale stereo matching. LS-ELAS is an extension of the popular ELAS algorithm (Geiger *et al.* (2011a)). Similar to the original ELAS, LS-ELAS computes the disparities in constant-time for most of the pixels in the image and in linear-time for a small subset of the pixels (support points). Our approach is based on line segments to determine the support points instead of uniformly selecting them over the image range. This way we find very informative support points which preserve the depth discontinuity. The prior of our Bayesian stereo matching method is based on a set of line segments and a set of support points. Both sets are plugged into a constrained Delaunay triangulation to generate a triangulation mesh

↖ 1	↑ 2	3 ↗
← 0	x	4 →
↙ 7	↓ 6	5 ↘

5	6	7
4	x	0
3	2	1

Figure 4.3: The left table contains the direction labels for the 8 adjacent pixels around the pixel x. The table on the right contains the inverse direction labels that are used for an inverse search.

which is aware of possible depth discontinuities. We further increased the accuracy by using an adaptive method to sample candidate points along edge segments.
In this section, we describe our algorithm in detail. We start by describing our method for extracting the edge segments and the way we compute the set of support points and the set of line segments. Then, we show the mathematical derivation of our Bayesian approach for the stereo problem.

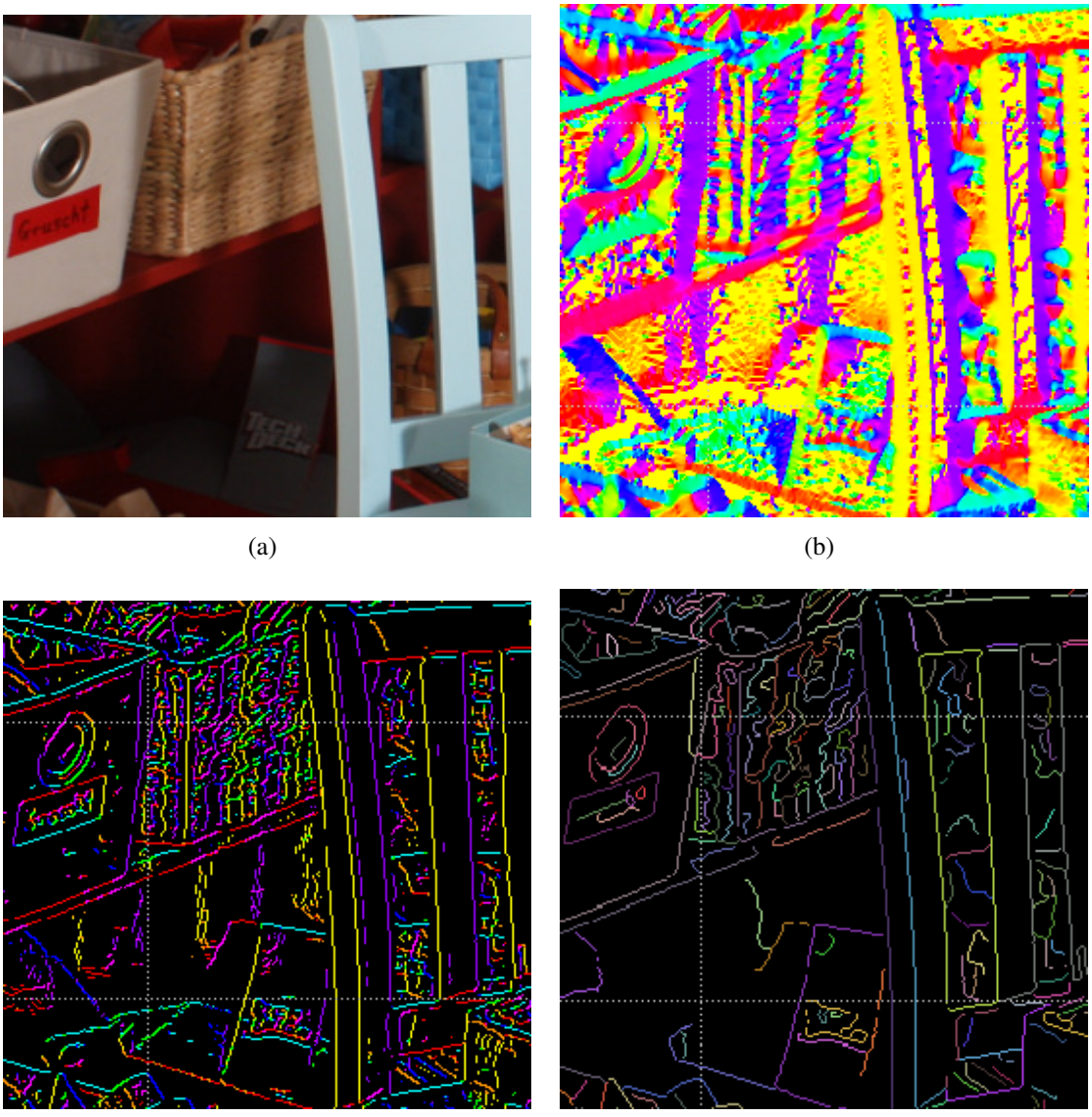
4.4.1 Edge extraction

The LS-ELAS algorithm, which is published in the ICRA-2017 conference (see Ait Jellal *et al.* (2017)), was a joint work between our cognitive systems department and the department of visual computing. Both departments are at the university of Tübingen in Germany. This section of our dissertation (i.e. Section: 4.4.1 Edge extraction) was developed and written by Benjamin Wassermann and Manuel Lange from the department of visual computing. For an easy understanding of the LS-ELAS algorithm, we describe in this section the efficient new method for edge segment extraction from Benjamin Wassermann and Manuel Lange. As their method has not been yet published (October 2019), we could not provide a reference to their work.

In ELAS (see Geiger *et al.* (2011a)), the support point candidates (the set $\boldsymbol{C} = \{c_{ij}\}$) are sampled uniformly over the image (see Fig. 4.7(a)). This method is motivated by the need for an efficient way to compute the support points. Other methods like using SIFT (Lowe (2004)) features for support point candidates have shown no significant improvements (Geiger *et al.* (2011a)). In this chapter, we present a new method for extracting support point candidates based on edge features. Our approach is motivated by the efficient edge extraction method of Benjamin Wassermann and Manuel Lange. We show that our method for computing the support matches is both robust and efficient. Furthermore, it allows us to use the line segments between two consecutive support points to generate a better triangulation.

Our fast edge segment extraction method is similar to the Canny edge detector approach (see Canny (1986)):

1. The input image is smoothed by a Gaussian filter to reduce noise.
2. The intensity gradients of the image are computed.



(a) (b)

(c) (d)

Figure 4.4: Intermediate steps of our edge segments extraction method. (b) shows the dirMap (gradient directions). (c) shows the edgeMap. The color encodes the gradient direction (same as in dirMap). (d) shows the final edge segments list. Each edge segment is drawn by a random color individually.

Algorithm 1: Main loop for extracting edge segments.

```
Data: seedList, edgeMap, dirMap, dir2Index
Result: edgeSegmentList
foreach seed in seedList do
    segForward = extractSeg(seed);
    reverse(dir2Index);
    segReverse = reverse(extractSeg(seed));
    edgeSegmentList.push(segReverse + segForward);
```

Algorithm 2: adjacentIndex: Find adjacent edge pixel.

```
Input: index, direction
Data: edgeMap, dir2Index
Result: adjIndex
for each adjIndex pixel do
    if edgeMap[adjIndex] > 0 then
        // Invalidates the pixel from seedList;
        edgeMap[adjIndex] = -1;
        return adjIndex;
return -1;
```

Algorithm 3: extractSeg: Extract single edge segment.

```
Input: index
Data: edgeMap, dirMap
Result: edgeSegment
while index >= 0 do
    edgeSegment.push(index) ;
    index = adjacentIndex(index, dirMap[index]) ;
return edgeSegment;
```

3. Non-maximum suppression is applied for thinning the spread edge responses to the maximum responses.
4. Two thresholds are applied to determine potential strong and weak edge pixels.
5. The edges are tracked by hysteresis to suppress weak edges that are not connected to strong edges.

The result of the classical Canny method only provides an edge pixel map without the connected edge components. We want to have a data structure which holds all pixels which belong to the same edge segment (see Fig. 4.4(d)). We need to split the edge map to individually determine the edge segments. So, we extended the Canny method as follows:

1. Within the third step of Canny the gradient orientation for each pixel has to be known. Fast algorithms compute a rough approximation for the 8 possible neighbouring pixels. In our method we store this information in a map for later use.
2. In our approach all edge responses over the lower threshold are marked in an edge map (see Fig. 4.4(c)). The indices of the edge responses over the higher threshold are stored in a seed list.
3. The hysteresis step is replaced by the connected edge components search (see Fig. 4.3).

Starting from a seed point and its initial direction (Canny modification 2 and 1), a connected edge component is extracted by traversing the adjacent edge points (see Algorithm 3). Instead of testing all 8 adjacent pixels, we use the stored direction to predict the next adjacent edge point as shown in Algorithm 2. The seed point could be the start point or the end point of a segment. For an end point, the search direction has to be reversed. It is also possible that the seed is a point in between the segment, which requires a reverse and a forward search. The ordered connected edge points are stored in an edge segment list. For more details see Algorithm 1.

4.4.2 Support points matching along edges

The process of computing the support points is illustrated in Fig. 4.7. We start by sampling candidate support points along the edge segments. A candidate support point includes the pixel coordinates (u, v) and the disparity d (d is initially set as invalid). The set of support points is $\boldsymbol{S}$. We set a feature vector for each candidate support point based on the gradients in X and Y direction around it. The feature descriptor is shown in Fig. 4.5. To match the support point candidates, we use the sum of absolute differences to compute the cost. We perform a left-right consistency check to filter the outliers. Since the feature vector is 32 elements of 8 Bits each, the 256 Bits feature vector does fit in the AVX2 (Advanced Vector Extensions 2 SIMD instructions) CPU registers and the

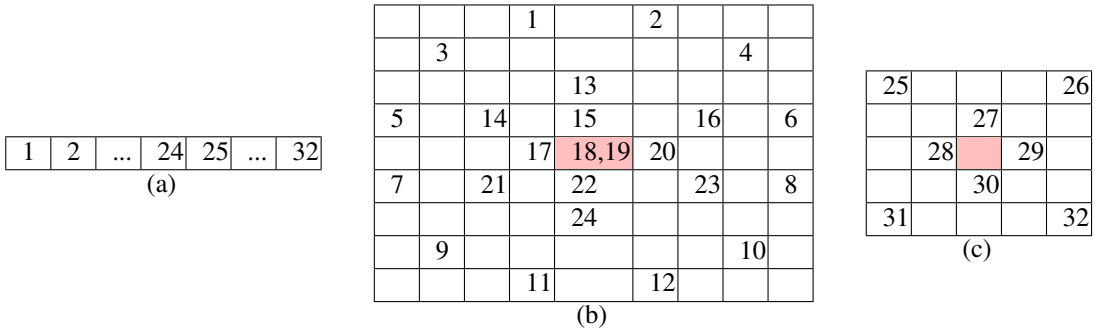


Figure 4.5: (a) the feature vector is composed of 32 elements of 8 Bits each. These elements are obtained from the image gradients (Sobel). (b) locations of the elements 1-24 of the feature vector on the X-Sobel gradient image. (c) locations of the elements 25-32 of the feature vector on the Y-Sobel gradient image. The candidate support point is at the center (red cell). The Sobel responses are normalized and shifted to have the size of 8 Bits and values between 0 and 255.

computation is accelerated using the AVX2 instructions. Table 4.1 shows a comparison of the percentage of successfully matched support points. The AVX2 is used only for support points matching. For pixels inside triangles we use a 128 Bits feature vector and SSE2 SIMD instructions as in ELAS.

We present two different methods for sampling candidate support points. In our first method, we uniformly sample candidate support points along the edges. A constant step separates two consecutive candidate points. This constant step is calculated from the image diagonal length. We present also a more elegant method for sampling the support point candidates. This sampling method is an adaptive method. Here we sample more points on curved parts of edges than on straight parts. We use an iterative process where we walk along the edge and consider the straight line between the last recently sampled support point candidate (the start) and the current point (the end). When the "curvature" (the projection distance of any of the points from the currently checked part onto the straight line, which lies between the current start and the current point) is over a certain threshold, then a new support point candidate is set. This point is the new start for the next steps of the walk. In case the curvature is low, we also set a new support point candidate with a constant step. This process is illustrated in Fig. 4.9.

4.4.3 Probabilistic disparity estimation

We extend the Bayesian approach introduced by Geiger et al. in ELAS (see Geiger *et al.* (2011a)) by including a list $\boldsymbol{L} = \{l_{i_1 j - i_2 j}\}$ of straight line segments in the probability model. $l_{i_1 j - i_2 j}$ is a straight line segment between two consecutive (matched) support points $s_{i_1 j}$ and $s_{i_2 j}$. The set of (matched) support points ($\boldsymbol{S}$) and the set of line segments

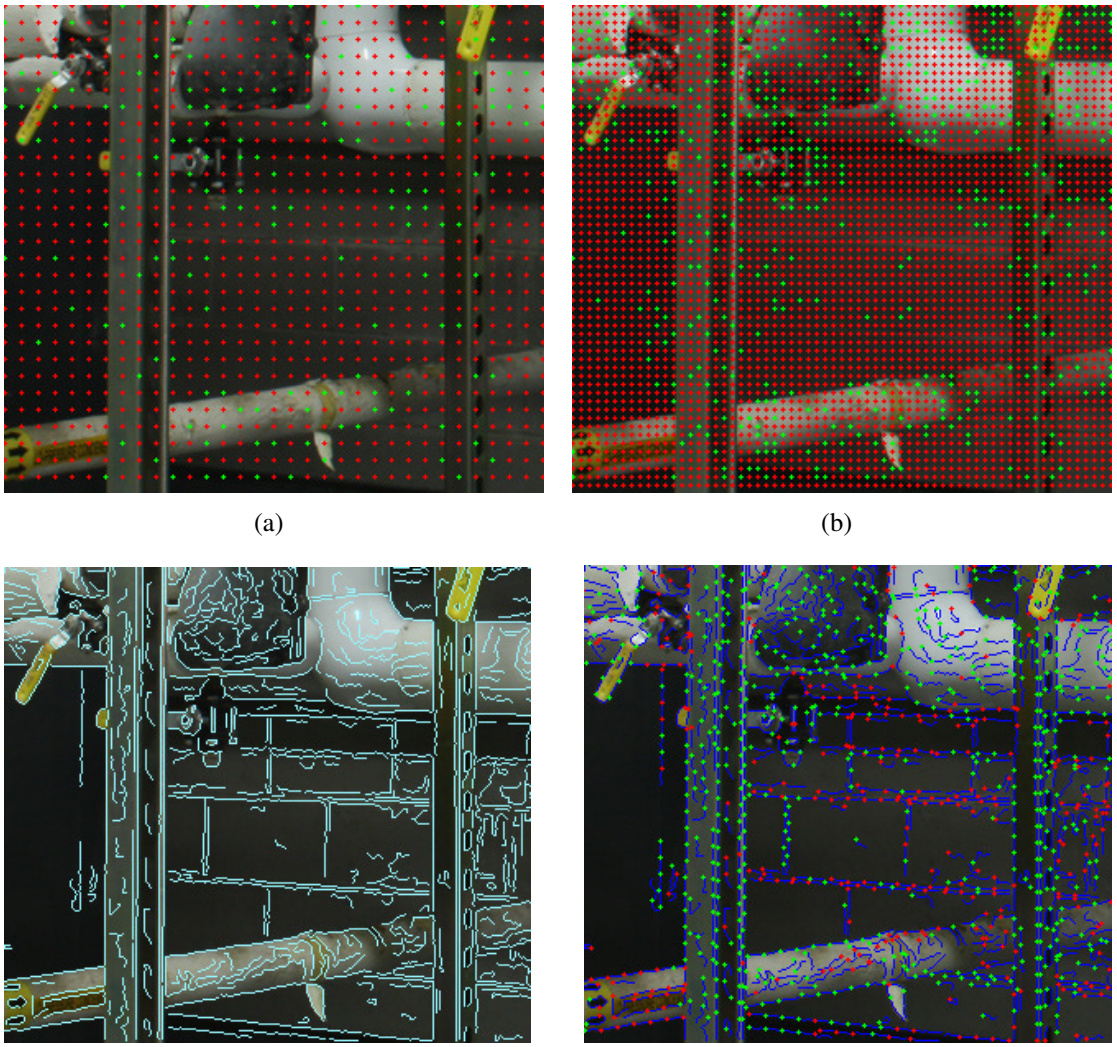
(a) (b)

Figure 4.6: (a) shows the left image of the Middlebury 2014 Pipes dataset from which we select the region defined by the red rectangle. (b) shows this area.

($\boldsymbol{L}$) are plugged into a constrained Delaunay triangulation algorithm to compute a mesh of triangles. In other words, the input to our triangulation is the set of all support points that have valid matches (this includes the isolated support points, which do not belong to any line segment $l_{i_1 j - i_2 j}$) and the set of line segments $\boldsymbol{L}$. The constrained Delaunay triangulation is a triangulation, which is forced to include the input edges (our line segments list $\boldsymbol{L}$). This way we nicely preserve the object boundaries. Fig. 4.8(a) shows the 2D mesh created by our method LS-ELAS. The line segments are represented with the green color. Eq. 4.2 shows the mathematical formulation of our prior:

$$p(d_n \mid \boldsymbol{S}, \boldsymbol{L}, o_n^{(l)}) = \begin{cases} \gamma + \exp\left(-\frac{(d_n - \mu(\boldsymbol{S}, \boldsymbol{L}, o_n^{(l)}))^2}{2\sigma^2}\right), \text{if } |d_n - \mu| < 3\sigma \vee d_n \in N_S \\ 0, \text{otherwise} \end{cases} \tag{4.2}$$

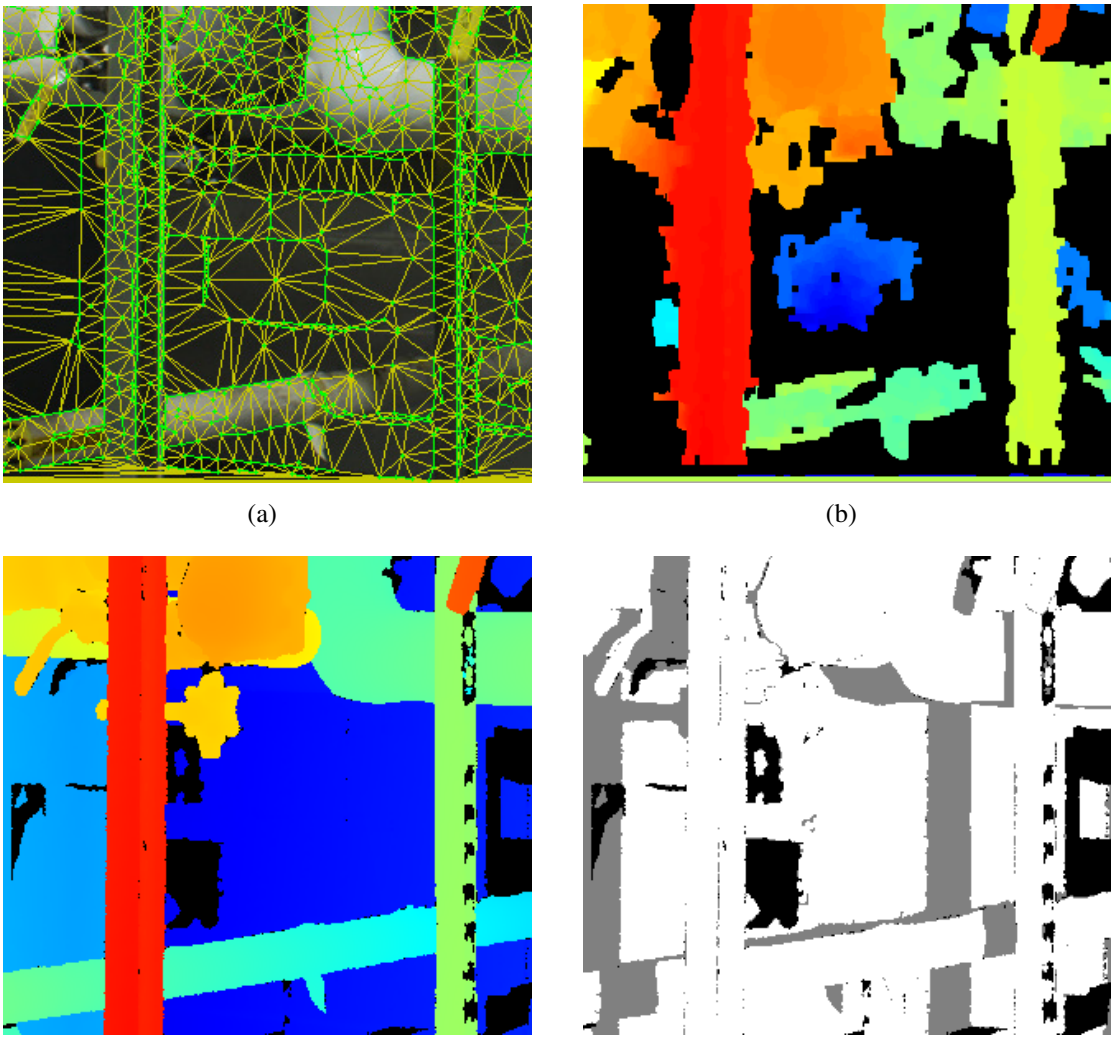
It includes a Gaussian part and a uniform part. The mean $\mu(\boldsymbol{S}, \boldsymbol{L}, o_n^{(l)}))$ in the Gaussian part depends on the set of line segments $\boldsymbol{L}$. The mean disparity $\mu(\boldsymbol{S}, \boldsymbol{L}, o_n^{(l)}))$ is obtained by interpolating the disparities of the corners (support points) of the triangles. Using the constrained Delaunay triangulation to generate the triangle mesh might result in a mesh, which does violate the Delaunay property locally for some triangles, because the constrained Delaunay triangulation is enforced to include the input line segment set $\boldsymbol{L}$. We remind that the Delaunay property means that no point is contained inside the circumcircle of any triangle in the mesh. This means avoiding skinny triangles with large



(a) (b)

(c) (d)

Figure 4.7: (a) (b) show the support point candidates c_{ij} (*red* and *green*) and the valid support point s_{ij} (green) obtained using ELAS method with a step of 10 and 5 pixels respectively. (c) shows the edges (in light blue) detected with our method. (d) shows the support point candidates c_{ij} (*red* and *green*) and the valid support point s_{ij} (green) obtained using our algorithm.



(a) (b)

(c) (d)

Figure 4.8: (a) shows the 2D mesh generated using our method for the example in Fig. 4.7. The discontinuity edges are nicely preserved. (b) shows the disparity map estimated using our algorithm (LS-ELAS). No hole filling post-processing is done and most black regions correspond to occluded regions (and they should be black). (d) shows the occlusion map for the Pipes dataset. The black grey pixels correspond show the half-occlusions. The black pixels show the invalid pixels for which the reference system could not find a reliable ground truth disparity.

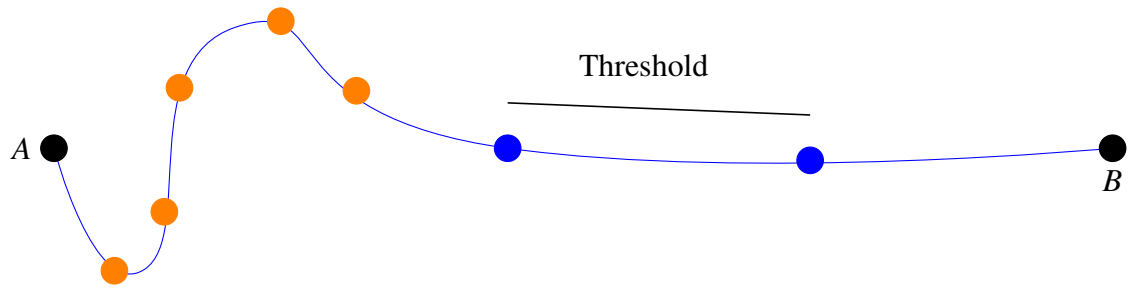


Figure 4.9: Illustration of the adaptive sampling on an edge segment AB. We sample more candidates support points on the regions with large curvature (brown points). On the flat regions of the edge segment, there are fewer candidate support points (blue points).

height to base ratio. We prefer to have a mesh which preserves the discontinuity rather than a mesh which conforms to Delaunay. The search domain is restricted to a small set (radius of 3σ pixels) of candidates disparities around the mean $\mu(\boldsymbol{S}, \boldsymbol{L}, o_n^{(l)}))$. For example we can restrict the search domain to 7 disparities ($\sigma = 1$) instead of a full range of 800 disparities. To account for depth discontinuity, the search domain is extended to include the disparities of the neighbor support points N_s.
Similar to Geiger *et al.* (2011a), we model the image likelihood using a constrained Laplace distribution (see Eq. 4.3).

$$p(o_n^{(r)} \mid d_n, o_n^{(l)}) = \begin{cases} exp(-\beta \| f_n^{(l)} - f_n^{(r)} \|_1), \\ \text{if } \begin{pmatrix} u_n^{(l)} \\ v_n^{(l)} \end{pmatrix} = \begin{pmatrix} u_n^{(r)} + d_n \\ v_n^{(r)} \end{pmatrix} \\ \\ 0, \text{otherwise} \end{cases} \tag{4.3}$$

The likelihood probability assumes that there is only one candidate match with non-zero probability. The likelihood probability evaluates the similarity between candidate matches based on local feature vectors (f_n^l and f_n^r). The "if condition" ensures that we search along the epipolar line (same v coordinate).
The posterior is the product of the prior and the likelihood. The maximun a-posteriori estimate of the depth map d_n^{MAP} is then obtained by maximizing the posterior probability (see Eq. 4.4).

$$d_n^{MAP} = \underset{d_n}{\text{argmax}}\, p(o_n^{(r)} \mid d_n, o_n^{(l)}) p(d_n \mid \boldsymbol{S}, \boldsymbol{L}, o_n^{(l)}) \tag{4.4}$$

Maximizing the posterior probability is equivalent to minimizing the negative logarithmic energy. Thus, to find the winning disparity map (d_n^{MAP}), we minimize the energy

function given by Eq. 4.5.

$$E(d) = \beta \| f_n^{(l)} - f_n^{(r)}(d) \|_1 - \log \left[\gamma + \exp \left(- \frac{(d - \mu(\boldsymbol{S}, \boldsymbol{L}, o_n^{(l)}))^2}{2\sigma^2} \right) \right] \tag{4.5}$$

where $f_n^{(r)}(d)$ is the feature vector at $(u^{(l)} - d, v^{(l)})$ on the right (target) image.
According to the definition of the prior probability (see Eq. 4.2), we need to evaluate this energy function (see Eq. 4.5) only for candidate disparities d, which satisfy either the condition $|d - \mu| < 3\sigma$ or the condition $d \in N_S$. As a result, the computation of the disparities (d) can be done in constant-time with respect to the disparity ($\mathcal{O}(1)$). We remind that this applies for the pixels within the triangles (of the triangle mesh). For the triangle corners (i.e. the support points) the search for matches is done in linear-time with respect to the disparity ($\mathcal{O}(d)$). Thus, LS-ELAS is a near constant-time algorithm.

4.5 Evaluation

4.5.1 The Middlebury stereo benchmark version 3

We sought to objectively evaluate our stereo matching algorithm. We identified many requirements which we consider that they contribute to achieve an objective evaluation. On the following, we summarize the most relevant requirements:

- The dataset shall be challenging. In particular the images should have high resolutions and large disparity ranges to show the gain of a near-constant-time algorithm. Varying lighting conditions need also to be considered.
- The algorithm needs to be compared with other algorithms based on the same input data and the same performance metrics.
- Quantitative results of the performance of the algorithm shall be reported by the algorithm developers. The other algorithms should be tuned by their developers.
- A variety of performance measures needs to be considered to illustrate the strengths and drawbacks of each algorithm.
- The algorithm needs to run using the same parameters and shall not be tuned for each dataset individually.
- It is preferred to rank the evaluation of the algorithm on some test data (for which there exists no available public ground truth).

We consider that the evaluation of our algorithm by a third party on a publically available stereo matching benchmark is an effective way to achieve an objective evaluation. We chose to evaluate the LS-ELAS algorithm on the Middlebury Stereo Evaluation Dataset - Version 3 from 2014 (Scharstein *et al.* (2014)). At the time of writing this dissertation


Figure 4.10: The stereo pair DjembeL from the Middlebury benchmark has strong lighting differences between the left and right image of the stereo pair.

(October 2019), the Middlebury Stereo Evaluation Dataset - Version 3 is the newest version.

The Middlebury Stereo Evaluation Dataset - Version 3 from 2014

The Middlebury Stereo benchmark is the standard benchmark for binocular stereo matching algorithms. It contains a variety of different kinds of scenes. This new Middlebury dataset (version 3) is more challenging compared to the previous datasets (version 2). The dataset includes images taken in industrial environments, such as the Pipes dataset and images taken in indoor environments. The benchmark contains 15 stereo image pairs for training with available ground truth disparity images, as well as 15 stereo image pairs for testing, for which no ground truth disparity images are available. The stereo image pairs have names. It is allowed to submit the results for the test dataset only once.
The 15 stereo pairs of the test dataset are: Australia, AustraliaP, Bicycle2, Classroom2, Classroom2E, Computer, Crusade, CrusadeP, Djembe, DjembeL, Hoops, Livingroom, Newkuba, Plants and Staircase. Table 4.3 on page 90 shows the resolutions and the number of possible disparities of those stereo pairs.
The 15 stereo pairs of the training dataset are: Adirondack, ArtL, Jadeplant, Motorcycle, MotorcycleE, Piano, PianoL, Pipes, Playroom, Playtable, PlaytableP, Recycle, Shelves, Teddy and Vintge.
Every stereo pair is available in three different resolutions: Full ($\sim 3000\times2000$ pixels), Half ($\sim 1500\times1000$ pixels), Quarter ($\sim 750\times500$ pixels). The abbreviations: F, H and Q are sometimes used and they respectively refer to the Full, Half and Quarter resolutions.

The Middlebury Stereo benchmark is very challenging. The images have high resolution of up to 6MP. The disparity range reaches 800 disparities. It includes versions with changed exposure and changed lighting. We mean here that the reference image have strong differences on exposure time and lighting compared to the target image of the same stereo pair (see 4.11). The impact of bad camera calibration can also be evaluated using images pair with imperfect rectification: for a pixel (x_l, y_l) on the reference image, the corresponding pixel (x_r, y_r) on the target images does not have the same y coordinate.

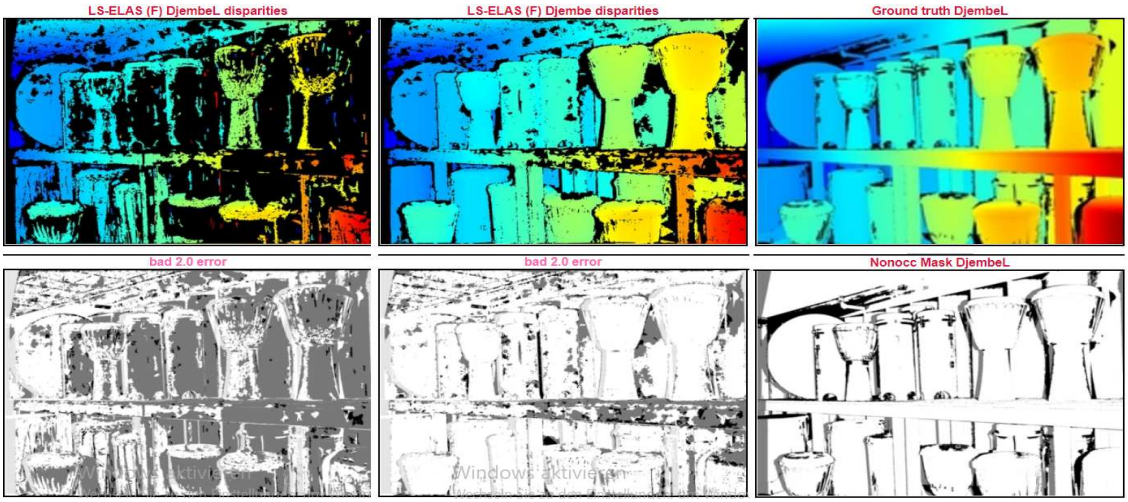


Figure 4.11: Chromatic changes. Top row: the estimated disparity maps for the stereo pair DjembeL (left) and Djembe (middle). Top row right: the ground truth. Second row: the corresponding error maps encoded in gray levels.

Figure 4.12: The stereo pair Classroom2E from Middlebury benchmark has strong exposure time differences between the left and right image of the stereo pair.

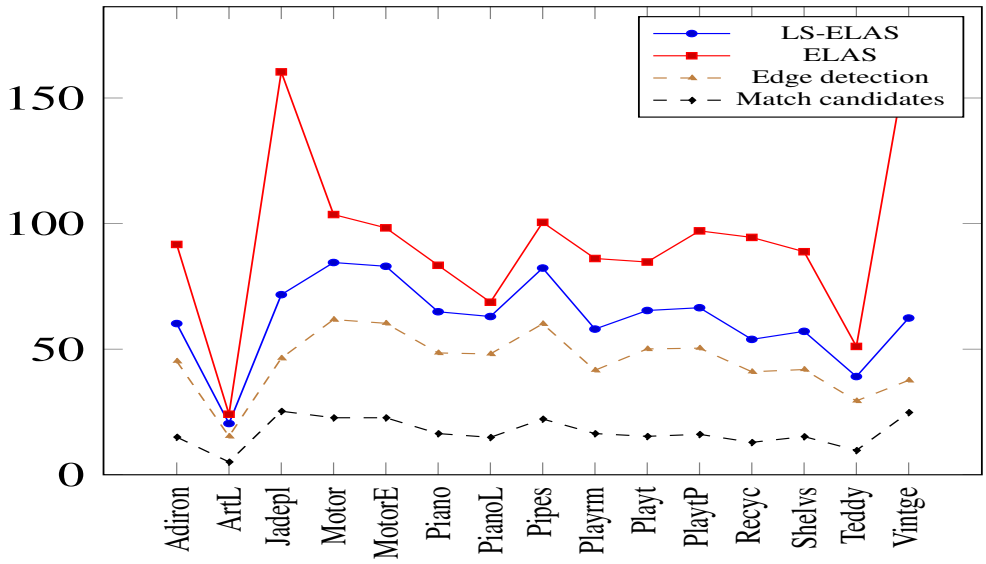


Figure 4.13: X-axis: the different dataset images. Y-axis: the time in ms for computing the support matches. The results from ELAS are in drawn in blue and those of LS-ELAS in red. For LS-ELAS the time is the sum of the edge detection step (brown) and the support candidate matching step (black). The results are reported for the training data set at half resolution ($\sim$ 1500$\times$1000 pixels. On the full resolution ($\sim$ 3000$\times$2000 pixels), our algorithm is even more efficient compared to ELAS (see Fig. 4.14).

We performed experiments on the Middlebury Stereo Evaluation Dataset - Version 3 from 2014 Scharstein *et al.* (2014) to evaluate our adaptive LS-ELAS algorithm. We conducted all our running time measurements on a notebook computer equipped with 16 GB RAM and an Intel Core i7 4700MQ, 2.4 GHz (3.4 GHz turbo), 6 MB cache. All results are single core performance. We have submitted our results to this benchmark, they are listed as "LS-ELAS" for broader comparison with other algorithms[2]. Our results are particularly good in the sparse datasets, because we did not implement a good post-processing method for correcting invalid matches (hole filling). In the rest of this section, we focus on the sparse evaluation. We remind that in the Middlebury benchmark, sparse disparity map means a "dense" disparity map for which the hole filling step is not performed. Most of the holes (filtered pixels) occur at half-occlusions on which the stereo matching is an ill-posed problem. We choose *nonocc* as mask to exclude occluded pixels from the computation of the performance metrics. The training and test datasets contain 15 stereo pairs each. The average (over all stereo pairs) is calculated by weighting the images. The weighting differs and the weights are set by Middlebury.

[2]http://vision.middlebury.edu/stereo/eval3/

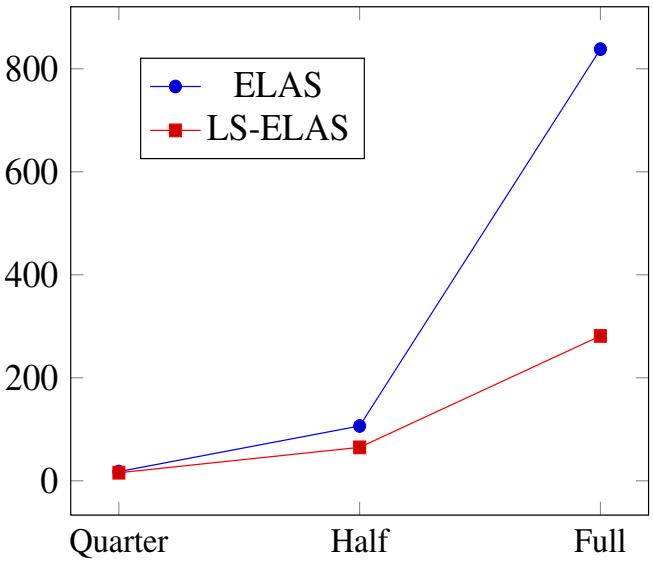


Figure 4.14: Evolution of the average time (in ms) for computing the support matches with respect to the image resolution. Time averaged for 15 stereo-pairs of the training dataset. X-axis: image resolutions: Full ($\sim 3000\times2000$), Half ($\sim 1500\times1000$), Quarter ($\sim 750\times500$). Y-axis: number of disparities.

4.5.2 Comparing LS-ELAS with ELAS

In this subsection, we compare the LS-ELAS algorithm with the original ELAS algorithm. We start by comparing the step concerning support point matching of both algorithms. We then compare the whole algorithms according to the different metrics of the Middlebury benchmark.
In Fig. 4.13, the time for the support point calculation and matching is drawn for the images of the training dataset in half resolution (typical image size 1500×1000 pixels). As ELAS places the support points on a uniform grid (5×5) over the image, calculating the position of the support point candidates is done immediately. We first have to detect the edge segments in the image. Then, we sample our support point candidates along them. This detection takes some time, which is visualized as *Edge detection*. After having placed our support point candidates along the edge segments, their matches in the target image are calculated. That time is plotted as *Match candidates*. The sum of both our steps is drawn as LS-ELAS to compare against the time ELAS takes in its corresponding steps. We can see that our method is faster, even though it performs the additional edge detection. This is because our candidate support points are much more likely to successfully match and the size of the candidate support points set is considerably smaller. By sampling our support along the edges we increase the chance to match them because it is guaranteed to have a given extent of texture differences. In particular, no support points are sampled on uniform areas and thus we do not waste time on those

areas. Table 4.1 shows the percentage of matched support point candidates from the two representative stereo pairs (Pipes and Arlt) from the Middlebury stereo benchmark. This table shows that LS-ELAS requires significantly fewer support points candidates than ELAS to obtain the same amount of matched support points. While the percentage of matched points exceeds the 50% in case of LS-ELAS, it only reaches $\sim 10\%$ in case of ELAS. Support point candidate matching needs some time because the whole disparity range has to be searched ($\mathcal{O}(d)$ complexity) plus the left-right consistency check. Having fewer support point candidates, we can make up for the additional time we need to detect edge segments.

Fig. 4.14 shows the comparison of the running time of the support point matching step, between LS-ELAS algorithm and the original ELAS algorithm. The Y-axis is the average time in *ms* of the images of the Middlebury Stereo Evaluation Dataset. The image size is on the X-axis. Q stands for quarter resolution, H and F for half and full resolution correspondingly. The average time for the quarter resolution images is $15.7\,ms$ for our method and $17.9\,ms$ for ELAS. While the $2.2\,ms$ difference is not significant, it takes 64% more time on the half resolution images. This increases to three times the calculation time on the full size images for the corresponding steps in ELAS. In other words, LS-ELAS requires $281.3\,ms$ for computing the support points on images of size 3000×2000, which is three times faster than ELAS ($838.2\,ms$). We can see that increasing the picture size affects ELAS much more than LS-ELAS. With a tendency to larger images, this difference becomes more relevant. The Table 4.2 shows the comparison results for all the Middlebury *metrics* on the *test sparse* dataset in full resolution. Our algorithm outperforms ELAS on all Middlebury metrics. The definitions of these metrics are described on page 91.

All the results above are from the adaptive sampling version of our algorithm. The version with uniform sampling is less accurate but slightly faster. The evaluation on the training sparse dataset showed that it is still more accurate than ELAS and most competing algorithms listed on Table 4.5. For example: "$bad2.0" = 11.2$ and "$Avgerr" = 3.56$.

Table 4.1: PERCENTAGE OF VALID SUPPORT POINTS

Dataset	Algorithm	Candidates	Matched	Percent
Pipes F	LS-ELAS	23311	15845	68%
	ELAS	228144	20834	9%
ArtL F	LS-ELAS	10797	6064	56%
	ELAS	61716	6529	10%

Table 4.2: Performance evaluation on the Middlebury Stereo Evaluation Dataset - Version 3, LS-ELAS vs ELAS

Algorithm	LS-ELAS vs ELAS: average times, Set: test sparse, Mask: nonocc												
	bad 0.5	bad 1.0	bad 2.0	bad 4.0	Avgerr	RMS	A50	A90	A95	A99	time	time/MP	time/GD
LS-ELAS (F)	**30.9**	**15.8**	**8.82**	**5.97**	**7.36**	24.6	**0.74**	**19.7**	**32.0**	**98.0**	**2.63**	**0.50**	**1.33**
ELAS (F)	45.2	26.4	16.4	11.5	8.53	**22.8**	1.55	20.1	34.5	102	3.00	0.56	1.45

Table 4.3: Description of the test dataset of the Middlebury stereo benchmark.

	Unit	Stereo pair														
		Aust	AustP	Bic2	Clas	ClasE	Comp	Crus	CrusP	Djem	DjemL	Hoop	Livgr	Nkub	Plant	Stair
Full resolution	MP	5.6	5.6	5.6	5.7	5.7	1.5	5.5	5.5	5.7	5.7	5.7	5.9	5.5	5.6	5.2
# Disparities	Pixel	290	290	250	610	610	256	800	800	320	320	410	320	570	320	450

4.5.3 Comparing LS-ELAS with other algorithms

We consider the following local and global binocular dense stereo algorithms to be relevant for comparison with our algorithm (see related work, page 72):

- Local methods: ELAS (Geiger *et al.* (2011a)), Cens5 (Hirschmüller *et al.* (2002)), SNCC (Einecke and Eggert (2010)).
- Global methods: SGM (Hirschmüller (2008)), SGBM1 (OpenCV 2.4.8 re-implementation of SGM), TSGO (Mozerov and van de Weijer (2015)), LPS (Sinha *et al.* (2014)), and NOSS-ROB (Superpixel alpha-expansion and normal adjustment for stereo matching. NOSS-ROB is a new algorithm. At the time of writing this thesis, the authors made an anonymous submission of their results into the Middlebury stereo benchmark because their paper has not yet been accepted).

We did not take IDR (Kowalczuk *et al.* (2013)) into the comparison, as it needs special hardware running only on a graphics card. Many robots can't supply the additional power, payload and space needed for a dedicated graphics cards.

The Middlebury stereo benchmark includes a set of performance measures which will be described below. The ranking can be done according to each individual metric. The Middlebury stereo benchmark maintains an online table where the results and ranking of other algorithms can be found. This is very important as it allows comparing our algorithm with other algorithms for which no open-source code is available. The online table allows to compare our results with the results from newly inserted algorithms into the table as well.

The metric time/MP

A stereo matching algorithm, which is meant to be used in real-time on-board a quadcopter MAV, needs to be efficient. Comparing the algorithms according to their running time is important. The time per Megapixels metric (time/MP) is the time normalized by the number of pixels. The unit is $s/Megapixels$. Table 4.4 shows the detailed results according to the time/MP metric. A graphical visualization of this table is shown in Fig. 4.15. In average (over all stereo pairs), LS-ELAS needs "0.50s" to process $1\,MP$ of data and ELAS needs "0.56s". LS-ELAS and ELAS are by far more efficient than the competing algorithms. In Fig. 4.15, the curves for LS-ELAS and ELAS almost overlap. The results of the algorithms NOSS-ROB and Cens5 were reported using the half-resolution dataset (H) as opposed to all other algorithms, which used the full-resolution dataset (F). Unfortunately, we do not have access to the source code of those algorithms to measure the time on the full-resolution dataset (F). The stereo pairs Crus, CrusP, ClassRoom, and ClassRoomE have the largest disparity search ranges with a maximum disparity of up to 800 disparities. While the constant-time algorithms show no increase in the $time/MP$

for processing these stereo pairs, all other algorithms exhibit a significant increase in the value of $time/MP$ metric. NOSS-ROB needs on average $502s$ for processing every MP of data, while the second slowest algorithm (SGM) needs $13.4s$ for processing the same amount of data. As can be seen on the accuracy evaluation sections (page 94), SGM wins the ranking according to most performance metrics. This comes at the cost of being 26.8 times slower than LS-ELAS. For a better visualization, the results of NOSS-ROB are not plotted in Fig. 4.15. The stereo pair Compu has a small size (i.e. $1.5MP$ see Table 4.3 page 90) compared to all other stereo pairs, which have sizes between $5.2MP$ and $5.9MP$. As a result, we excluded Compu from the plot on Fig. 4.15.

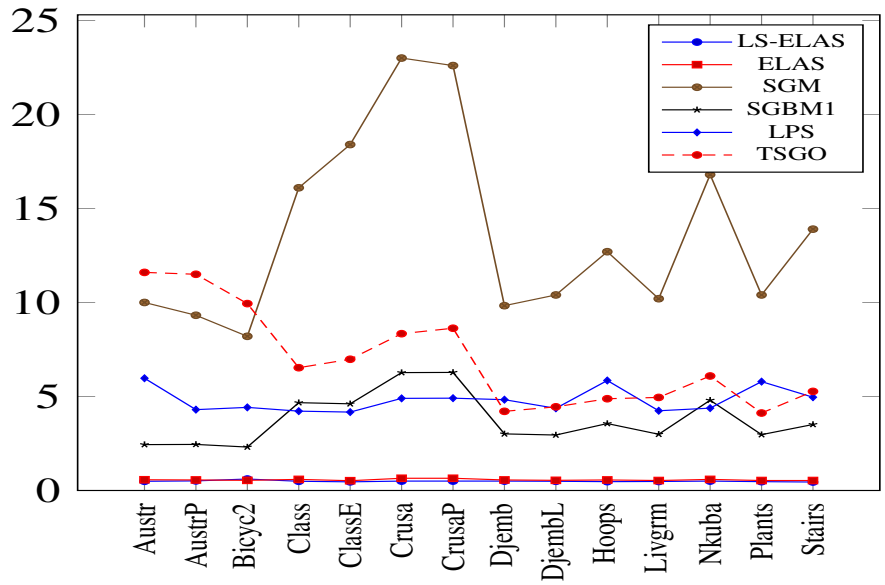


Figure 4.15: Results according to the metric Time/MP. This is a graphical representation of table 4.4. The ranking according to the weighted average results over all the 15 stereo pairs is as follows: LS-ELAS: 0.50, ELAS: 0.56, SNCC (H): 1.02, Cens5 (H): 1.35, SGBM1: 3.69, LPS: 5.28, TSGO: 8.26, SGM: 13.4, NOSS-ROB: 502. LS-ELAS and ELAS almost overlap.

Table 4.4: Performance evaluation on the Middlebury Stereo Benchmark Version 3, TIME / MP, test sparse, Nonocc

Algorithm	Avg	Time per Megapixel														
		Aust	AustP	Bic2	Clas	ClasE	Comp	Crus	CrusP	Djem	DjemL	Hoop	Livgr	Nkub	Plant	Stair
LS-ELAS (F)	**0.50**	**0.49**	**0.51**	**0.61**	**0.49**	**0.46**	**0.48**	**0.50**	**0.50**	**0.50**	**0.49**	**0.47**	**0.48**	**0.50**	**0.47**	**0.45**
ELAS (F)	0.56	0.57	0.56	0.54	0.59	0.52	0.48	0.65	0.65	0.56	0.54	0.56	0.53	0.59	0.53	0.53
SNCC (H)	1.02	0.73	0.69	0.68	1.25	1.23	0.68	1.57	1.62	0.77	0.80	0.95	1.16	1.24	0.77	1.01
Cens5 (H)	1.35	1.03	1.01	0.85	1.83	1.84	0.82	2.21	2.20	1.00	1.02	1.24	1.02	1.64	1.01	1.42
SGBM1 (F)	3.69	2.44	2.45	2.31	4.67	4.61	1.84	6.27	6.28	3.01	2.95	3.56	2.99	4.80	2.97	3.51
LPS (F)	5.28	5.97	4.30	4.42	4.22	4.17	11.4	4.90	4.91	4.83	4.37	5.85	4.24	4.38	5.79	4.96
TSGO (F)	8.26	11.6	11.5	9.94	6.53	6.98	22.4	8.34	8.63	4.21	4.45	4.88	4.95	6.09	4.12	5.27
SGM (F)	13.4	10.0	9.32	8.20	16.1	18.4	8.03	23.0	22.6	9.83	10.4	12.7	10.2	16.8	10.4	13.9
NOSS-ROB (H)	502	502	502	502	502	502	502	502	502	502	502	502	502	502	502	502

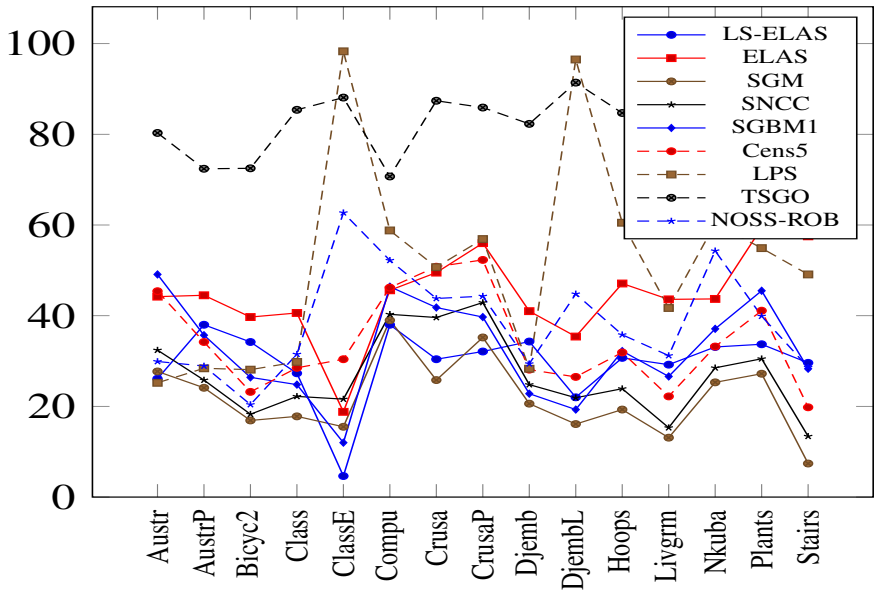


Figure 4.16: Accuracy results according to the error metric "bad 1.0". The ranking according to the weighted average results over all the 15 stereo pairs is as follows: SGM: 8.77, SNCC: 12.7, NOSS-ROB: 13.2, LS-ELAS: 15.8, SGBM1: 16.7, Cens5: 18.1, ELAS: 26.4, LPS: 27.6, TSGO: 63.7.

The metrics: "bad 1.0", "bad 2.0", "bad 4.0"

The standard metric for measuring the performance of a stereo matching algorithm is the percentage of bad pixels. That is the percentage of pixels on the reference image for which the estimated disparity (d_n) exhibit an inaccuracy larger than a given threshold δ_d. For example, the "*bad*1.0" ($E_{bad}(1.0)$) metric has a threshold of 1.0 (pixel) and measure the percentage of pixels for which the error in disparity is larger than 1.0 pixels. It is worth to remind that the Middlebury stereo benchmark provides disparity ground truth (d_n^{gt}) with sub-pixel precision which is not relevant for our algorithm, as we estimate discrete disparities. "*bad*2.0" is the default metric used on the Middlebury stereo benchmark. Eq. 4.6 shows the mathematical formula of the percentage of bad pixels metric.

$$E_{bad}(\delta_d) = \frac{1}{N}\sum_n (|d_n - d_n^{gt}| > \delta_d) \tag{4.6}$$

Fig. 4.16, 4.17 and 4.18 show the results of the selected algorithms according to the metrics "*bad* 1.0", "*bad* 2.0" and "*bad* 4.0", respectively. A significantly lower error value of our method compared to ELAS is clearly visible. LS-ELAS is also more accurate than SGBM1, Cens5, LPS and TSGO. While SGM yields a lower error value, it is a much slower method. The same applies to NOSS-ROB and SNCC. One important re-

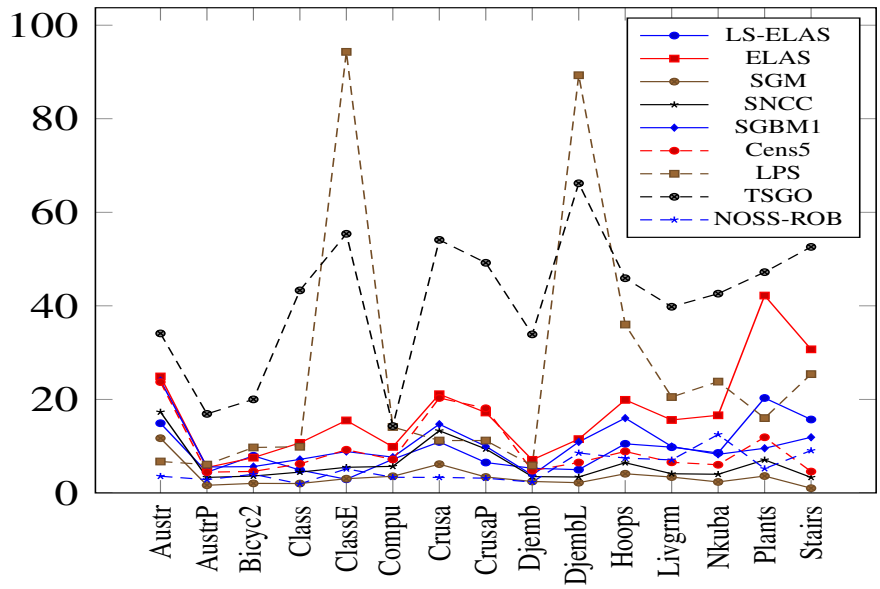


Figure 4.17: Accuracy results according to the error metric "bad 2.0". The ranking according to the weighted average results over all the 15 stereo pairs is as follows: SGM: 3.33, NOSS-ROB: 5.01, SNCC: 6.11, LS-ELAS: 8.82, Cens5: 9.31, SGBM1: 9.44, ELAS: 16.4, LPS: 20.30, TSGO: 39.10.

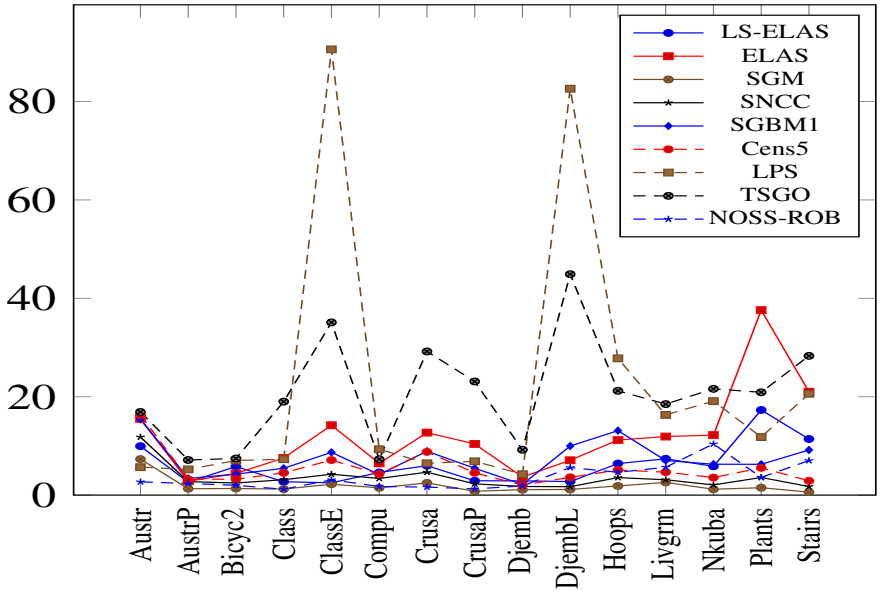


Figure 4.18: Accuracy results according to the error metric "bad 4.0". The ranking according to the weighted average results over all the 15 stereo pairs is as follows: SGM: 1.73, SNCC: 3.26, NOSS-ROB: 3.26, Cens5: 4.95, LS-ELAS: 5.97, SGBM1: 6.52, ELAS: 11.5, LPS: 16.60, TSGO: 18.90.

mark that arises when analyzing the plots is that when increasing the tolerance from one pixel ("*bad* 1.0") to two pixels ("*bad* 2.0"), the error decreases considerably. When the tolerance is further increased to four pixels ("*bad* 4.0"), the error still decreases but not as strongly as in the previous case. The algorithms TSGO and LPS seem to be less accurate even if they are global stereo algorithms. The reason for that is that TSGO and LPS do not filter out occluded pixels. Since we are using the sparse dataset for our evaluation, the occluded pixels contribute to the computation of the error and thus penalize these two algorithms. On the dense dataset from Middlebury, these two algorithms perform much better, and LPS is ranked second (after NOSS-ROB) according to the metric "*bad* 1.0" among the selected algorithms. Nevertheless, LPS and TSGO show strong performance degradation on the challenging stereo pairs ClassE (see Fig. 4.12 page 86), which have strong exposure time differences between the two images of the stereo pair, and DjembL (see Fig. 4.10 page 85), with challenging lighting conditions.
The metric percentage of bad pixels applies a binary classification of the estimate. There are correct estimates for which the error is less than the threshold (δ_d) and incorrect estimates for which the disparity error is larger than the threshold. Once the estimate is out of the tolerance radius, this metric does not quantify how far the estimate is from the correct value. This will severely penalize the algorithms which provide estimates, which are close to the correct values but still out of the tolerance radius. Fortunately, there are other metrics, which address this problem, such as the average disparity error (Avgerr) and root mean squared disparity error (RMS). In the following, we remind the definitions of these metrics (Avgerr and RMS) and we show our results according to them.

Table 4.5: Performance evaluation on the Middlebury Stereo Benchmark - Version 3, BAD 2.0, test sparse, nonocc

Algorithm	Avg	Bad 2.0														
		Aust	AustP	Bic2	Clas	ClasE	Comp	Crus	CrusP	Djem	DjemL	Hoop	Livgr	Nkub	Plant	Stair
SGM (F)	3.33	11.7	1.64	2.04	2.01	3.04	3.56	6.15	3.41	2.45	2.19	4.10	3.39	2.35	3.61	1.01
NOSS-ROB (H)	5.01	3,57	2.84	3.99	1.93	5.15	3.34	3.32	3.15	2.32	8.55	7.45	7.06	12.5	5.20	9.06
SNCC (H)	6.24	17.3	3.32	3.61	4.45	5.48	7.39	13.3	9.40	3.49	3.40	6.46	4.10	3.99	7.07	3.32
LS-ELAS (F)	**8.82**	**14.9**	**4.43**	**7.94**	**4.89**	**2.92**	**7.20**	**10.9**	**6.50**	**5.27**	**4.97**	**10.5**	**9.78**	**8.55**	**20.3**	**15.7**
SGBM1 (F)	9.44	24.0	5.55	5.63	7.17	8.85	7.65	14.7	9.97	3.82	10.9	16.0	9.93	8.25	9.55	11.9
Cens5 (H)	9.48	23.7	4.44	4.60	6.21	9.23	9.21	20.3	18.1	4.77	6.50	8.87	6.55	6.02	11.9	4.57
ELAS (F)	16.4	24.9	5.44	7.55	10.7	15.5	9.85	21.1	17.2	7.07	11.5	19.9	15.6	16.6	42.2	30.7
LPS (F)	20.3	6.72	6.06	9.72	9.87	94.3	14.1	11.2	11.2	5.88	89.3	36.0	20.5	23.8	16.0	25.4
TSGO (F)	39.1	34.1	16.9	20.0	43.3	55.4	14.3	54.1	49.2	33.9	66.2	45.9	39.8	42.6	47.2	52.6

The metric: average disparity error

The average disparity error (Avgerr) is given by Eq. 4.7.

$$E_{Avgerr} = \frac{1}{N}\sum_{n} |d_n - d_n^{gt}| \tag{4.7}$$

Fig. 4.20 shows a comparison of the selected algorithms according to the Avgerr metric. As in the case of the RMS metric, SGBM1 and LPS show a large error on the Classroom2E (ClassE) stereo pair. ELAS and LS-ELAS have a slightly larger Avgerr on the Plants stereo pair. The final Avgerr (averaged over all stereo pairs) from LS-ELAS is 7.36, which is slightly better than 7.75 from ELAS.

The metric: root mean square disparity error

The root mean square disparity error (RMS) evaluates the square root of the mean of the squares of the disparity errors ($d_n - d_n^{gt}$). Eq. 4.8 shows the mathematical formula of this metric.

$$E_{RMS} = (\frac{1}{N}\sum_{n}(d_n - d_n^{gt})^2)^{\frac{1}{2}} \tag{4.8}$$

Fig. 4.19 shows a comparison of the selected algorithms according to the RMS metric. SGBM1 and LPS show a large error on the Classroom2E (ClassE) stereo pair. ELAS and LS-ELAS have a slightly large RMS disparity error on the Plants stereo pair. The metrics Avgerr and RMS have the drawback of being very sensitive to outliers. If the disparity error for an element of the sum in Eq. 4.7 (and Eq. 4.8) is very large, then its contribution on the sum might be undesirably large. The median disparity error metric does not suffer from this drawback. On the following paragraph we will show our results according to the median disparity error metric.

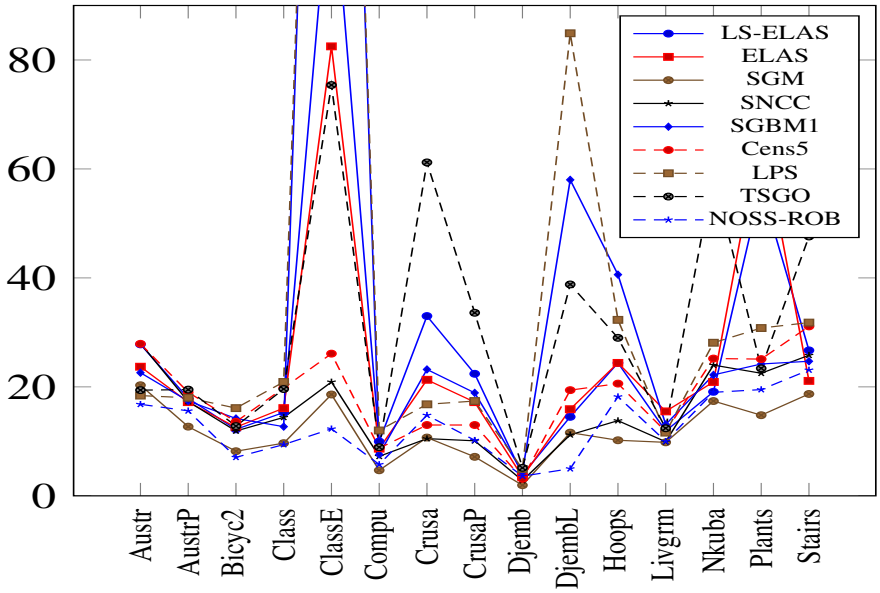


Figure 4.19: Accuracy results according to the error metric "RMS". The ranking according to the weighted average results over all the 15 stereo pairs is as follows: SGM: 10.90, NOSS-ROB: 12.40, SNCC: 14.40, Cens5: 17.2, ELAS: 22.8, LS-ELAS: 24.60, TSGO: 28.80. SGBM1: 29.60, LPS: 31.90.

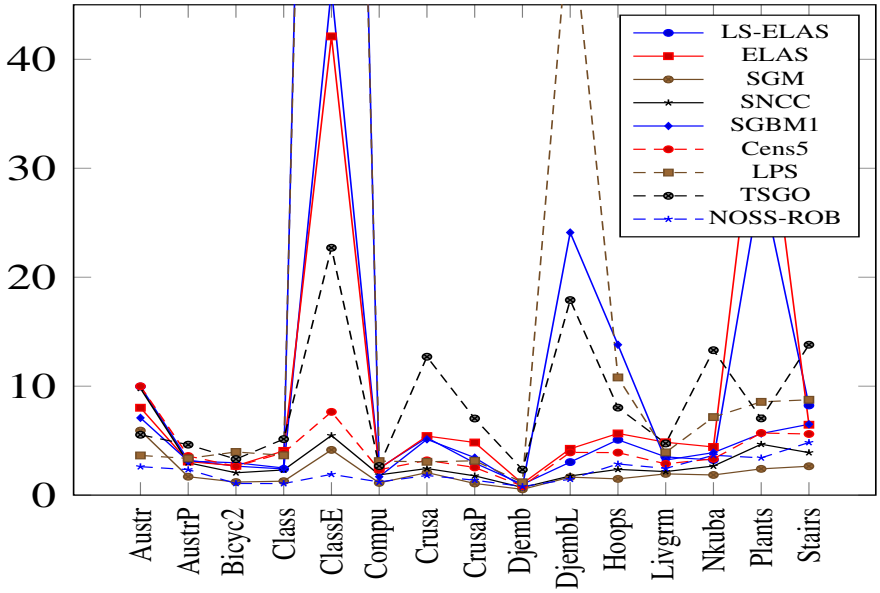


Figure 4.20: Accuracy results according to the error metric "Avgerr". The ranking according to the weighted average results over all the 15 stereo pairs is as follows: SGM: 1.85, NOSS-ROB: 2.08, SNCC: 2.82, Cens5: 3.71, LS-ELAS: 7.36, TSGO: 7.75, ELAS: 8.53, SGBM1: 12.3, LPS: 13.7.

The metric: median disparity error

The median disparity error A50% of the disparity map is the central disparity error of the pixels such that 50% of the disparity errors are less than or equal to A50% and the remaining 50% disparity errors are greater than or equal to A50%. The median is the 50% disparity error quantile in pixels. To compute this metric, we need to sort the disparity errors and select the value at the middle of the sorted list. Fig. 4.21 shows the results according to the median disparity error metric. The averaged A50 error over all stereo pairs from ELAS is 1.55. It is twice as large as the one from LS-ELAS (i.e. 0.74). The algorithms SGBM1 and LPS have significant performance degradation on the challenging stereo pair Classroom2E (see Fig. 4.12). For better visualization of the results of all algorithms, we chose to exclude Classroom2E from the plot on Fig. 4.21. While SGBM1 has a small A50 error (i.e. 0.55) on the stereo pair DjembL (see Fig. 4.10), LPS has a large A50 error (i.e. 30.20) on that stereo pair. For a better visualization, we have also excluded DjembL from the plot on Fig. 4.21.

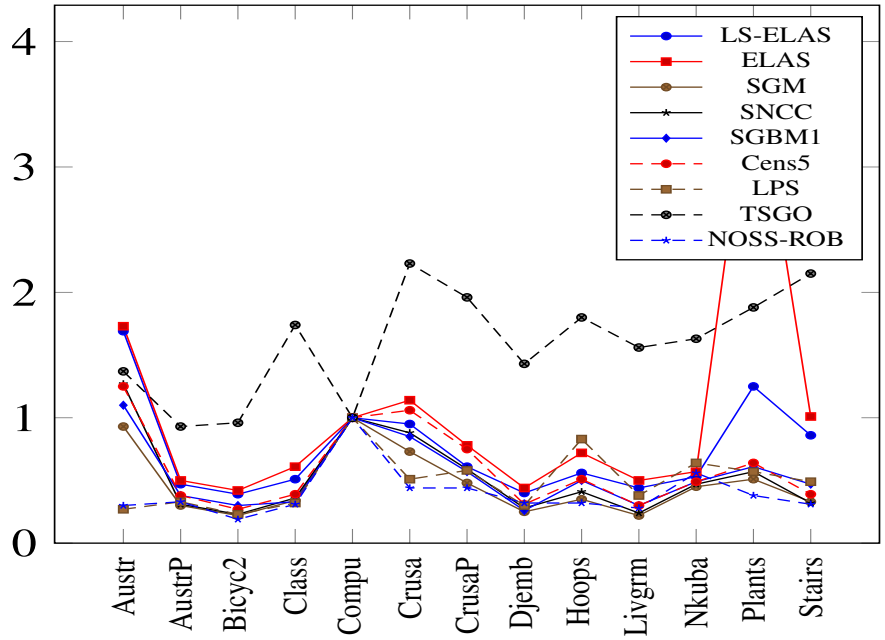


Figure 4.21: Accuracy results according to the median disparity error (A50). The ranking according to the weighted average results over all the 15 stereo pairs is as follows: NOSS-ROB: 0.42, SGM: 0.46, SNCC: 0.52, Cens5: 0.58, LS-ELAS: 0.74, ELAS: 1.55, TSGO: 1.67, SGBM1: 4.84, LPS: 4.99.

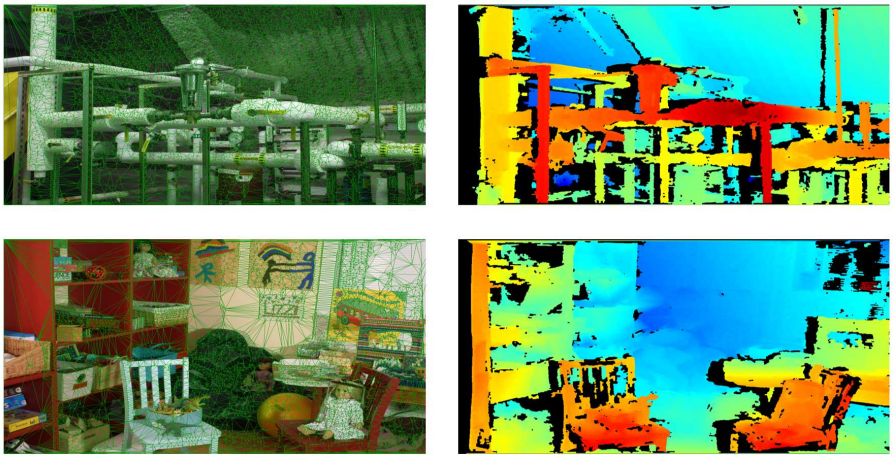


Figure 4.22: Disparity maps estimated by our algorithm. The images are from the Middlebury stereo dataset. Left: the triangle mesh. Right: the final disparity maps.

Figure 4.23: Synthesis of four views by virtually changing the camera viewpoint.

4.5.4 Disparity map examples and view synthesis

In this section we show some examples of the estimated disparity maps using LS-ELAS. Fig. 4.22 shows estimated disparity maps for two stereo pairs from the Middlebury stereo dataset. Fig. 4.23 shows the estimated disparity map for the stereo pair Bicycle. It also shows four views which are synthesized by virtually changing the camera viewpoint.

4.6 Conclusion

4.6.1 Summary

In this chapter, we presented the dense stereo matching algorithm LS-ELAS. LS-ELAS is an extension of the popular ELAS stereo algorithm. LS-ELAS stands for line segments based efficient stereo matching. As its name suggests it relies on line segments. As in ELAS, we first compute a small set ($\sim 5\%$ of the pixels) of support points using a large search interval for candidate disparities. Then we use this set of support points to reduce the disparity search interval for all remaining pixels ($\sim 95\%$ of the points). While in ELAS a grid is used to select the candidate support points, we sample them along the edge segments. The result is a huge reduction in computational time, because LS-ELAS requires significantly fewer support points candidates than ELAS to obtain the same amount of matched support points. While the percentage of matched points exceeds 50% in case of LS-ELAS, it only reaches $\sim 10\%$ in case of ELAS. On average, LS-ELAS requires $281.3\,ms$ for computing the support points on images of size 3000×2000, which is three times faster than ELAS ($838.2\,ms$). Reducing the time for computing the support matches has contributed to efficiency increase of the algorithm overall. The average time needed for processing $1\,MP$ of data ($time/MP$ metric) using LS-ELAS is $0.50s$. ELAS needs $0.56s$ to process the same amount of data.
The selected sparse set of matched support points and the set of line segments relating them are used to construct the prior disparity for our Bayesian stereo matching algorithm. The prior is computed as follows: a 2D triangle mesh is generated from the support points. We used the constrained Delaunay triangulation to generate a triangulation mesh which, is aware of possible depth discontinuities. The third dimension (the disparity) is interpolated for each pixel (inside a given triangle) from the triangle corners (support points with known disparities). The prior disparity for a pixel is modeled as a Gaussian distribution where the mean is the interpolated disparity at the given location inside the triangle (to which the pixel belongs). The likelihood is modeled as a Laplace distribution. A feature vector around the pixel is used to measure the similarity against candidate pixels on the target image. The final disparity is obtained by maximizing the posterior probability. The computation of this posterior is done in the negative log domain.
We validated our approach using the newest version of the public Middlebury stereo Benchmark (version 3, 2014). Many error metrics are implemented to show different

aspects of the compared algorithms. The averaged results over all stereo pairs on the test dataset show that LS-ELAS outperforms ELAS according to all metrics except the RMSE metric. The percentage of bad pixels (wrong matches) of ELAS is almost two times larger than the one from LS-ELAS. For an error tolerance of one pixel ("*bad*1.0" metric), LS-ELAS has 15.80% wrong matches, while ELAS has 26.4%. When increasing the tolerance to two pixels ("*bad*2.0" metric), the percentage of wrong matches decreases for both LS-ELAS and ELAS to 8.82% and 16.4%, respectively. When the tolerance is further increased to four pixels ("*bad*4.0" metric), the percentage of wrong matches decreases for both algorithms but not as strongly as in the previous case. The average disparity error (*Avgerr* metric) from LS-ELAS is 7.36, which is slightly better than 7.75 from ELAS. The averaged median error (*A*50 metric) over all stereo pairs from ELAS is 1.55. It is twice as large as the one from LS-ELAS (i.e. 0.74).
We submitted our results and they are available online on the permanent ranking table of the Middlebury stereo Benchmark. This will allow a broad evaluation of our results.

4.6.2 Discussion: curved or straight line segments

We considered a method for extending ELAS by using straight line segments. Starting by the detection of preferably long straight line segments in both images, match them and use the matches as additional constraints. While this would work on human-made environments, it will sometimes fail to find straight lines on outdoor environments. Furthermore, the use of straight lines is problematic at depth discontinuities. The line might have points on the foreground and points on the background. One more disadvantage is that the straight line segments on target image might be split into multiple parts. So, we preferred to use curved edge segments instead of straight line segments because they work fine for both straight and curved contours. In the image regions with poor texture changes, we still detect small edges there and we sample candidate support points.

Chapter 5

Application: Outdoor Obstacle Avoidance using Stereo for a Quadcopter MAV

In the previous chapter, we discussed dense stereo matching and its application for estimating the depth from disparity. With that method, the robot can estimate the depth map of the visible part of the scene. A depth map can be seen as a snap-shot of the environment geometry at camera pose. Relying on one single shot of the environment geometry is sensitive to inaccuracies of the depth estimation. In particular, distant points suffer from depth inaccuracy. A second drawback of relying on a single depth image is that the estimated depth images might be inconsistent and the overlapping surfaces would not perfectly match. The reasons for potential misalignment of two or more depth images are numerous. This can be due to inaccuracy of the pose estimation and to the inaccuracy of the dense matching estimation. The fusion of multiple depth maps can yield more consistent depth estimation. One efficient way to perform this fusion is to reconstruct a 3D occupancy grid map.

Large parts of this work have been pre-published in Ait Jellal and Zell (2017).

5.1 Introduction

Levitt and Lawton (1990) identified the following three important questions for which an autonomous mobile robot needs to compute answers.

- "Where am I?"
- "Where are other places with respect to me?"
- "How do I get to other places from here?"

We design a system which provides answers to these questions to achieve full autonomous obstacle avoidance in unknown environments. The hybrid SLAM algorithm (see chapter 3) provides an answer to the first question. The hybrid SLAM provides also a context

(the map) on which the pose is estimated. The second question can be partially answered using a volumetric map. We say that this question is only partially solved because the identification of other places would require more than a simple volumetric map. For example, additional semantic segmentation and object classification methods would provide a better answer to the second question. We use LS-ELAS for estimating depth maps which are used for constructing the volumetric map. To answer the third question, we integrated an open-source 3D path planner.

5.2 Motivation

The motivation behind this chapter is twofold. First, we want to experimentally show that our algorithms (in chapter 3 and chapter 4) can be effectively used in real-time on-board our quadcopter. Second, we show that stereo vision can be successfully applied in conditions where other sensors (and systems) would fail. We discussed the benefits of using stereo vision in the introduction of this thesis. A conventional RGB-D camera would fail in those environments when there is substantial sunlight. Furthermore, the existence of high structures, such as building, degrades the GPS accuracy. As a result, the GPS-based navigation might fail. In this chapter, we show experimentally that stereo vision can be used on-board to perform autonomous navigation in unknown environments. The challenge is how to run all of the software components needed for a safe navigation in real-time on the on-board CPU. Keeping the efficiency in mind is a key for developing such a system. During the development of our hybrid SLAM algorithm and our LS-ELAS stereo matching algorithm, we took into consideration the computation time as a major factor. In addition to our efficient algorithms, we needed additional efficient algorithms for building occupancy grid maps and for planning collision-free paths.

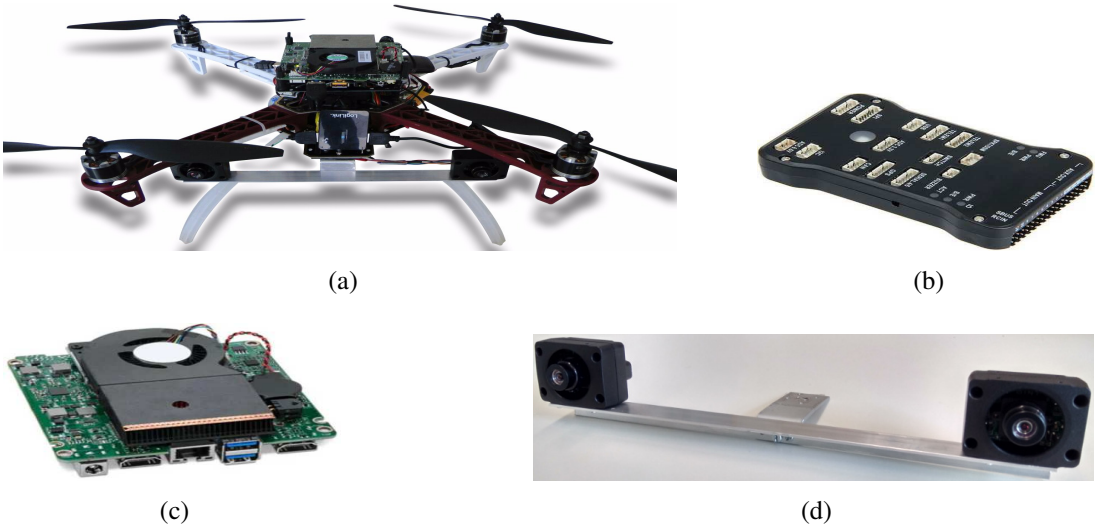
5.3 Related work

Nuske *et al.* (2015) proposed an interesting system for mapping rivers using an octacopter MAV. For sensing the environment, they equipped their MAV with a stereo camera and a lightweight spinning laser scanner. They build and maintain an obstacle map using Scherer *et al.* (2012a), which they use for path planning. They rely on SPARTAN (Sparse Tangential Network) from Cover *et al.* (2013) for planning the paths. They contribute on the exploration and motion planning aspects of the system and use existing algorithms for stereo visual odometry (see Rehder *et al.* (2012)) and for building obstacle map (see Scherer *et al.* (2012a)). In contrast, we create our own SLAM system for pose estimation. Furthermore, we developed an efficient dense stereo matching algorithm for building occupancy grid maps. Heng *et al.* (2011) have proposed a stereo approach for obstacle avoidance. They build an Octomap and use the anytime dynamic A* planner (see Likhachev *et al.* (2005)) to generate collision-free path planning in 2D. For pose

estimation, they use either artificial landmarks or a Vicon tracking system. In contrast, our work is not limited to 2D. We plan paths in full 3D-space using the efficient RRT* (see Şucan *et al.* (2012)) algorithm and we run a hybrid SLAM on-board.

5.4 Experimental platform

In our real experiments, we used a custom-made quadcopter. It is shown in Fig. 5.1. The relevant components of our research quadcopter are as follows: a stereo camera with a pair of Point-Grey Firefly monochrome cameras with a resolution of 640×480 pixels. The baseline between the two cameras is 22 *cm*. The stereo camera delivers synchronized stereo pairs at 30 *Hz*. The stereo camera is calibrated offline to estimate the intrinsic camera parameters using the OpenCV library (see Bradski (2000)). The flight control unit used in this research is the px4-AutoPilot from the open-source Pixhawk project (see Meier *et al.* (2012)). The brushless motors are of type 2212-13 980 *kv* with 10:45 propellers. The on-board computer is an Intel NUC mini-pc. It has an Intel core i7-5557U CPU with 2x3.1 *GHz*, 4 *MB* cache, 28 Watts thermal design power (TDP) and 8 *GB* RAM. The on-board computer is connected to the autopilot via a serial link of type USB to UART (Universal Asynchronous Receiver Transmitter). The quadcopter is powered using a 4 cells Lipo battery with 5000 *mAh* capacity. The quadcopter length is 450 *mm* and it has a total weight of approximately 1.62 *kg*.



(a) (b)

(c) (d)

Figure 5.1: (a) the custom made quadcopter platform used in this research. (b) the Pixhawk px4-AutoPilot with IO-board. (c) the on-board mini-pc Intel NUC. (d) the stereo camera.

5.5 System description

5.5.1 System overview

Fig. 5.2 shows an overview of our system for outdoor obstacle avoidance. The system components include:

- The hybrid stereo SLAM algorithm (see chapter 3, page 39).
- The LS-ELAS dense stereo matching algorithm (see chapter 4, page 69).
- The robust OctoMap for creating volumetric 3D maps (see Schauwecker and Zell (2014)).
- The RRT* path planner for planning collision-free paths on the OctoMap (see J. *et al.* (2011) and (LaValle, 1998)).
- The position and attitude controllers from the Pixhawk project (see Meier *et al.* (2012)).
- The hardware of the experimental platform (see page 107).

The hybrid stereo SLAM algorithm is in charge of keeping track of the quadcopter pose in real-time while at the same time building a sparse map of the environment. At keyframes, we compute dense stereo matching. The resulting disparity map is then used to create a 3D point cloud, which is inserted in the occupancy grid map. The disparity map is also used by the tracking thread to refine the poses using direct image alignment. The planner gets an up to date binary occupancy grid map and the start (current) pose and destination pose and then plans a safe path for obstacle avoidance. The planner outputs a smooth trajectory of way-points. The poses from the localization thread of the SLAM system are fused with the measurements from the on-board IMU using an extended Kalman filter (EKF) to estimate the quadcopter state. The controller gets the desired pose and the current pose and controls the motor speeds to fly to the desired position. The Pixhawk px4-AutoPilot (see Meier *et al.* (2012)) controller is a cascaded controller which includes a high level position controller and a low level attitude controller. The attitude controller is critical and it outputs the motors speeds at high control loop frequencies.
Our system is implemented in a set of software packages using the robot operating system (ROS) middleware. As a result, we gain modularity and clear interfaces between the software packages.

5.5.2 Environment mapping

The map built by the Hybrid SLAM algorithm is too sparse to be used for path planning. Traditionally, 2D maps are widely used in mobile robotics. This can be granted to their

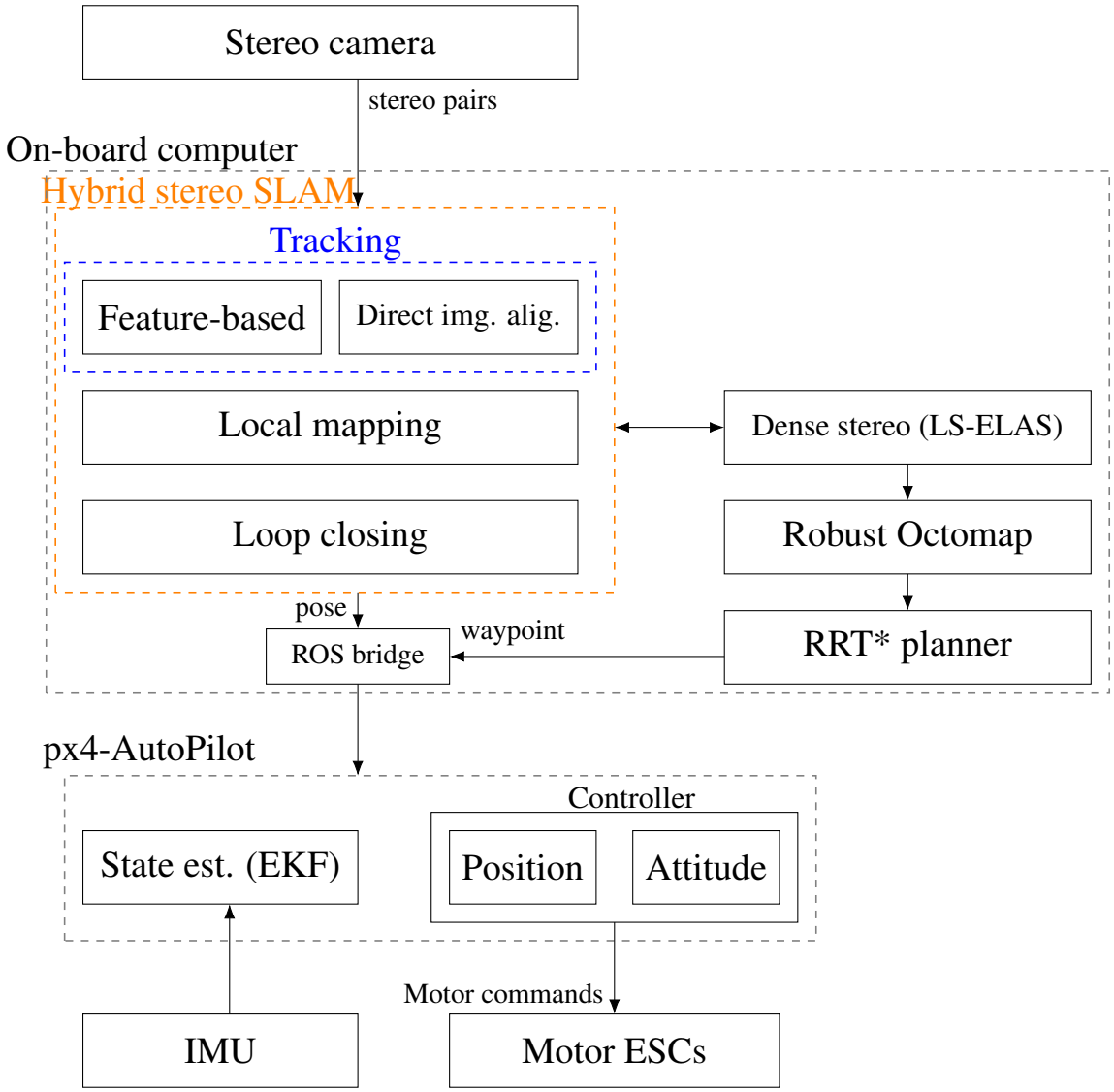
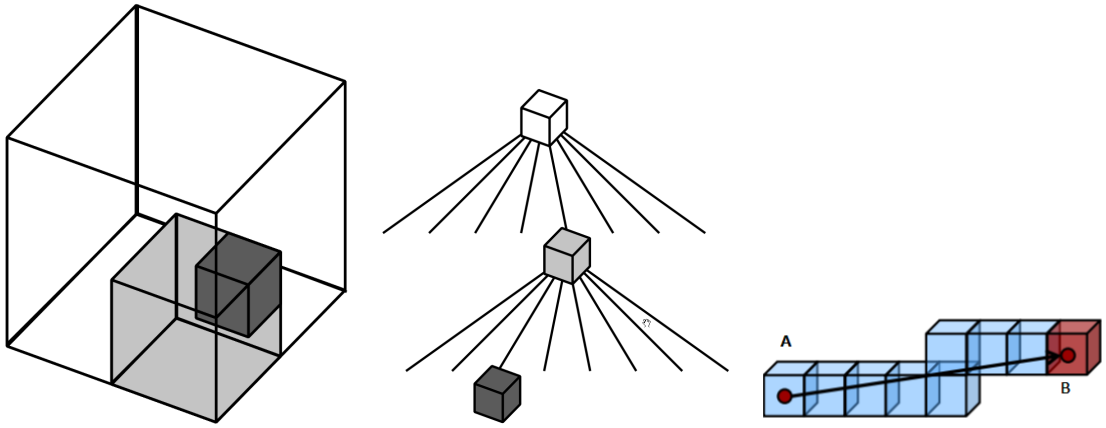


Figure 5.2: System overview. The system includes a hybrid SLAM algorithm and a path planner. The tracking thread of the SLAM algorithm provides an estimate of the current pose to the controller. The path planner provides intermediate destinations to the controller. The Pixhawk position controller manages to fly smoothly through the intermediate way-points.

Figure 5.3: An outdoor experiment.

small memory footprint and reduced map update time. While 2D maps are suitable for many ground mobile robots, they are not suitable for mobile robots which navigate freely in 3D space such as multirotor MAVs. Even for ground robots, 2D maps are sometimes not sufficient. A tall ground robot which needs to navigate below overhanging structures requires also 3D maps. The 3D maps can be, however, memory demanding when naively implemented. Fortunately, the octree data structure (see Meagher (1980)) gives a solution to the memory footprint issue for 3D mapping. An octree represents the 3D space as tree of cubes. At the root of the tree data structure, there is a large cube, which represents the whole modeled 3D space. For storing a 3D point on the octree, we need to recursively subdivide the octree into smaller and smaller subcubes. At each subdivision's iteration, a parent cube is divided into 8 children. Fig. 5.4 illustrates the subdivision process of the octree data structure. Only the child subcube which contains the 3D point is then subdivided in the next iteration. The remaining 7 subcubes do not need to be subdivided if they do not contain other 3D points. This results in huge memory efficiency as memory is not allocated for empty subcubes.

Wurm *et al.* (2010) developed OctoMap, which is an open-source library for 3D mapping. They rely on the octree data structure to efficiently represent the 3D space. Occupied, free and unknown spaces are all modeled to accommodate for different applications. For example, the identification of unknown space is very important for exploration tasks and the free space is used for planning collision-free paths. A probabilistic representation of occupancy is used in OctoMap to handle sensor noise. The binary Bayes filter from Moravec and Elfes (1985) is used for recursively updating the map as new measurements are made. At time t, the fused occupancy probability $P(n|z_{1:t})$ of a voxel n is estimated by updating its occupancy probability $P(n|z_{1:t-1})$ at time $t-1$ using the new measurement (at time t). The map update is given in Eq. 5.1 (see Moravec and Elfes



(a) (b) (c)

Figure 5.4: (a) recursive subdvision process of the 3D space. A parent node (cube) can be subdivided into 8 children. (b) modelling of the subdivision process using the tree-based octree data structure. Memory is not allocated for empty nodes. (c) illustration of the ray casting process for identifying the voxels to be updated. A ray is fired from the sensor origin (A) toward the 3D endpoint (B). The figures are reprinted from Schauwecker (2014)(p. 103).

(1985)).

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

For efficiency, the map update step is done using log-odds.
In the case of stereo vision, the 3D endpoints are estimated from the disparity map (see page 31). Given the sensor location and the 3D endpoints, OctoMap uses ray-casting (see Amanatides and Woo (1987)) to identify the voxels (octree leaf subcubes) that need to be updated. The leaf voxel on which the 3D point is located is considered as occupied and also determines the visibility limit along the casted ray. The voxels between the sensor origin and the occupied voxel are considered empty. Fig. 5.4(c) illustrates the process of identifying the voxels to be updated. The map update rule (Eq. 5.1) is then applied to all voxels along the casted ray up the the visibility limit. In order to ensure the updatability of the occupancy probability of the voxels, a clamping policy is introduced. An upper and a lower bounds are set for the probability of the voxels.
OctoMap has been used in many robotic works, and extensions to the original OctoMap have been proposed. Among these extensions, an open-source variant, which is well suited for stereo vision has been proposed by Schauwecker and Zell (2014). As we are using a stereo camera, we use this variant of OctoMap to estimate a 3D occupancy grid map of the robot environment.
Fig. 5.5 shows some intermediate steps for mapping the robot environment using the robust OctoMap. The data is recorded during one of our outdoor experiments. During

the navigation, the hybrid SLAM algorithm keeps track of the robot pose and decides to create new keyframes (KF). At keyframes, we estimate the depth maps using LS-ELAS. The robust OctoMap uses the poses of the keyframes and the estimated depth images as input to create a volumetric map of the environment. On the next section, this volumetric map will be used for planning collision-free paths.

5.5.3 3D path planning using RRT*

The 3D path planner gets a binary version of the up-to-date octree and updates its bounds using the current octree bounding boxes. The planning in 3D is more challenging than in 2D and efficient methods need to be used. Standard path planners such as A^* are not efficient for use on-board a quadcopter. We choose to use a variant of the efficient rapidly-exploring random trees RRT (see LaValle (1998)). We use the open-source library OMPL, which implements a set of planning algorithms. One of the algorithms used for path planning in this setup is the RRT* algorithm from Karaman and Frazzoli (2011). It is a variant of the original RRT. In the following the choice of this algorithm is illustrated. RRT is a sampling-based algorithm and provides *probabilistic completeness* (see Karaman and Frazzoli (2011)). That means that the probability that the planner fails

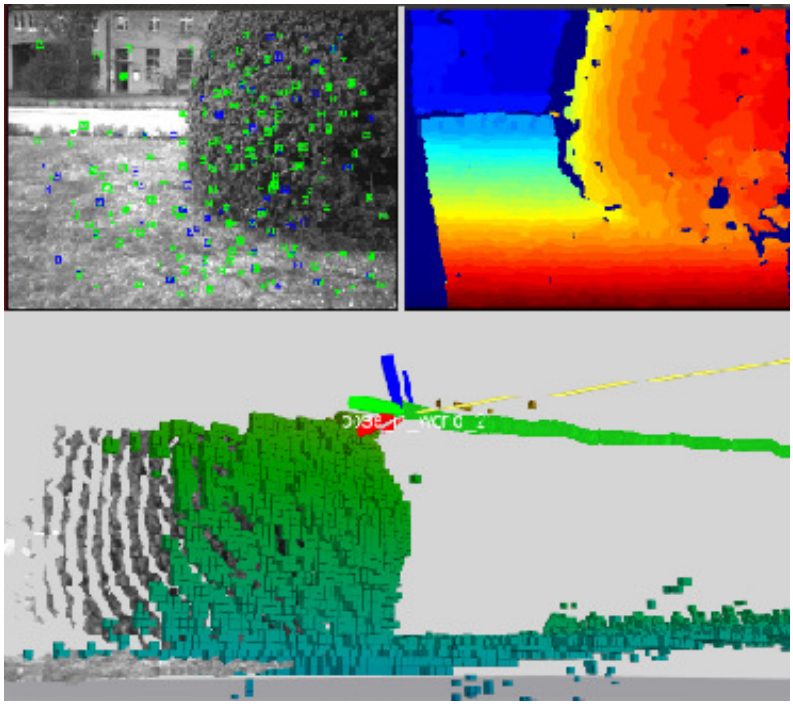
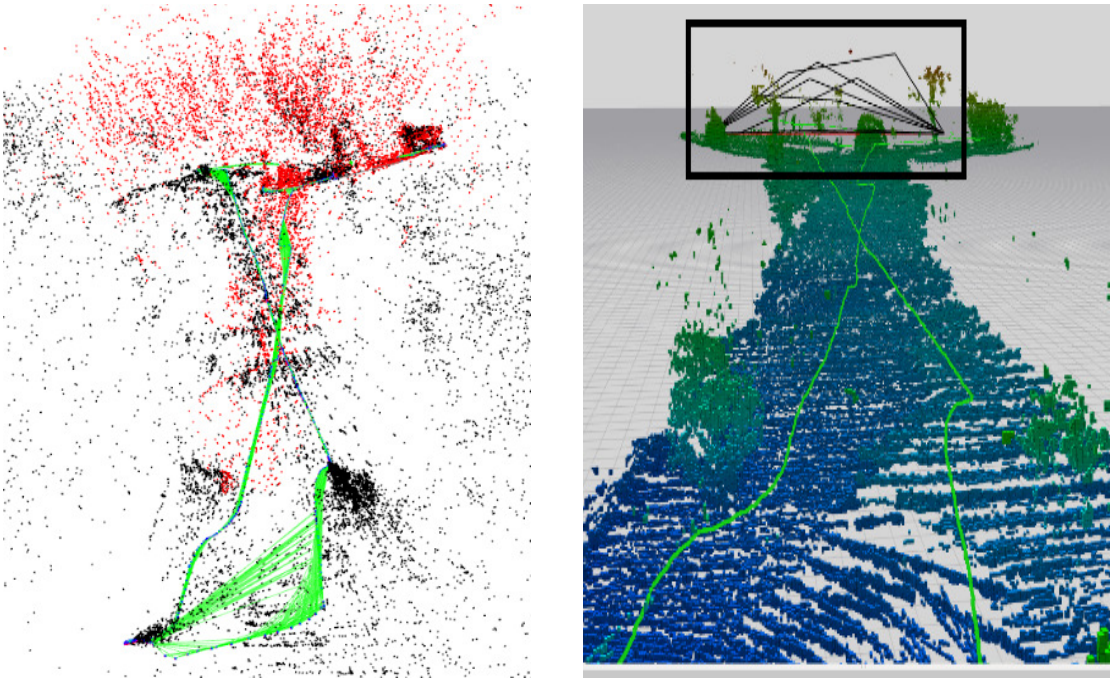


Figure 5.5: This figure shows the features tracked on the current frame, the disparity map of the last keyframe (the color encodes the disparity value) and a view of the volumetric reconstruction. The current point cloud which overlaps with the map is also drawn.

to find a solution, if there is one, goes to zero as the number of samples approaches infinity. Such a sampling-based planner requires a collision checking module as it does not represent obstacles explicitly (see Karaman and Frazzoli (2011)). A problem of RRT is that it doesn't provide any guarantees to an optimal solution. If we use RRT in this setup we get a valid solution very fast but as stated it is not necessarily optimal. Experiments show that it is very often not even close to optimal. But as we are planning paths for a quadcopter optimal paths (or at least short paths) are important to not waste battery power. To achieve an optimization of the planned path RRT* is used. Basically, the tree is constructed in the same manner as in normal RRT but not all feasible connections will be inserted. Normal RRT just inserts a connection between the new node and its nearest neighbor (considering the Euclidean distance). The RRT* algorithm will check all nodes in the surrounding of a new node and only insert the shortest path to the new node, considering a cost function, into the tree. This cost function differs from the Euclidean distance, because on the path from the root to the node, which will be connected to the new node, there might be some obstacle that increases the cost in comparison to a straight line connection. Therefore the nearest neighbor of the new node does not have to be on the shortest path to the new node. After that, all the surrounding nodes are checked again whether there is a new shortest path to each of them using the newly inserted node. If that is the case, then the tree gets rewired to maintain it a tree structure. As stated in Karaman and Frazzoli (2011), RRT* is *probabilistically complete*, like the normal RRT, but moreover it is *asymptotically optimal*. That means that the returned solution converges almost surely to the optimal path (see Karaman and Frazzoli (2011)). This property made the RRT* algorithm a good choice for this path planning scenario. Moreover, the algorithm is not limited to finding geometric shortest paths but can also optimize towards a mixed optimization objective taking different costs (e.g. avoiding power extensive maneuvers) into account. In the quadcopter online planning scenario it has to be considered that one has to make a trade-off between spending energy and time flying a non-optimal path vs. spending energy and time hovering too long to compute the optimal path plus flying this path. Once a path is planned the way-points are sent to the Pixhawk controller via a serial connection. The Pixhawk controller takes care of flying the quadcopter to the desired destination.

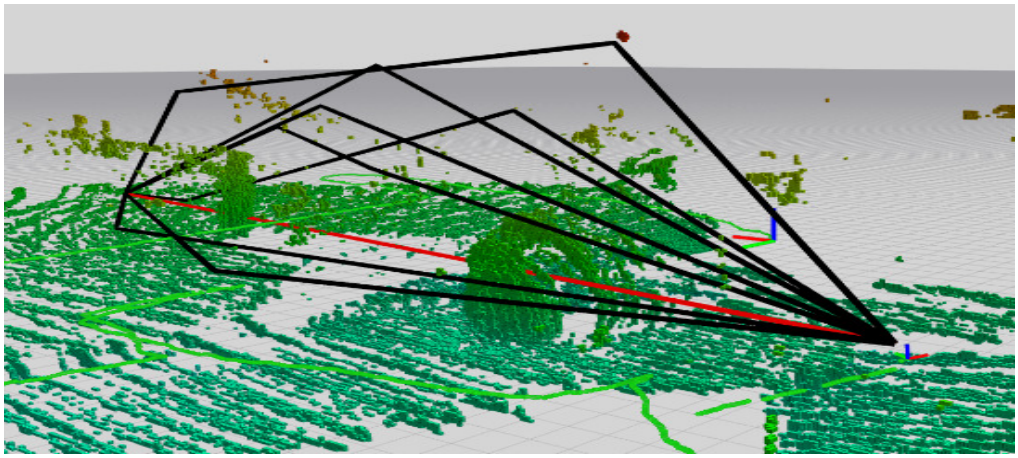
(a) (b)

Figure 5.6: Thanks to the volumetric global map maintained by our algorithm, we can plan collision-free paths between any two positions on the global map. (a) a view of the outdoor environment in which we did the experiment. (b) a Google Maps view of this area. The yellow rectangle shows the mapped area and the red line inside it shows the straight line between the start and goal position for the planner.

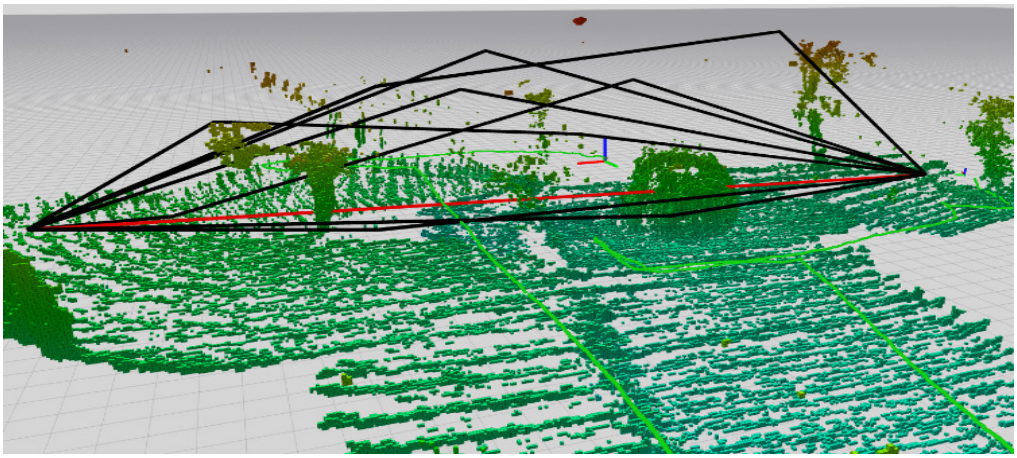


(a) (b)

Figure 5.7: (a) the sparse map built by the SLAM algorithm. This map is used for pose estimation and re-localization. (b) the 3D volumetric map (Octomap) which is used by the planner. The green dots show the robot trajectory.



(a)



(b)

(c)

Figure 5.8: (a) and (b) show two views of a set of 3D paths between the start and destination positions. These views correspond to the black rectangle region of the map in (b).

5.6 Results from outdoor experiments

We performed outdoor experiments in an environment with vegetation (grass), trees, walls and asphalt. Some pictures of this environment can be seen in Fig. 5.3, Fig. 5.8(c), Fig. 5.6(a) and Fig. 5.6(b). We uploaded a video to Youtube to show some of our results. Link: https://youtu.be/h3yBGVeW8zQ.
Fig. 5.8 shows the details of an outdoor experiment. It shows the sparse map and the volumetric map. A set of collision-free paths between two points are also shown. Table 5.1 shows some statistics from the outdoor experiment. The running times are reported for the case where all software packages of our system (see Fig. 5.2) are running at the same time on-board the quadcopter. The average time for the tracking thread of the hybrid SLAM system is $72ms$, which corresponds to a robot pose update at $14fps$. As we are not performing any aggressive flight maneuvers, the pose update meets the real-time requirements. We set two configurations for performing the dense stereo matching. In the first configuration, we use the full image resolution (640×480) and we set the disparity search range to 96 disparities. On average, the running-time of the dense stereo matching at this resolution is $183ms$. In the second option, we down-scale the images and perform the dense stereo matching at half-resolution (320×240) and we set the disparity search range to 48 disparity candidates. The second configuration is much faster ($55ms$) while still providing depth maps which are accurate enough to be used for path planning. The update time of the OctoMap depends on the resolution of the estimated depth maps. The average OctoMap update time is $191ms$ for the full depth map resolution (640×480) and $62ms$ for the half resolution. The octree resolution (voxel size) is set to $10cm$. The ray casting is set to identify the voxels which need to be updated up to a distance of $8m$ along the ray. The planning time is set to $10s$. The planner outputs a set of smooth intermediate way-points for stable control of the autonomous flight. The paths include sufficient intermediate way-points between the start and destination positions. In the examples shown in the Fig. 5.8, the RRT* planner made on average 86 intermediate way-points between the start and destination positions. In the current system, the destination position for the planner is set manually using a script. If the goal position is outside of the bounding box then the planner sets the destination as the farthest position, along the straight line between the start position and destination position, which is inside the mapped area (bounding box).

5.7 Conclusion

This chapter presents a system for outdoor obstacle avoidance using a stereo camera. We try to answer the basic three questions for autonomous navigation: (1) “where am I?”, (2) “where are other places with respect to me?” and (3) “how do I get to other places from here?”. We used our algorithms described in chapter 3 (Hybrid SLAM) and chapter 4 (LS-ELAS) to build a system for outdoor obstacle avoidance using stereo

Sequence length (number of stereo pairs)	9040
Number of keyframes	1128
Number of map points	30616
Time for the feature based tracking	52 ms
Time for direct alignment refinement	21 ms
Run-time for dense stereo matching [SGM]	55 ms (320 $\times$ 240 pixels); 183 ms (640 $\times$ 480 pixels)
Octree update time	62 ms (320 $\times$ 240 pixels); 191 ms (640 $\times$ 480 pixels)
Size of the octree	360 MB
Size of the binary Octree	1.1 MB
Time for planning using RRT*	10 s
Average number of intermediate way-points	86 way-points along the path

Table 5.1: Some statistics from the outdoor obstacle avoidance experiment. The different software packages can run in real time on-board the quadcopter. The localization of the robot, which is very critical, meets the real-time performance and runs at $14 fps$.

vision. This system performs on-board the quadcopter and in real-time the following tasks: localization and mapping using our hybrid stereo SLAM, disparity estimation at keyframes using our LS-ELAS algorithm, volumetric occupancy grid maps estimation using OctoMap and 3D trajectory planning using RRT*.
During navigation, the tracking thread of the hybrid stereo SLAM algorithm provides the robot pose at 14 fps. As we are not performing any aggressive flight maneuvers, the update rate of the robot pose is sufficient to meet real-time requirements. The tracking thread decides when to create new keyframes. We rely on the keyframes to create the occupancy grid map. First, we run our LS-ELAS dense stereo matching algorithm. The produced disparity maps are converted to depth maps and they are fused to construct the 3D occupancy grid map of the robot environment. The robust OctoMap is memory efficient and could map a large environment (sequence of 9040 stereo pairs) using only $360 MB$ of memory. Furthermore, if a binarization of the OctoMap is performed, the size reduces to $1.1 MB$. This map is global and it allows the robot to keep track of the explored parts of the scene which are no longer visible. For simplicity, the robot environment is assumed to be static. We then integrated a module for path planning in 3D. For simplicity, we manually set the starting and destination points and the 3D planner generates collision-free paths. The 3D planner is very efficient as it is based on the efficient RRT* algorithm.

Chapter 6

Conclusions

6.1 Summary

In this dissertation, we have investigated the use of stereo cameras to enable full autonomy of a MAV. The MAV that we have chosen was a quadcopter with the advantages of having a high degree of freedom and the possibility to hover at a fixed position. While having the afore-mentioned advantages, the quadcopter suffers from limited payload and limited endurance time. The need for efficient algorithms was key for successful fully autonomous flights.

We developed an efficient hybrid SLAM algorithm (chapter 3), which combines feature-based SLAM with direct image alignment. We addressed the following important question: is it better to estimate the pose from a set of pre-computed feature correspondences or to estimate the pose and the feature correspondences simultaneously? The hybrid SLAM combines both methods. Firstly, it performs pose estimation using features. The hybrid SLAM extends the ORB-SLAM2 algorithm (see Mur-Artal and Tardós (2016)). Large camera movements can be efficiently and effectively handled using the feature-based motion estimated. This is due to the availability of repeatable feature detectors and invariant feature descriptors (invariant to geometric and photometric changes). Secondly, the hybrid SLAM refines the estimated pose using direct image alignment. In this step of the algorithm, the hybrid SLAM takes into consideration the details that have been ignored on the first step to simultaneously estimate the pose and the pixel correspondences. We have evaluated our algorithm using the KITTI odometry benchmark. In trajectories with no loops and no dynamic objects, our algorithm has shown significant improvement of the accuracy of the trajectory estimate compared to the original ORB-SLAM2. For trajectories with loops, no significant improvement was achieved by our refinement step. This behavior has been seen on trajectories with a single loop and it becomes even more obvious when there are multiple loops on the trajectory. The refinement step requires about 25% of the tracking time.

In chapter 4, we have introduced LS-ELAS which is an efficient dense binocular stereo matching algorithm. LS-ELAS is used for estimating the depth from disparity. The depth is used on our autonomous quadcopter for two purposes. Firstly, on the hybrid stereo

SLAM for warping the (left) intensity images (see chapter 3). Secondly, for constructing a volumetric occupancy grid map (see chapter 5). LS-ELAS is based on line segments and it extends the popular ELAS algorithm (see Geiger *et al.* (2011a)). Similar to ELAS, LS-ELAS computes the disparities in near-constant-time. For a small set of pixels (the support points), the search for matches is done in linear-time with respect to the disparity search window. While ELAS samples the candidate support points using a uniform grid, LS-ELAS uses edge segments to sample the support point candidates. This results in efficient computation of the support points. LS-ELAS has a larger ratio of "successfully matched pixels over all candidates" than ELAS. Moreover, the support points from LS-ELAS allow for depth discontinuity awareness, as they are sampled along edges. A 2D triangle mesh is then constructed using the set of support points and the set of line segments. As the disparity is computed for the triangles corners (support points), one may compute a mean disparity for all pixels (inside the triangles) by means of interpolation. A prior based on the mean disparity can be set for all the remaining pixels on the image. Thus, the matching of the remaining pixels can be achieved in constant-time by searching for matches in a fixed interval of candidates disparities around the mean disparity. This results in a near-constant-time algorithm for stereo matching. We validated our approach using the newest version of the Middlebury stereo benchmark (version 3, 2014). Many error metrics are implemented to show different aspects of the compared algorithms. The metrics include the percentage of bad pixels, the average disparity error, the root mean square disparity error (RMSE), the median error, and the time/MP metric. The averaged results over all stereo pairs on the test dataset show that LS-ELAS outperforms ELAS according to all metrics except the RMSE metric. ELAS slightly outperforms LS-ELAS according to RMSE.

Finally (in chapter 5), we used these algorithms to build a system for outdoor obstacle avoidance in unknown environments. The goal of this system was to show that our approaches can work in real-time on-board the experimental platform. The challenge was to run all the required algorithms on-board the quadcopter in real-time at sufficient frame rates.

6.2 Future work

The research presented in this thesis has raised many questions. The approaches that have been proposed still have limitations.
The combination of feature-based SLAM with direct image alignment has increased the accuracy on trajectories with no loops. In trajectories with loops, the original ORB-SLAM2 (based only on features) yields almost the same results as our approach. The reason for that is the powerful SLAM back-end from ORB-SLAM2, which is designed to work with features. For efficiency reasons, we used the SLAM back-end from ORB-SLAM2. There is need for a better SLAM back-end for our approach. An option would be to design a SLAM back-end based on photometric bundle adjustment (see Alismail

et al. (2016b)) to jointly optimize the camera trajectory and scene geometry using direct image alignment (without explicit extraction of the features). Unfortunately, photometric bundle adjustment is computationally expensive and thus not suitable for real-time application on our hardware. A practical option to improve the accuracy of our method (including loopy trajectories) would be to mount other sensors such as LIDAR and fuse the visual SLAM estimates with the LIDAR estimates. At the time of writing this dissertation (december 2018), the algorithm V-LOAM (see Zhang and Singh (2015)), which combines visual SLAM with LIDAR, provides the best accuracy results on the KITTI odometry benchmark.

The LS-ELAS algorithm presented in chapter 4 has shown promising results on the sparse dataset of the Middlebury stereo benchmark. However, the results on the dense dataset are not as good as the results on the sparse dataset. The reason for that is that LS-ELAS does not implement a suitable post-processing step for correcting invalid pixels (hole filling). Rather than that, only a simple extrapolation is performed to correct the invalid pixels. The design of a suitable post-processing method would have increased the overall performance of LS-ELAS. Additionally, the explicit identification of discontinuity edges can potentially increase the accuracy of LS-ELAS.

The system presented in chapter 5 addressed the following three important questions for autonomous navigation (see Levitt and Lawton (1990)): (1) "Where am I?", (2) "Where are other places with respect to me?", and (3) "How do I get to other places from here?". The second question can be partially answered using a volumetric map. We say that this question is only partially solved because the identification of other places would require more than a simple estimation of the environment geometry using grid maps. For example, additional semantic segmentation and object detection and recognition methods would provide a better answer to the second question. Furthermore, the destination pose is set manually by the pilot. Thus, there is a need for integrating a module for choosing where to go (see Shade (2011)).

Symbols

B	Baseline of the stereo camera.
C	Camera center of projection.
$\boldsymbol{C} = \{c_{ij}\}$	set of candidate support points sampled along the edges. c_{ij} Candidate support point i sampled on edge segment e_j.
$\mathcal{C}$	Camera coordinate system.
d_n	Disparity of the pixel n.
e	Error function (residual).
$\mathcal{E} = \{e_j\}$	set of edge segments.
f_n	16-dimensional feature vector around the pixel n.
f_x, f_y, c_x, c_y	Camera intrinsic parameters.
H	Hessian matrix.
$\{I^l_{cur}, I^r_{cur}\}$	and $\{I^l_{ref}, I^r_{ref}\}$ the left and right images of the current frame and reference keyframe, respectively.
$\mathcal{I}$	Image coordinate system.
J	Jacobian matrix.
$\boldsymbol{L} = \{l_{i_1 j - i_2 j}\}$	set of straight line segments between two consecutive support points $s_{i_1 j}$ and $s_{i_2 j}$ which belong to the same edge e_j.
N_s	$N_s = \bigcup_{i=1,2,3} \{d_{pi}-1, d_{pi}, d_{pi}+1\}$. Where $p1$,$p2$ and $p3$ are the vertices of the triangle to which the pixel belongs. d_{pi} are the disparities of the triangle vertices.
$o_n = (u_n, v_n, f_n)^T$	Observation with coordinate (u_n, v_n) and feature vector f_n.
$O = \{o_1, ..., o_N\}$	Set of all image observations. The observations on the left image (reference image) are denoted by $o_n^{(l)}$ and $o_n^{(r)}$ for the right image.
$p = (x, y)$	Pixel with the coordinates (x, y).
$P = (X, Y, Z)$	Point in 3D space corresponding to the image pixel $x = (u, v)$.
P_{proj}	Projection matrix.
π, π^{-1}	Pinhole camera projection model and its inverse.
K	Camera calibration matrix.
R	3×3 rotation matrix.
s_m	Support point defined by the coordinates (u_m, v_m) and the disparity d_m. $s_m = (u_m, v_m, d_m)^T$.
$\boldsymbol{S} = \{s_1, ..., s_M\}$	set of the support points . These are elements of $\boldsymbol{C}$ which are successfully matched on the target image.

t	3D translation (vector).
$T \in SE(3)$	3D rigid body transform. $T = [R\|t]$ where $R \in SO(3)$ and $t \in \mathbb{R}^3$.
T_f	Transform estimated by the feature-based approach.
T_d	Transform estimated by the direct image alignment approach using the inverse compositional algorithm.
σ, γ, β	Parameters.
$\mu(\boldsymbol{S}, \boldsymbol{L}, o_n^{(l)})$	Mean disparity obtained it by interpolating the disparities of the triangle corners.
$W(p, \xi)$	Warp function with parameter ξ. It maps pixel a p to a pixel $W(p, \xi)$.
$\mathcal{W}$	World coordinate system.
∇I	Image intensity gradients (Jacobians).
$\check{x}$	Initial guess for the parameter x used to initialize the optimization.
x^*	Optimal value of the parameter x (after the optimization).
$\xi \in se(3)$	Minimal parametrization of the 3D rigid body transform in the Lie algebra.

Abbreviations

BA	Bundle adjustment
BP	Belief propagation
BRIEF	Binary robust independent elementary features
deg	Degree (°)
Det	Determinant of a matrix
DoF	Degrees of freedom
EKF	Extended Kalman filter
ELAS	Efficient large-scale stereo
Eq.	Equation
ESC	Electronic speed control
Fig.	Figure
fps	Frames per second
g2o	Generalized graph optimization
GPS	Global positioning system
GPU	Graphics processing unit
KF	Keyframe
KITTI	Karlsruhe institute of technology and Toyota technological institute at Chicago
IMU	Inertial measurement mnit
LS-ELAS	Line segment-based efficient large-scale stereo
MAV	Micro aerial vehicle
MB	MegaByte
MP	MegaPixel
ms	Milliseconds
ORB	Oriented FAST and rotated BRIEF
PnP	Perspective n-points
PTAM	Parallel tracking and mapping
RANSAC	Random sample consensus
RGB-D	Red Green Blue Depth
RMS	Root mean square
RMSE	Root mean square error
ROS	Robot operating system
RRT	Rapidly-exploring random tree
SAD	Sum of absolute differences

s	Seconds
SGM	Semi-global matching
SIFT	Scale invariant feature transform
SFM	Structure from motion
SLAM	Simultaneous localization and mapping

Bibliography

Agarwal, S., Snavely, N., Simon, I., Sietz, S. M., and Szeliski, R. (2009). Building rome in a day. *12th. IEEE International Conference on Computer Vision (ICCV)*, pages 72–79.

Ait Jellal, R. and Zell, A. (2015). A fast dense stereo matching algorithm with an application to 3D occupancy mapping using quadrocopters. *17th IEEE International Conference on Advanced Robotics (ICAR)*, pages 587–592.

Ait Jellal, R. and Zell, A. (2017). Outdoor obstacle avoidance based on hybrid visual stereo SLAM for an autonomous quadrotor MAV. *European Conference on Mobile Robots (ECMR)*, pages 1–8. `http://www.cogsys.cs.uni-tuebingen.de/publikationen/2017/JellalECMR17.pdf`, Youtube video: `https://www.youtube.com/watch?v=h3yBGVeW8zQ`.

Ait Jellal, R., Lange, M., Wassermann, B., Schilling, A., and Zell, A. (2017). LS-ELAS: line segment based efficient large scale stereo matching. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 146–152. `http://www.cogsys.cs.uni-tuebingen.de/publikationen/2017/JellalICRA17.pdf`.

Alismail, H., Browning, B., and Lucey, S. (2016a). Bit-planes: Dense subpixel alignment of binary descriptors. *Computing Research Repository (CoRR) in arXiv*. September 2016, volume abs/1602.00307.

Alismail, H., Browning, B., and Lucey, S. (2016b). Photometric bundle adjustment for vision-based SLAM. *Computing Research Repository (CoRR) in arXiv*. August 2016, volume abs/1608.02026.

Amanatides, J. and Woo, A. (1987). A fast voxel traversal algorithm for ray tracing. *Proceedings of the European Association for Computer Graphics (EuroGraphics)*. volume 87, EG 1987-Technical Papers.

Ascending Technologies (2018). UAV applications – used worldwide for individual requirements. `www.asctec.de/en/uav-uas-drone-applications/`, accessed on October 25, 2019.

Badino, H. and Kanade, T. (2011). A head-wearable short-baseline stereo system for the simultaneous estimation of structure and motion. *Proceedings of the 12th IAPR Conference on Machine Vision Applications, MVA 2011*, pages 185–189.

Baker, S., Dellaert, F., and Matthews, I. (2001). Aligning images incrementally backwards. *Carnegie Mellon University Pittsburgh, PA USA. The Robotics Institute. Technical report CMU-RI-TR-01-03.*

Black, M. J. and Jepson, A. (1998). Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. *Springer, International Journal of Computer Vision (IJCV)*, **26**(1), 63–84.

Bleyer, M., Rhemann, C., and Rother, C. (2011). Patchmatch stereo - stereo matching with slanted support windows. *Proceedings of the British Machine Vision Conference (BMVC)*, pages 14.1–14.11.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 1222–1239.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. `https://opencv.org/`.

Buczko, M. and Willert, V. (2016). Flow-decoupled normalized reprojection error for visual odometry. *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1161–1167.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **8**(6), 679–698.

Chang, C. and Chatterjee, S. (1992). Quantization error analysis in stereo vision. *Conference Record of the 26th. Asilomar Conference on Signals, Systems and Computers*, **2**, 1037 – 1041.

Conte, S. D. and de Boor, C. (1980). *Elementary Numerical Analysis: An Algorithmic Approach*. International series in pure and applied mathematics, third edition of the book. McGraw-Hill, New York, Montreal.

Corke, P. (2011). *Robotics, Vision and Control - Fundamental Algorithms in MATLAB®*, volume 73. `https://www.youtube.com/watch?v=fVJeJMWZcq8&t=97s`.

Cover, H., Choudhury, S., Scherer, S., and Singh, S. (2013). Sparse tangential network (spartan): Motion planning for micro aerial vehicles. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2820–2825.

Dehn, E. (1960). *Algebraic Equations: An introduction to the theories of Lagrange and Galois*, volume 24. Dover Publications.

Desjardins, J. (2018). Amazon and UPS are betting big on drone delivery. 2018.10.01.

Einecke, N. and Eggert, J. (2010). A two-stage correlation method for stereoscopic depth estimation. *2010 IEEE International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 227–234.

Emilie, H. (1968). On the Schur complement, Basel Mathematical Notes,(University of Basel). *Journal of Combinatorial Theory (JCT)*, **14**.

Engel, J., Schöps, T., and Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. *European Conference on Computer Vision (ECCV)*, **8690**, 1–16.

Engel, J., Stueckler, J., and Cremers, D. (2015). Large-scale direct slam with stereo cameras. *International Conference on Intelligent Robots and Systems (IROS)*.

Fang, Z. and Scherer, S. (2014). Experimental study of odometry estimation methods using RGB-D cameras. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 680–687.

Felzenszwalb, P. and Huttenlocher, D. (2006). Efficient belief propagation for early vision. *International Journal of Computer Vision (IJCV)*, **70**(1), 41–54.

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 Association for Computing Machinery (ACM)*, **24**(6), 381–395.

Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). SVO: Fast semi-direct monocular visual odometry. *IEEE International Conference on Robotics and Automation (ICRA)*.

Forster, C., Zichao, Z., Michael, G., Manuel, W., and Scaramuzza, D. (2016). SVO 2.0: Semi-direct visual odometry for monocular and multi-camera systems. *IEEE Transactions on Robotics (T-RO)*.

Geiger, A., Roser, M., and Urtasun, R. (2011a). Efficient large-scale stereo matching. *Proceedings of the 10th Asian Conference on Computer Vision (ACCV) - Volume Part I*, pages 25–38.

Geiger, A., Ziegler, J., and Stiller, C. (2011b). Stereoscan: Dense 3D reconstruction in real-time. *IEEE Intelligent Vehicles Symposium (IV)*, pages 963 – 968.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361.

Grunert, J. A. (1841). Das pothenotische problem in erweiterter gestalt nebst über seine anwendungen in der geodäsie. *Grunerts Archiv für Mathematik und Physik*, **1**, 238–248.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. *In Proc. of Fourth Alvey Vision Conference*, pages 147–151.

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

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition.

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. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2472 – 2477.

Hirschmüller, H. (2008). Stereo processing by semi-global matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **30**(2), 328–341.

Hirschmüller, H., Innocent, P. R., and Garibaldi, J. (2002). Real-time correlation-based stereo vision with reduced border errors. *Springer, International Journal of Computer Vision (IJCV)*, pages 229–246.

Honig, Z. (2011). T-Hawk UAV enters Fukushima danger zone, returns with video. `http://www.engadget.com/2011/04/21/t-hawk-uav-entersfukushima-danger-zone-returns-with-video/`, accessed 2018.10.01.

J., E., V., K., and D., C. (2011). Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, page 846–894.

Jakob, E., Vladlen, K., and Daniel, C. (2016). Direct sparse odometry. *arXiv reference 1607.02565*. `https://arxiv.org/abs/1607.02565`.

Kanade, T. and Okutomi, M. (1994). A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **16**(9), 920–932.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research, Sage Publications*, **30**(7), 846–894.

Kelly, L. (2018). Unimate: the first industrial robot and why it failed. `https://www.history101.com/unimate-first-industrial-robot`. Accessed: 2019-02-06.

Kerl, C. (2012). Odometry from RGB-D cameras for autonomous quadrocopters. *Master thesis, Technical University Munich (TUM), Department of Informatics, Munich, Germany*.

Klaus, A., Sormann, M., and K., K. (2006). Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. *IEEE International Conference on Pattern Recognition (ICPR)*, pages 15–18.

Klein, G. and Murray, D. W. (2007). Parallel tracking and mapping for small ar workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234.

Klose, S., Heise, P., and Knoll, A. (2013). Efficient compositional approaches for real-time robust direct visual odometry from rgb-d data. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1100–1106.

Kowalczuk, J., Psota, E. T., and Perez, L. C. (2013). Real-time stereo matching on cuda using an iterative refinement method for adaptive support-weight correspondences. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, **23**(1), 94–104.

Kuemmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g2o: A general framework for graph optimization. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613.

Kümmerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., and Kleiner, A. (2009). On Measuring the Accuracy of SLAM Algorithms. *Kluwer Academic Publishers, Autonomous Robots*, **27**(4), 387–407.

Labbé, M. and Michaud, F. (2018). RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Wiley Publisher, Journal of Field Robotics (JFR)*, **36**(2), 416–446.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. *Department of Computer Siene Iowa State University Ames, IA 50011 USA, Citeseer*.

Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Brown University, Quarterly of Applied Mathematics*, **2**(2), 164–168. `http://www.jstor.org/stable/43633451`.

Levitt, T. S. and Lawton, D. T. (1990). Qualitative navigation for mobile robots. *Elsevier Science Publishers Ltd., Artificial Intelligence*, **44**(3), 305–360. `http://dx.doi.org/10.1016/0004-3702(90)90027-W`.

Likhachev, M., Ferguson, D., Stentz, A., and Thrun, S. (2005). Anytime dynamic A*: An anytime, replanning algorithm. *International Conference on Automated Planning and Scheduling (ICAPS)*, **5**, 262–271.

Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, **293**, 133–135. Nature Publishing Group.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, **60**(2), 91–110. Kluwer Academic Publishers, Hingham, MA, USA.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. *International Joint Conferences on Artificial Intelligence Organization*, **2**, 674–679.

Ma, Z., He, K., Wei, Y., Sun, J., and Wu, E. (2013). Constant time weighted median filtering for stereo matching and beyond. *ICCV*.

Madsen, K Nielsen, B. and Tingleff, O. (2004). Methods for non-linear least squares problems. *Informatics and mathematical modelling*.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, **11**(2), 431–441.

Masselli, A. and Zell, A. (2014). A new geometric approach for faster solving the perspective-three-point problem. *IEEE International Conference on Pattern Recognition (ICPR)*, pages 2119–2124.

Meagher, D. (October 1980). Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3D objects by computer. *Rensselaer Polytechniggc Institute (Technical Report IPL-TR-80-111)*.

Meier, L., Tanskanen, P., Heng, L., Lee, G. H., Fraundorfer, F., and Pollefeys, M. (2012). PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, **33**(1-2), 21–39. `ttps://doi.org/10.1007/s10514-012-9281-4`.

Moons, T., Van-Gool, L., and Vergauwen, M. (2010). 3D reconstruction from multiple images part 1: Principles. *Foundations and Trends® in Computer Graphics and Vision*, **4**(4), 287–404. `https://www.nowpublishers.com/article/DownloadSummary/CGV-007`.

Moravec, H. (1980). *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Phd dissertation (cmu-ri-tr3), Carnegie Mellon University, Pittsburgh. PA, USA.

Moravec, H. and Elfes, A. E. (1985). High resolution maps from wide angle sonar. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 116 – 121.

Mozerov, M. G. and van de Weijer, J. (2015). Accurate stereo matching by two-step energy minimization. *IEEE Transactions on Image Processing (TIP)*, **24**(3), 1153–1163.

Mur-Artal, R. and Tardós, J. D. (2016). ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *arXiv preprint arXiv:1610.06475*.

Newcombe, R., Lovegrove, S., and Davison, A. (2011). DTAM: Dense tracking and mapping in real-time. *IEEE International Conference on Computer Vision (ICCV)*, pages 2320–2327.

Nicas, J. (2014). Deutsche Post DHL to deliver medicine via drone. Deutsche Post DHL follows Amazon and Google in testing delivery drones. `https://www.wsj.com/articles/deutsche-post-dhl-to-deliver-medicine-via-drone-1411576151`, accessed October 25 2019.

Nuske, S. T., Choudhury, S., Jain, S., Chambers, A. D., Yoder, L., Scherer, S., Chamberlain, L. J., Cover, H., and Singh, S. (2015). Autonomous exploration and motion planning for an unmanned aerial vehicle navigating rivers. *Journal of Field Robotics*, **32**, 1141 – 1162.

Olsson, C. (2013). Lecture notes in computer vision. *Mathematical Imaging Group, Lund institute of technology,Lund University, Sweden*. `http://www.maths.lth.se/matematiklth/personal/calle/datorseende13/`.

Pollefeys, M., Kolev, K., Aksoy, Y., and Zeisl, B. (2014). 3D vision course, https://www.cvg.ethz.ch/teaching/3dvision/2014/index.php. *Computer Vision and Geometry Group Swiss Federal Institute of Technology in Zurich (ETH Zuerich)*. `https://www.cvg.ethz.ch/teaching/3dvision/2014/index.php`.

Rehder, J., Gupta, K., Nuske, S., and Singh, S. (2012). Global pose estimation with limited gps and long range visual odometry. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 627–633.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. R. (2011). ORB: An efficient alternative to SIFT or SURF. *IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571.

Saxena, A., Schulte, J., and Ng, A. (2007). Depth estimation using monocular and stereo cues. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2197–2203.

Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision (IJCV)*, **47**(1-3), 7–42.

Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nešić, N., Wang, X., and Westling, P. (2014). High-resolution stereo datasets with subpixel-accurate ground truth. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 31–42.

Schauwecker, K. (2014). *Stereo Vision for Autonomous Micro Aerial Vehicles*. PhD dissertation, Faculty of science at the university of Tübingen, Germany. `https://publikationen.uni-tuebingen.de/xmlui/handle/10900/55173`.

Schauwecker, K. and Zell, A. (2014). Robust and efficient volumetric occupancy mapping with an application to stereo vision. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6102–6107. `http://www.ra.cs.uni-tuebingen.de/publikationen/2014/schauwecker_icra2014.pdf`.

Scherer, S., Rehder, J., Achar, S., Cover, H., Chambers, A. D., Nuske, S. T., and Singh, S. (2012a). River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, **32**(5), 189 – 214.

Scherer, S., Dubé, D., and Zell, A. (2012b). Using depth in visual simultaneous localisation and mapping. *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5216–5221. `http://www.cogsys.cs.uni-tuebingen.de/publikationen/2012/scherer2012.pdf`.

Scherer, S. A. and Zell, A. (2013). Efficient Onboard RGBD-SLAM for Fully Autonomous MAVs. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*.

Scherer, S. A., Dube, D., and Zell, A. (2012c). Using Depth in Visual Simultaneous Localisation and Mapping. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5216–5221. St. Paul, Minnesota, USA.

Shade, R. (2011). *Choosing Where To Go: Mobile Robot Exploration*. Ph.D. thesis, University of Oxford, Oxford, United Kingdom.

Shum, H. and Szeliski, R. (2000). Systems and experiment paper: Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision (IJCV)*, **36**, 101–130.

Simon, B. and Matthews, I. (2004). Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision (IJCV)*, **56**(3), 221 – 255.

Sinha, S., Scharstein, D., and Szeliski, R. (2014). Efficient high-resolution stereo matching using local plane sweeps. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1582–1589.

Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2012). Visual SLAM: Why filter? *Elsevier Image and Vision Computing (IVC)*, pages 65–77. volume 30.

Şucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, **19**(4), 72–82. `http://ompl.kavrakilab.org`.

Tomasi, C. and Kanade, T. (1991). Shape and motion from image streams: a factorization method—part 3 detection and tracking of point features technical report CMU-CS-91-132. Technical report, International Journal of Computer Vision (IJCV).

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

Yang, Q., Wang, L., Yang, R., Stewénius, H., and Nistér, D. (2009). Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **31**(3), 492–504.

Yang, Q., Wang, L., and Ahuja, N. (2010). A constant-space belief propagation algorithm for stereo matching. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1458–1465.

Yoon, K.-J. and Kweon, I.-S. (2006). Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **28**(4), 650–656.

Zhang, J. and Singh, S. (2015). Visual-lidar odometry and mapping: Low-drift, robust, and fast. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2174–2181.

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **22**(11), 1330–1334.