INSTITUTE FOR COMPUTER SCIENCE
KNOWLEDGE-BASED SYSTEMS

Master's Thesis

An Analysis of VISUAL MONO-SLAM

Sven Albrecht

October 2009

First supervisor: Prof. Dr. Joachim Hertzberg
Second supervisor: Prof. Dr. Andreas Nüchter

Abstract

This thesis presents an analysis of the VISUAL MONO-SLAM algorithm used to create sparse consistent 3D maps in real-time from images perceived by a monocular hand-held camera developed by Davison et al. [13,16,20]. To understand the workings of VISUAL MONO-SLAM, foundations concerning camera models and lens distortion are presented and followed by an excursion about visual features and image procession techniques. The concept of the Extended Kalman filter is introduced and it is shown how an Extended Kalman filter can be used to obtain both a 6D pose estimation for the camera and position estimates for feature points in a 3D coordinate frame. An encoding for 3D point estimations using inverse depth is presented, allowing for immediate feature initialization without any prior knowledge about the depth of the feature point. It is shown that this encoding performs well even for features at great depth showing small or no parallax in contrast to conventional XYZ encoding. To save computational load a conversion mechanism from inverse depth encoding to common 3 dimensional XYZ encoding for features showing high parallax is discussed. An implementation using OPENCV and OPENGL is used to evaluate the discussed methods in a simulation, on provided sample image sequences and with a real time camera.

Zusammenfassung

Die vorliegende Arbeit beschreibt die Funktionsweise des VISUAL MONO-SLAM Algorithmus zur Erstellung von dünn besetzten 3D Karten vorgestellt von Davison et al. [13, 16, 20]. Als Sensor zur Kartenerstellung dient eine einfache handelsübliche Webcam, die per Hand durch die Umgebung geführt wird. Um den VISUAL MONO-SLAM Ansatz besser verständlich zu machen werden als Grundlagen ein einfaches Kameramodell und seine Erweiterungen zur Behandlung von Linsenfehlern vorgestellt und durch ein Kapitel zur Erkennung von besonderen Bildmerkmalen und verschiedenen Bildverarbeitungstechniken abgerundet. Das Konzept des Erweiterten Kalman-Filters wird vorgestellt und am Beispiel von VISUAL MONO-SLAM praktisch erläutert. Dabei wird gezeigt, wie man mit Hilfe des Erweiterten Kalman-Filters sowohl die 6D Pose der Kamera als auch die 3D Positionen beobachteter Merkmale im 3 dimensionalen Raum schätzen kann. Eine zusätzliche Repräsentationsmöglichkeit für 3D Punkte, bei der die inverse Tiefe mit einfließt, ermöglicht eine sofortige Initialisierung von 3D Merkmalen im Erweiterten Kalman-Filter ohne zusätzliche Informationen über die räumliche Tiefe des Merkmals zu haben. Es wird gezeigt, dass diese Repräsentation, im Gegensatz zur gewöhnlichen XYZ Repräsentation eines 3D Punktes, auch die korrekte Modellierung von Punkte in großer Entfernung mit wenig Parallaxe ermöglicht. Um Rechenkapazität zu sparen wird gezeigt, wie Merkmale mit genügend kleiner Unsicherheit bezüglich ihrer Tiefe in eine gewöhnliche XYZ Repräsentation umgewandelt werden können. Um die vorgestellten Methoden zu bewerten werden einige Experimente in einer Simulationsumgebung, mit festen Bild-Sequenzen und einer in Echtzeit mit einer realen Kamera durchgeführt.

Acknowledgement

I would like to thank Kai Lingemann for his patience concerning LaTeX layout, proofreading and helpful suggestions during implementation and evaluation; Andreas Nüchter for suggesting the topic; Joachim Hertzberg for his robotic courses initiating my interest in this field; Jochen Sprickerhof for debugging and coding style advice and last but not least my parents for financing my studies and their support over the years.

Contents

Chapter 1

Introduction

1.1 Robotic Mapping

Map creation remains a very active field in the robotics and AI community. Especially in the domain of *mobile robotics* reliable sensor information and its comparison with a given model are crucial in order for self-localization and meaningful navigation. To avoid tedious map creation by hand several approaches for automatic map creation have emerged over the past years, with the so called SLAM-approach being one of the most popular at the moment. A great overview over several techniques for map creation is provided by Thrun in [46] and is recommended to familiarize oneself with this topic.

SLAM is short for Simultaneous Localization And Mapping and aggregates a number of approaches of automated map generation without any additional *pose* knowledge apart from sensor information. That means that while the map is constructed the robot has to correctly localize itself in the map it has constructed so far in order to expand the map with new sensor information. The interplay between map construction and localization is crucial in SLAM: If the localization if faulty, new sensor information added to the existing map will not be consistent, thus not resembling the environment. However if the environment is not correctly modeled sensor information gathered by the robot will not correspond with expected sensor measurements suggested by the map and the localization will become erroneous.

The underlying methods (for example probabilistic methods vs. non-probabilistic methods) to solve the SLAM problem differ, oftentimes depending on the type of sensor information available and the time constraints imposed by the application scenario (online map generation vs. batch-processing). Along with the methods and sensor information the resulting maps will differ in their dimensionality (2D or 3D) and their representation of the environment (for example point clouds or occupancy grids). Since SLAM approaches can be discerned by a large amount of attributes, it becomes hard to strictly cluster existing approaches in a meaningful way. In the following a short description of the state of the art in SLAM will be given, distinguished first by the type of sensor employed.

1.1.1 Range finder based approaches

For many applications laser range finders are the sensor of choice. Laser range finders use laser light to measure distances. Thus by rotating it with a known rotation at a fixed position it becomes possible to obtain 3D data points in a reference frame with the range finder at its origin. Dependent one the type of laser range finder complete 3D rotation may be already built-in, thus one *scan* will consist of a full

3D point cloud around the range finder. Other sensor types just obtain distance measurements aligned on a plane with a certain opening angle (for example an opening angle of 180°would return distance measurements of points in a plane to the sides and the front of the range finder). Accuracy, frame rate and the type of scan (full 3D point cloud or plane) are in close correlation with the money one is willing to spend on the laser range finder. However it can generally be said that laser range finders provide far better accuracy and higher frame rates than other common sensors measuring depth like sonar or infra-red sensors. For 2 dimensional maps many successful approaches exist today, often employing probabilistic methods like the Kalman filter and its derivatives (for examples please refer to [46]). A inherent disadvantage of Kalman filter based approaches is that they are monomodal, which means that they can only model one hypothesis at a time. If the pose is lost in a monomodal system (i.e. if the error between the estimation and the real state of the system grows too large) it is hardly ever recovered. To address this problem techniques sustaining multiple hypotheses at a time were introduced. One example for such a technique is the particle filter, where multiple hypotheses and their probability are maintained. If the probability of a single particle becomes to small it is pruned and regularly new particles a spawned to prevent the system to differ to much from one of the sustained hypotheses. An example for a particle filter based SLAM approach can be found in FASTSLAM by Hähnel et al. [23]. Furthermore particle filters are often employed in Monte Carlo Localization, requireing a map of the environment and tackling only the localization aspect of the SLAM problem.

However the vast amount of data obtained in 3 dimensional scans hampers the performance of probabilistic approaches so that for full 6D SLAM (3 coordinates denoting position and 3 angles denoting orientation) non-probabilistic approaches like scan-matching perform well as demonstrated in [35, 36]. Scan matching approaches usually try to fuse two partially overlapping 3D point clouds (scans) into a larger consistent point cloud. As a first guess for the relative translation and orientation of the scans oftentimes odometry information is used. This estimation is refined by minimizing the overall point-to-point distances in both scans via ICP or other suitable algorithms. Thus 3D maps can be built incrementally by fusing a new scan with the already existing combined point cloud. If loops are detected the created map can be made globally consistent through an adaption of the algorithm of Lu an Milios to 6DoF (see Borrmann et al. [5]) or by the recently published ELCH algorithm of Sprickerhof [45].

1.1.2 Vision based approaches

Apart from range finders (including sonar, laser, infra-red and time-of-flight cameras), cameras are also used to construct 3 dimensional maps. Basically 2 different types of cameras can be distinguished: Stereo and mono cameras. Stereo cameras consist of at least two cameras which are arranged in a fixed position to each other and observe the scene. Via *triangulation* (explained later in section 2.4) stereo cameras are able to obtain 3D information from the 2 dimensional data they perceive. Due to the nature of image data and image processing it is not possible to generate dense depth maps or 3D point clouds for fast real-time applications. Therefore 3D maps with vision based sensors are usually sparse and less suited for scan mathing techniques. While the reduced amount of data in such sparse maps do not resemble a complete 3 dimensional model of the environment, probabilistic methods become applicable again for 6D pose estimation.

For example Se, Lowe and Little use in [40, 41] a mobile robot platform equipped with a trinocular stereo head employing SIFT features (see subsection 3.1.2) to gain 3D information from the robots surroundings by epipolar geometry. A first guess for the egomotion of the robot is obtained by its

odometry and the stereo vision system is then used to improve to odometry estimation and determine the position of the visual landmarks. Compared to other real-time visual SLAM approaches the obtained maps are quite dense. To maintain a consistent map Se et al. use Kalman filter techniques to track landmarks and model their uncertainty even in dynamic environments. Davison and Kita [18] equiped a robot with an active stereo head (featuring 4 degrees of rotational freedom) to sequentially create sparse 3D maps on the fly for navigational purposes of the robot. One keypoint in their approach is the matching of visual features by active vision, which means that features promising the most informational gain are preferably matched. Once such a feature is determined the active head can be driven to is predicted position to obtain measurements. Davion and Kita applied an EKF based SLAM algorithm to combine visual information with odometry and inclinometer information to allow for stable localization in undulating terrain. A similar system is used in [19] by Davison and Murray, where they conduct several experiments concerning automatic map growing and pruning as well as comparisons of their estimations with ground truth. However off the shelf stereo systems pricing is above the low-cost segment and the calibration of self-made stereo cameras requires much fine tuning and tends to be sensitive towards shaking often found on moving mobile robots.

If the pose of a single camera is known at each time, moving a single camera may likewise acquire 3D information from images. In case of cameras mounted on top of a robot the current camera pose is usually not known, but can only be estimated as a rough first guess by other sensor information like odometry. For cameras not mounted on a robot, but hand-held devices and the like even these crude information are not available to estimate their poses. Naturally pose estimation is crucial for single camera approaches, since all depth measurements are dependent in the estimated camera poses. Thus single camera SLAM becomes even a bit harder than the SLAM problem with a range sensor: A correct localization and pose estimation is not only necessary to built a consistent map, but also to obtain measurements in the first place. General insight in the domain of visual map generation with a single camera, apart from specific approaches is provided by Lepetit and Fua in [28] where they present an overview of miscellaneous 3D tracking techniques of rigid objects with single cameras. Although 3D tracking has not exactly the same objective as SLAM, both topics are closely related in the case of monocular sensors and share a lot methods.

Single camera techniques can be divided into two subcategories: The first type of approaches use a complete sequence of multiple images to find suitable correspondences between each frame and uses information of all images to estimate camera movement and 3D position of the identified features. These algorithms are off-line in their nature and thus the employed techniques do not have to fulfill real-time constraints. After initial estimations of camera movement and 3D positions of single features, methods to reduce the global error may be applied and finally dense depth maps can be constructed for the given scene. This approach is often called *structure from motion* in the literature and various solutions can be found in the computer vision community. In [21] Fitzgibbon and Zisserman use a multi-step approach to recover geometric information from given video sequences. In a first phase stable features are detected over all input images. In a second step the features are matched and their 3D position is estimated. Afterwards other steps using triangulation and plane fitting are applied to finally generate a VRML scene of the observed geometry. Sato et al. [39] estimate 3D positions of features through a multi-baseline approache and fuse the resulting depth maps in a voxel space model of the environment. A few predifined markers with known 3D positions are sufficient to scale the positions of automatically generated feature points in a consistent manner. Pollefeys et al. describe in [38] an approach to recover 3D information from uncalibrated video sequences. The resultant textured 3D models are fused in an

exemplary application with real video sequences to create a new virtually augmented scene.

Other approaches do not analyze a complete sequence of images, but incrementally incorporate information gathered from a single image in their estimations before considering the next image. While such an approach inhibits an analysis and reduction of the overall error it potentially allows for online or real-time map creation, since only the current image has to be processed. Currently the method of choice for pose estimation of the camera and estimation of feature positions is the Extended Kalman Filter. One of the forerunners in this domain is Andrew Davison who in collaboration with others publishes extensively on this topic [11–13, 16, 17, 20, 33]. Usually such approaches need a certain number of features with known positions in order to work and new features have to be observed over a certain period to guess their depth before they can be added to the EKF. Recently Davison, Civera et al. introduced an alternative feature representation. By representing a 3D point by a 6 dimensional vector employing inverse depth, new features can be added without any prior knowledge and contribute to overall state estimation even if they show little or no parallax.

The purpose of this thesis is to introduce the reader to this novel approach. Firstly the needed background knowledge concerning camera models and image processing will be provided. Afterwards the underlying mechanisms of the VISUAL MONO-SLAM algorithm without prior knowledge as presented by Civera, Davison and Montiel in [13] are discussed and analysed in detail. The main focus is the complete derivation of the EKF mathematics and their meaning in VISUAL MONO-SLAM, with additional information concerning an inverse depth representation of 3D points compared to the conventional XYZ representation. The discussed methods are evaluated in a simulated environment and with real image sequences to provide the reader with information concerning the quality of the VISUAL MONO-SLAM approach.

1.2 Thesis Outline

Chapter 1: A short explanation of the SLAM problem along with different solution approaches. Furthermore an outline of the complete thesis.

Chapter 2: An introduction to the basic theoretical camera model commonly used in computer vision and its extensions to better fit real world cameras.

Chapter 3: A description of different image interest operators, including Harris Corners, SIFT and SURF descriptors. This is followed by some general remarks about image processing, the benefits of integral images and a mechanism to compare image patches.

Chapter 4: An in-depth analysis of the VISUAL MONO-SLAM algorithm. This includes a brief introduction of the Extended Kalman Filter and how this concepts can be used to correctly model the specific demands of the VISUAL MONO-SLAM application. Furthermore two alternative encoding methods for a given 3D point and their adavantages and disadvantages are discussed.

Chapter 5: To evaluate the workings of VISUAL MONO-SLAM a simulation environment is presented along with results obtained from the simulation. Practical evaluation for given image sequences using Shi-Tomasi based features and SURF are compared qualitatively and presented alongside real-time experimental results.

Chapter 6: This chapter concludes the thesis, presenting the findings, open topics for future work and remarks.

Chapter 2

Camera

As mentioned in the introduction in chapter 1 the only sensor information in Mono-SLAM is gathered from a standard low-cost USB digital camera. Low-cost devices typically use a CMOS sensor and do not exceed an image resolution of 640×480 pixels.

The following chapter will first introduce an ideal basic camera model in section 2.1 and show how this ideal model can be modified with a distortion model (section 2.2) to better fit the imperfections found in real camera lenses. How to estimate model parameters for the distortion model will be discussed in section 2.3. The remainder of this chapter will explain in section 2.4 how to estimate 3D positions from the collected 2D data in subsequent camera images.

2.1 The Pinhole Camera Model

This section will first introduce the basic *pinhole camera model* in subsection 2.1.1. Although the model requires some assumptions lacking in real cameras it still allows for a reasonable first approximation. Due to its mathematical convenience and simplicity it is nowadays widely used in the domain of computer vision and computer graphics. In subsection 2.1.2 some relaxations of the assumptions of the basic model presented in 2.1.1 are introduced to better emulate properties found in real cameras.

2.1.1 The basic model

The pinhole camera model consists of 2 dimensional plane, dividing a 3 dimensional coordinate system. The two dimensional plane is referred to as the *pinhole plane* and it features an infinitesimal hole (the eponymous *pinhole*). The pinhole corresponds to the origin O of the 3 dimensional camera coordinate system and is also known as the *optical center* of the camera. The coordinate axes are referred to as X_c, Y_c and Z_c, where X_c points to the side of the camera, Y_c points up and Z_c is pointing in the viewing direction of the camera. Thus the plane generated by X_c and Y_c corresponds with the pinhole plane. The Z_c-axis is often referred to as the *optical axis* or *principal axis*. The *image plane* is parallel to the pinhole plane and located at distance $f, f > 0$ from the origin O along the negative Z_c-axis. The intersection of the image plane with the negative Z_c-axis is called *principal point* or *image center* and denoted as R. The 3 dimensional world in front of the pinhole camera (i.e. in direction of the positive Z_c-axis) will be projected through the aperture on the image plane. The distance between image plane and pinhole plane

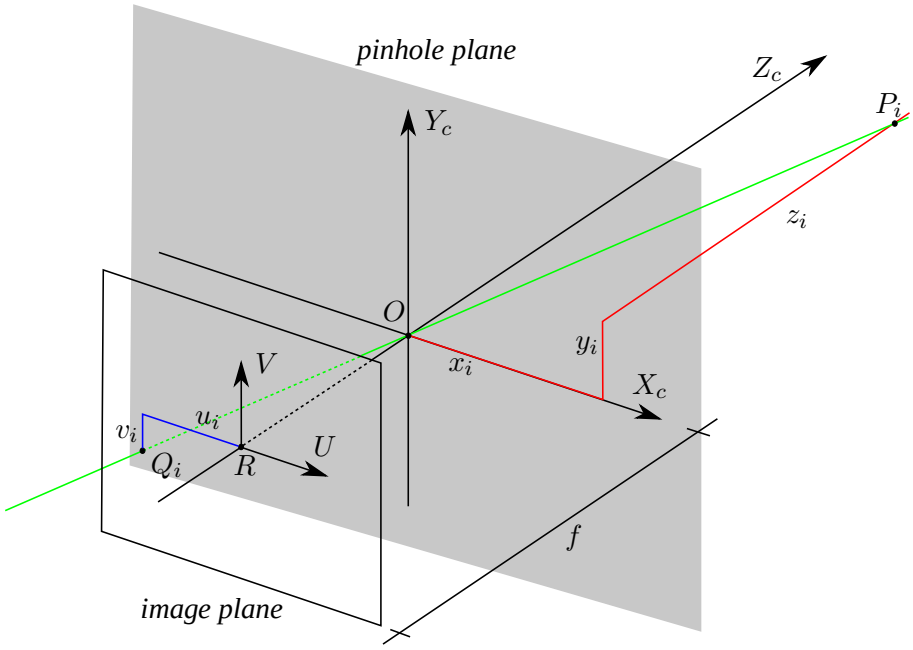


Figure 2.1: Pinhole camera model. The red lines indicates the displacement from the optical center O in X_c, Y_c and Z_c direction for P_i, while the blue lines show the displacement of the projection Q_i from the image center R. The ray from point P_i through the optical center O to its projection Q_i is shown in green. The focal length f defines the distance between the pinhole plane and the image plane.

is the *focal length* of the pinhole camera.

Since the model assumes the pinhole to be of infinitesimal size, from any 3 dimensional point $P_i = (x_i, y_i, z_i)^T, z_i > 0$, exactly one ray of light will pass through the pinhole and project this point on the image plane at its image coordinates $Q_i = (u_i, v_i)^T$. According to the intercept theorems the following equation holds:

$$\begin{pmatrix} -\frac{u_i}{f} \\ -\frac{-v_i}{f} \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \end{pmatrix} \implies \begin{pmatrix} -u_i \\ -v_i \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} f \\ \frac{y_i}{z_i} f \end{pmatrix} \tag{2.1}$$

Due to this assumption the projection of the 3 dimensional world on the image plane is always in focus. The projection of a point P_i through the optical center onto the image plane is depicted in Figure 2.1.

While the image coordinates (u_i, v_i) for any point P_i are uniquely determined by the focal length f, the reverse cannot be determined. Two 3 dimensional points P_i and P_j are projected on the same image coordinate, if $\frac{x_i}{z_i} = \frac{x_j}{z_j}$ and $\frac{y_i}{z_i} = \frac{y_j}{z_j}$ hold. That means for a given pair of image coordinates $(u_i, v_i)^T$ any 3 dimensional point $P_i = (x_i, y_i, z_i)^T, z_i > 0$ on the line going through $(u_i, v_i)^T$ and the optical center O could create the given image coordinates. This follows directly from equation (2.1).

To further simplify the pinhole camera model oftentimes a *virtual image plane* is introduced. Like the image plane the virtual image plane is parallel to the pinhole plane, but it is located at distance f on the opposite side of the pinhole plane. Every point $P_i = (x_i, y_i, z_i)^T, z_i > f$ is projected on the virtual image plane in the same way as on the image plane, but the image coordinates are not inverted. Thus

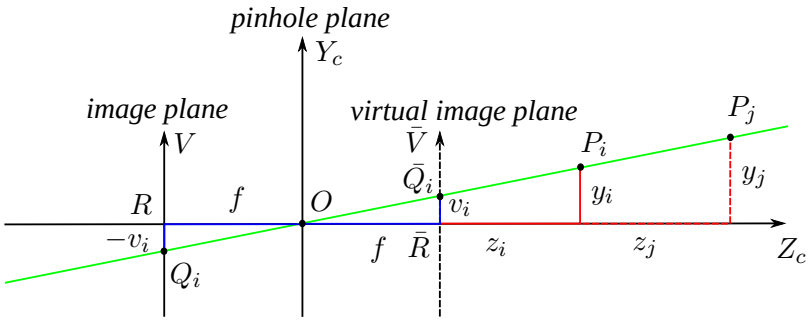


Figure 2.2: Pinhole camera model as seen from the X_c-axis. Note that points P_i and P_j will be projected on the same point $Q_i = (u_i, v_i)^T$ on the image plane. In front of the pinhole plane the virtual image plane is depicted with a dashed line. In contrast to the projection Q_i on the image plane the projection on the virtual image plane $\bar{Q}_i$ is not inverted.

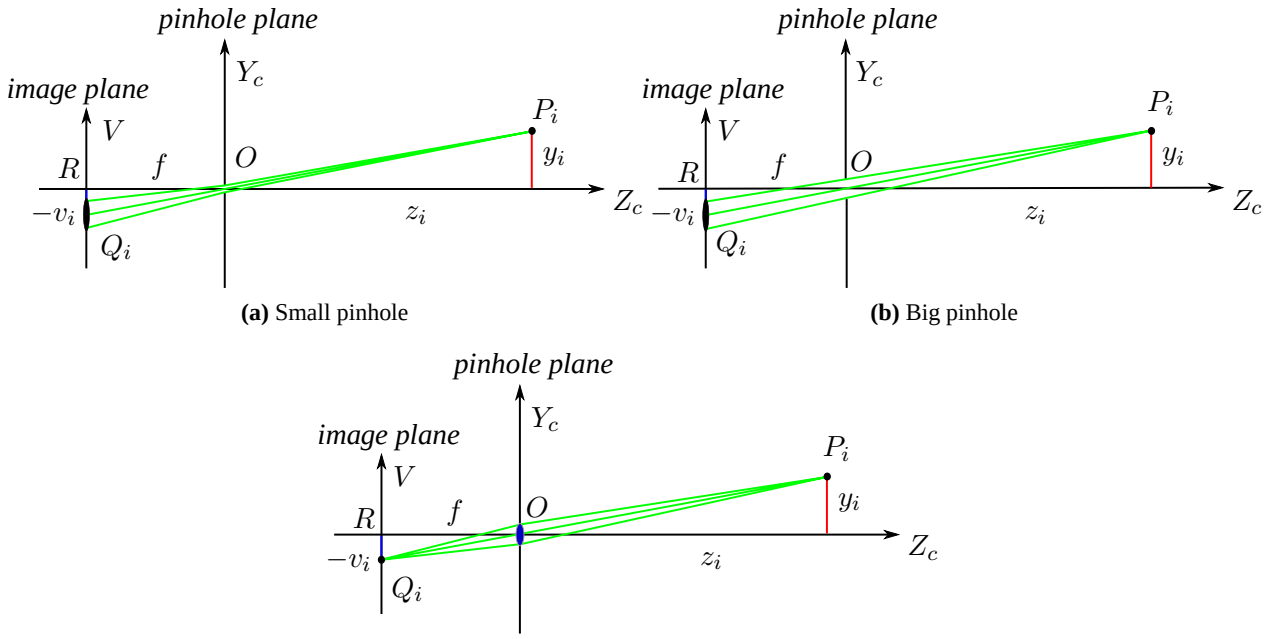
equation (2.1) can be transferred to the generally more convenient form

$$\begin{pmatrix} \frac{u_i}{f} \\ \frac{v_i}{f} \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \end{pmatrix} \implies \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} f \\ \frac{y_i}{z_i} f \end{pmatrix} \tag{2.2}$$

The pinhole camera model with a virtual image plane is depicted in Figure 2.2. Equation (2.2) (or (2.1) for that matter) implies also some other common effects on the projection of 3 dimensional objects: Increasing the distance between an object and the camera will result in a smaller projection, since this is equivalent to increasing z_i of a given point P_i. Furthermore parallel lines on a plane not parallel to the pinhole plane will not be parallel in the projection on the image plane. For example image a line on the X_c-Z_c-plane, parallel to the Z_c-axis. For all points P_i on such a line coordinates x_i and y_i stay constant, while z_i varies. Thus the more distant a point on this line is from the origin O the closer its projection will be to the image center R and the projection of the line will be at image center R on the horizon of the projection. The projection on the (virtual) image plane induced by the pinhole camera model is called *perspective projection* and closely resembles output generated by commonly used cameras. Apart from perspective projection, other projection models namely *affine projection* and *spherical projection* are sometimes used in computer vision. The properties of these projection models differ from perspective projection, which can be beneficial in certain applications. However, affine and spherical projection do not contribute to the remainder of this thesis, so the interested reader is kindly referred to [22] for more details.

2.1.2 Adaption of the Basic Model

The basic pinhole camera model (see 2.1.1) can be modified at several points in order to better describe properties of real cameras. Most cameras use photographic lenses instead of a pinhole. Though an infinitesimal small pinhole in the basic model provides a projection always in focus, in reality shrinking the hole beyond a certain point is not beneficial. The smaller the hole becomes, the less light will pass through the hole and the material on the image plane (either CCD/CMOS-Sensors or photographic film). If not enough light is passed through the pinhole the projection might eventually become dark with little distinctions, since the image sensor usually requires a certain amount of light to be successfully



(a) Small pinhole

(b) Big pinhole

(c) Pinhole with lens

Figure 2.3: Projection under different pinholes. **(a)**: The small pinhole causes diffraction, resulting in a blurred projection. Furthermore less light than in (b) or (c) is permitted, eventually leading to dark projections, low in contrast. **(b)**: The projection Q_i of point P_i appears out of focus and blurred, since multiple rays of light reflected from this point are projected on several different image coordinates. **(c)**: Notice that the projection of point $P_i = (x_i, y_i, z_i)^T$ is in focus and corresponds therefore with with one single point $Q_i = (u_i, v_i)^T$ on the image sensor. Still the lens admits more light than the camera with a small pinhole (a).

triggered in case of digital cameras. In addition a small hole might cause diffraction and therefore blur the projection on the image plane. This is visualized in Figure 2.3a. However if the hole is too big, the projection will be out of focus, since a single 3D point $P_i = (x_i, y_i, z_i)^T$ will be projected on several image points (see Figure 2.3b). Using one or more lenses allows for a bigger hole to provide more light and still leave the projection in focus. A graphical comparison of the three scenarios above is depicted in Figure 2.3c.

Since most lenses have certain imperfections these need to be modeled separately in order to be consistent with the basic pinhole camera model. How to exactly determine the imperfections in the lens of a given camera and how to adapt the model to these imperfections will be discussed in section 2.3 in detail.

Furthermore in real cameras the world is not projected on an image plane, but on a planar light-sensitive material (CCD/CMOS in digital cameras) of limited size which will in the following be referred to as *image sensor* for simplification. The size of this sensor limits the *field of view* of the camera. While in the basic pinhole camera model every point $P_i = (x_i, y_i, z_i)^T$ where $z_i > 0$ can be projected on the image plane, not every point fulfilling this condition is projected on a sensor of limited size. The

boundaries of the image sensor specify the size of the field of view according to the following formula:

$$\begin{pmatrix}\alpha_u\\ \alpha_v\end{pmatrix} = 2\begin{pmatrix}\arctan\left(\frac{U_{\text{dim}}}{2f}\right)\\ \arctan\left(\frac{V_{\text{dim}}}{2f}\right)\end{pmatrix} \tag{2.3}$$

where f is the focal length, $U_{\text{dim}} \times V_{\text{dim}}$ specifies the physical size of the image sensor (for example $3.6\,\text{mm} \times 2.7\,\text{mm}$) and $(\alpha_u, \alpha_v)^T$ yields the opening angles of the field of view. These relations are illustrated in Figure 2.4. In the case of digital camera equation (2.3) can also be stated as

$$\begin{pmatrix}\alpha_u\\ \alpha_v\end{pmatrix} = 2\begin{pmatrix}\arctan\left(\frac{\text{width}\cdot d_u}{2f}\right)\\ \arctan\left(\frac{\text{height}\cdot d_v}{2f}\right)\end{pmatrix} \tag{2.4}$$

where $\text{width} \times \text{height}$ states the image resolution in pixels and d_u and d_v refer to the physical width and height of a pixel on the image sensor. Placing the image sensor in a way that corresponds to the inverted image coordinates (i.e. the image sensor is placed upside down) of the basic pinhole camera model (see equation (2.1)). emulates the effect of the virtual image plane (see equation (2.2)). Not only does the size of the image sensor determined the field of view, but it also affects the size and resolution of the created image. The most commonly resolutions found in low-cost digital cameras are currently either 320×240 pixels or 640×480 pixels. Since the origin of images in computer applications is located in the top left corner of the image it is convenient to incorporate this into the model by shifting the origin of the image coordinate system. The resulting pixel coordinates $(\text{u}_i, \text{v}_i)^T$ in the obtained image are calculated by

$$\begin{pmatrix}\text{u}_i\\ \text{v}_i\end{pmatrix} = \begin{pmatrix}\lfloor ku_i + o_u + 0.5\rfloor\\ \lfloor lv_i + o_v + 0.5\rfloor\end{pmatrix} = \begin{pmatrix}\lfloor kf\frac{x_i}{z_i} + o_u + 0.5\rfloor\\ \lfloor lf\frac{y_i}{z_i} + o_v + 0.5\rfloor\end{pmatrix} \tag{2.5}$$

where o_u and o_v describe the displacement of the upper left corner of the image from its center in pixel units. If a camera has a resolution of $\text{width} \times \text{height}$ then $o_u = \frac{\text{width}}{2}$ and $o_v = \frac{\text{height}}{2}$. Note that while $(u_i, v_i)^T$ with $u_i, v_i \in \mathbb{R}$ are tuples of real values, the corresponding pixel coordinates $(\text{u}_i, \text{v}_i)^T$ are integer values and only perceivable of the image sensor if $\text{u}_i \wedge \text{v}_i \in \mathbb{Z}^+,\ 0 \leq \text{u}_i < \text{width}$ and $0 \leq \text{v}_i < \text{height}$ holds. Parameters k and l are scale parameter to map the distance obtained by $f\frac{x_i}{z_i}$ and $f\frac{y_i}{z_i}$ respectively to the corresponding pixels. If distance f for instance is measured in m, then a pixel has the dimension of $\frac{1}{k} \times \frac{1}{l}$, where k and l are expressed in pixel $\cdot\, 1/\text{m}$. Oftentimes the dimensions of a pixel are denoted as $d_u \times d_v$, where d_u and d_v are measured in m or any other distance measure. Thus equation (2.5) is often also denoted as

$$\begin{pmatrix}\text{u}_i\\ \text{v}_i\end{pmatrix} = \begin{pmatrix}\lfloor \frac{f}{d_u}\frac{x_i}{z_i} + o_u + 0.5\rfloor\\ \lfloor \frac{f}{d_v}\frac{y_i}{z_i} + o_v + 0.5\rfloor\end{pmatrix} = \begin{pmatrix}\lfloor f_u\frac{x_i}{z_i} + o_u + 0.5\rfloor\\ \lfloor f_v\frac{y_i}{z_i} + o_v + 0.5\rfloor\end{pmatrix} \tag{2.6}$$

where parameters k and l of equation (2.5) have been replaced by $\frac{1}{d_u}$ and $\frac{1}{d_v}$. Thus parameters f_u and f_v express the focal length f in terms of pixel-units and are typically obtained by camera calibration methods instead of the actual focal length f. The distinction between f_u and f_v becomes necessary, since the physical pixels on an image sensor are not always squared, but sometimes only rectangular. To avoid unnecessarily cluttered equations in the following the explicit rounding to the nearest integer value will be omitted. Please keep in mind that perceived image coordinates are nevertheless always integer values.

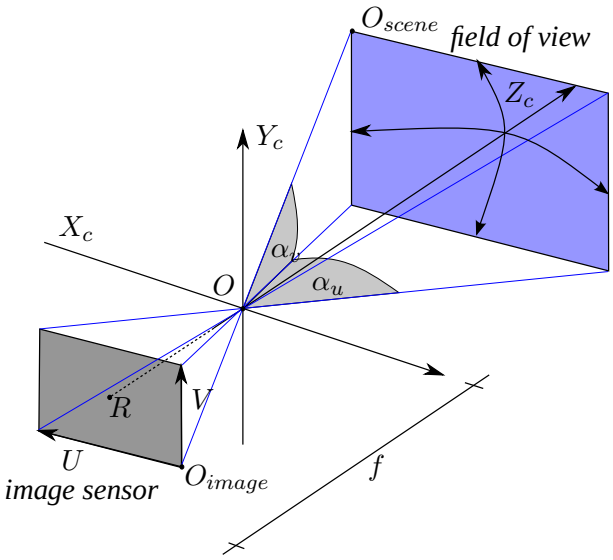


Figure 2.4: Field of View. The boundaries of the field of view, determined by the size of the *image sensor* are depicted in blue. The resulting volume could be extended further in direction of the positive X_c and is just cut for visualization purposes. The corresponding angles, defining the field of view are labeled as α_u and α_v (see equations (2.3) and (2.4)), respectively. Please note that the origin for the resulting image (its upper left corner) is placed at O_{image} which is inverted from its corresponding real world position O_{scene}.

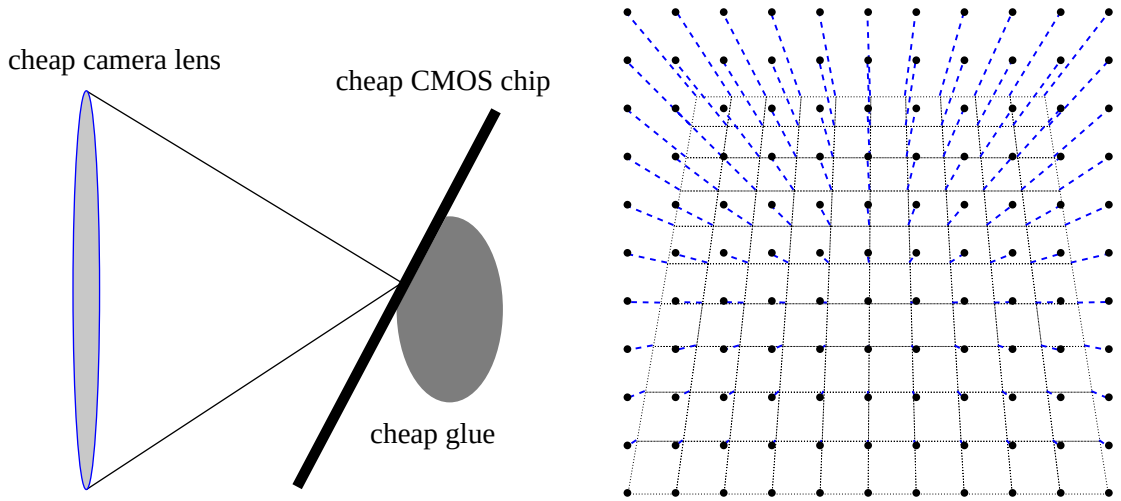
2.2 A Simple Distortion Model

Section 2.1.2 introduced a first step to adapt the basic pinhole camera model to represent real cameras more closely, by means of the field of view and the adaption of the origin to reflect digital image sensors. However the adaptions discussed so far still describe a "perfect" camera. Each real camera tends to have its own unique imperfections, introduced by imperfect manufacturing and assembly processes. Especially in the low-cost sector individual cameras of the same type tend to vary in their camera properties. One possible error induced by the manufacturing process is depicted in Figure 2.5, where the image sensor is not aligned perfectly parallel to the camera lens.

Since the displacement of the image sensor is unique for each produced camera a way to analyze the inherent distortions and to model them is needed. A first step is to further modify equation (2.6). Equation (2.6) implies that the center of the image sensor is placed precisely at the ideal image center R which is the intersection with the Z_c-axis and the image plane. Considering the CMOS sensor of the HERCULES WebCam Classic with size $3.6\,\mathrm{mm} \times 2.7\,\mathrm{mm}$ yields 640×480 pixels of resolution, the physical size for the sensor of one pixel on the chip is $0.005625\,\mathrm{mm} \times 0.005625\,\mathrm{mm} = 5.625\,\mu\mathrm{m} \times 5.625\,\mu\mathrm{m}$. So any displacement larger than $\frac{5.625}{2}\,\mu\mathrm{m}$ would mean that (2.6) is not accurate, still assuming that the image sensor is perfectly parallel to the pinhole plane. If however the displacement of the center of the image sensor is known, the adaption of (2.6) becomes quite simple:

$$\begin{pmatrix} \mathrm{u}_{\mathrm{d},i} \\ \mathrm{v}_{\mathrm{d},i} \end{pmatrix} = \begin{pmatrix} f_u \frac{x_i}{z_i} + u_0 \\ f_v \frac{y_i}{z_i} + v_0 \end{pmatrix} \tag{2.7}$$

where u_0 and v_0 denote the image coordinate, which corresponds to the image center R. Similar to



(a) Exemplary cheap camera schematics **(b)** Tangential distortion

Figure 2.5: Exemplary result of production inaccuracy. **(a)** Illustrates a common displacement of the CMOS sensor, which should ideally be parallel the the camera lens. Such a displacement induces tangential distortion, depicted in **(b)**: For an ideal camera the black dots positioned in a grid should be projected, preserving the undistorted uniform grid structure. The CMOS sensor displacement induces the projection of a distorted grid, shown by dotted black lines. Correspondences between the original black dots and their projection are shown in blue.

equation (2.6) $\mathrm{u}_{\mathrm{d},i}$ and $\mathrm{v}_{\mathrm{d},i}$ denote integer values bounded by 0 and width and height respectively, while u_i, v_i, u_0 and v_0 are given as real values. The coordinates of point $P_i = (x_i, y_i, z_i)^T$ are also specified by real values, of course. The pair $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ are referred to as the *distorted image coordinates* (marked by subscript $_{\mathrm{d}}$), since they are still subject to lens distortion although they are calculated according to the actual image center $(u_0, v_0)^T$. Distortions like the one depicted in Figure 2.5 are more complicated to deal with and will be discussed together with radial distortions in the following paragraph.

Flaws and imperfections in a camera are not only limited to the correct placement of the image sensor, but can (and usually do) also occur in the lens or lens system of a camera. A compact overview on properties of optical lenses like *aberration* or *vignetting* is presented in [22]. A far more detailed discussion of these topics can be found in [25, 27].

However once a suitable *distortion model* is applied on top of the current camera model lens effects like aberration and vignetting can be ignored in the VISUAL MONO-SLAM context and are therefore not discussed any further. The interested reader is referred to [22, 25, 27] for more information on these topics.

Closely related to a distortion model is the *camera calibration* process. The distortion model describes how to calculate the ideal undistorted image coordinates $(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ which would be perceived by an ideal camera from a pair of actually perceived distorted image coordinates $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ for a given point $P_i = (x_i, y_i, z_i)^T$. Camera calibration describes techniques to estimate the unique camera parameters, used in a distortion model for an individual camera. A historical overview of different calibration techniques and their corresponding models is presented in [14]. The distortion model used by Davison et al. in [13, 16, 20] is a simplified version of the "Brown-Conrady-model" (also known as "plumb bob

model") proposed by Brown in 1965 [8]. This model uses 5 *distortion coefficients* to describe the imperfections found in a camera image. If all 5 coefficients are known it is quite simple to compute the corresponding undistorted image coordinates of an ideal camera for a given pair of distorted image coordinates. The next paragraph will present the whole model first and offer some explanations concerning the different components of the model. Subsequently the model will be simplified to the model used in [13, 16, 20]. In section 2.3 the techniques used to estimate the distortion coefficients will be shown as well as some exemplary results of camera calibration.

The model of Brown distinguishes between two different kinds of distortions, namely *radial distortion* and *tangential distortion*. Radial distortions are caused by the shape of the used lens and can cause *pincushion* or *barrel* distortions of the image, where straight lines will be projected in a curved fashion. Usually the effects of radial distortion become stronger the larger the distance between a projected point $Q_i = (\mathrm{u}_i, \mathrm{v}_i)^T$ and the image center $R = (u_0, v_0)^T$ becomes. Tangential distortion generally refers to distortions due to imperfections in the centering of the camera lens. To model radial distortion 3 distortion coefficients are used (k_1, k_2, k_3) while tangential distortion is described by 2 coefficients (p_1, p_2). If the radial distortion coefficients are known, the correction terms $(\hat{\mathrm{u}}_i, \hat{\mathrm{v}}_i)^T$ for radial distortion are defined as:

$$\begin{pmatrix} \hat{\mathrm{u}}_i \\ \hat{\mathrm{v}}_i \end{pmatrix} = \begin{pmatrix} \mathrm{u}_{\mathrm{d},i} - u_0 \\ \mathrm{v}_{\mathrm{d},i} - v_0 \end{pmatrix} \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{2.8}$$

where

$$r = \sqrt{\left(\frac{1}{f_u}((\mathrm{u}_{\mathrm{d},i} - u_0)\right)^2 + \left(\frac{1}{f_v}(\mathrm{v}_{\mathrm{d},i} - v_0)\right)^2} \tag{2.9}$$

denotes the distance of pixel $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ from the image center $(u_0, v_0)^T$ on the image sensor. To recap how to calculate $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ please refer to equation (2.7).

To compensate for tangential distortion effects the following terms are proposed, incorporating tangential distortion coefficients p_1 and p_2:

$$\begin{pmatrix} \tilde{\mathrm{u}}_i \\ \tilde{\mathrm{v}}_i \end{pmatrix} = \begin{pmatrix} f_u \left(2p_1 \mathrm{u}_{\mathrm{d},i} \mathrm{v}_{\mathrm{d},i} + p_2(r^2 + 2\mathrm{u}_{\mathrm{d},i}^2)\right) \\ f_v \left(p_1(r^2 + 2\mathrm{v}_{\mathrm{d},i}^2) + 2p_2 \mathrm{u}_{\mathrm{d},i} \mathrm{v}_{\mathrm{d},i}\right) \end{pmatrix} \tag{2.10}$$

where r is defined as in equation (2.9).

The complete model, considering radial (2.8) and tangential (2.10) distortion defines the undistorted image coordinates $(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ as

$$\begin{aligned} \begin{pmatrix} \mathrm{u}_{\mathrm{u},i} \\ \mathrm{v}_{\mathrm{u},i} \end{pmatrix} &= \begin{pmatrix} \hat{\mathrm{u}}_i + \tilde{\mathrm{u}}_i \\ \hat{\mathrm{v}}_i + \tilde{\mathrm{v}}_i \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \\ &= \begin{pmatrix} f_u \left(\frac{\mathrm{u}_{\mathrm{d},i} - u_0}{f_u}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2p_1 \mathrm{u}_{\mathrm{d},i} \mathrm{v}_{\mathrm{d},i} + p_2(r^2 + 2\mathrm{u}_{\mathrm{d},i}^2)\right) \\ f_v \left(\frac{\mathrm{v}_{\mathrm{d},i} - v_0}{f_v}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + p_1(r^2 + 2\mathrm{v}_{\mathrm{d},i}^2) + 2p_2 \mathrm{u}_{\mathrm{d},i} \mathrm{v}_{\mathrm{d},i}\right) \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \\ &= \begin{pmatrix} f_u \frac{x_i}{z_i}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + 2p_1 f_u \frac{x_i}{z_i}\frac{y_i}{z_i} + p_2\left(r^2 + 2f_u \frac{x_i^2}{z_i^2}\right) \\ f_v \frac{y_i}{z_i}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) + p_1\left(r^2 + 2f_v \frac{y_i^2}{z_i^2}\right) + 2p_2 f_v \frac{x_i}{z_i}\frac{y_i}{z_i} \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \end{aligned} \tag{2.11}$$

Luckily in most cases this full distortion model is not actually needed, but can be a bit simplified.

As mentioned earlier Davison et al. [13, 16, 20] propose a simplified version of the general model. For standard field of view cameras (i.e. non-wide angle or "fish-eye" cameras) it is not recommended or necessary to push the radial component of the distortion beyond the 4th order. Therefore only cameras providing a highly distorted image actually need distortion coefficient k_3, in other cases it is advised to assume $k_3 = 0$. Currently most manufactured cameras are assembled with very little imperfection in centering the image sensor, so that tangential distortion becomes less important. This corresponds with the observations by Zhang [50] who states that the distortion function is clearly dominated by coefficients k_1 and k_2. In other words for most low-cost cameras the assumption $p_1 = p_2 = 0$ is valid. That leaves a distortion model with just 2 radial distortion coefficients (namely k_1 and k_2) which corresponds to the distortion model used in [50]:

$$\begin{pmatrix} \mathrm{u}_{\mathrm{u},i} \\ \mathrm{v}_{\mathrm{u},i} \end{pmatrix} = \begin{pmatrix} (\mathrm{u}_{\mathrm{d},i} - u_0)\left(1 + k_1 r^2 + k_2 r^4\right) \\ (\mathrm{v}_{\mathrm{d},i} - v_0)\left(1 + k_1 r^2 + k_2 r^4\right) \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \tag{2.12}$$

The distortion model used by Davison et al. in [13, 16, 20] is quite similar to the distortion model of Zhang [50]. Davison et al. define an undistortion function h_u that maps a pair of distorted image coordinates $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ to a pair of undistorted image coordinates $(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ as follows:

$$\begin{pmatrix} \mathrm{u}_{\mathrm{u},i} \\ \mathrm{v}_{\mathrm{u},i} \end{pmatrix} = h_u \begin{pmatrix} \mathrm{u}_{\mathrm{d},i} \\ \mathrm{v}_{\mathrm{d},i} \end{pmatrix} = \begin{pmatrix} (\mathrm{u}_{\mathrm{d},i} - u_0)\left(1 + \mathring{k}_1 r_d^2 + \mathring{k}_2 r_d^4\right) \\ (\mathrm{v}_{\mathrm{d},i} - v_0)\left(1 + \mathring{k}_1 r_d^2 + \mathring{k}_2 r_d^4\right) \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \tag{2.13}$$

with

$$r_d = \sqrt{\left(d_u\left(\mathrm{u}_{\mathrm{d},i} - u_0\right)\right)^2 + \left(d_v\left(\mathrm{v}_{\mathrm{d},i} - v_0\right)\right)^2} \tag{2.14}$$

Apart from the distance of the distorted image coordinates $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ to the image center defined in equation (2.14) the distortion model (2.13) used in [13, 16, 20] corresponds to the distortion model (2.12) of Zhang [50]. For the definition of distance r in Zhangs's model, please refer to equation (2.9). However the difference between r and r_d also means that the distortion coefficients are generally not equal (i.e. $k_1 \neq \mathring{k}_1$ and $k_2 \neq \mathring{k}_2$). From camera calibration (see 2.3) usually coefficients k_1 and k_2 are obtained, but not $\mathring{k}_1, \mathring{k}_2$. Since (2.12) and (2.13) should yield the same mapping for a pair $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ of distorted image coordinates the assumptions $\mathring{k}_1 r_d^2 = k_1 r^2 \Leftrightarrow \mathring{k}_1 = k_1 \frac{r^2}{r_d^2}$ and $\mathring{k}_2 r_d^4 = k_2 r^4 \Leftrightarrow \mathring{k}_2 = k_2 \frac{r^4}{r_d^4}$ are valid.

Thus $\mathring{k}_1$ and $\mathring{k}_2$ can be calculated according to the following equations:

$$\begin{aligned}
\mathring{k}_1 &= k_1 \frac{r^2}{r_d^2} \\
&= k_1 \frac{1}{f_u^2 f_v^2} \frac{f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 + f_u^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2}{d_u^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 + d_v^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2} \\
&= k_1 \frac{1}{f_u^2 f_v^2} \frac{f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 + f_u^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2}{f^2 \left(\frac{1}{f_u^2}\left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 + \frac{1}{f_v^2}\left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2\right)} \\
&= k_1 \frac{1}{f_u^2 f_v^2 f^2} \frac{f_u^2 f_v^2 \left(f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 + f_u^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2\right)}{f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 + f_u^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2} \\
&= k_1 \frac{1}{f^2}
\end{aligned} \tag{2.15}$$

and

$$\begin{aligned}
\mathring{k}_2 &= k_2 \frac{r^4}{r_d^4} \\
&= k_2 \frac{1}{f_u^4 f_v^4} \frac{f_v^4 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^4 + 2 f_u^2 f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2 + f_u^4 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^4}{d_u^4 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^4 + 2 d_u^2 d_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2 + d_v^4 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^4} \\
&= k_2 \frac{1}{f_u^4 f_v^4} \frac{f_v^4 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^4 + 2 f_u^2 f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2 + f_u^4 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^4}{f^4 \left(\frac{1}{f_u^4}\left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^4 + 2\frac{1}{f_u^2 f_v^2}\left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2 + \frac{1}{f_v^4}\left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^4\right)} \\
&= k_2 \frac{1}{f_u^4 f_v^4 f^4} \frac{f_u^4 f_v^4 \left(f_v^4 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^4 + 2 f_u^2 f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2 + f_v^4 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^4\right)}{f_v^4 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^4 + 2 f_u^2 f_v^2 \left(\mathrm{u}_{\mathrm{d},i} - u_0\right)^2 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^2 + f_v^4 \left(\mathrm{v}_{\mathrm{d},i} - v_0\right)^4} \\
&= k_2 \frac{1}{f^4}
\end{aligned} \tag{2.16}$$

While camera calibration usually obtains only f_u and f_v the focal length f can be calculated with the help of d_u and d_v, which can usually be derived from the data sheet of the camera. In the remainder of this thesis undistortion will always be calculated according to equation (2.13). For readability purposes the use of the $\mathring{}$ symbol to indicate the distortion coefficients of (2.13) will be omitted henceforth.

Sometimes it might prove useful or be necessary to obtain for a given pair of undistorted image coordinates $(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ the corresponding distorted coordinates $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$. For a given pair of undistorted coordinates are the distorted coordinates are calculated as

$$\begin{pmatrix} \mathrm{u}_{\mathrm{d},i} \\ \mathrm{v}_{\mathrm{d},i} \end{pmatrix} = h_d \begin{pmatrix} \mathrm{u}_{\mathrm{u},i} \\ \mathrm{v}_{\mathrm{u},i} \end{pmatrix} = \begin{pmatrix} u_0 + \frac{(\mathrm{u}_{\mathrm{u},i} - u_0)}{\left(1 + k_1 r_d^2 + k_2 r_d^4\right)} \\ v_0 + \frac{(\mathrm{v}_{\mathrm{u},i} - v_0)}{\left(1 + k_1 r_d^2 + k_2 r_d^4\right)} \end{pmatrix} \tag{2.17}$$

with

$$r_u = r_d \left(1 + k_1 r_d^2 + k_2 r_d^4\right) \tag{2.18}$$

$$r_u = \sqrt{\left(d_u \left(\mathrm{u}_{\mathrm{u},i} - u_0\right)\right)^2 + \left(d_v \left(\mathrm{v}_{\mathrm{u},i} - v_0\right)\right)^2} \tag{2.19}$$

While r_u can be directly computed by (2.19), r_d can not be solved directly (see equation (2.18)). Thus r_d has to be solved numerically, for example by the Newton-Raphson method. The basic form for Newton-Raphson iteration is given by

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)} \tag{2.20}$$

where g denotes a function of x_n and g' its derivative with respect to x_n. In order to apply this method to approximate r_d function g and its first order derivative g' need to be defined as well as the initial value x_0. For this case equation (2.20) looks like this:

$$r_{dn+1} = r_{dn} - \frac{g(r_{dn})}{g'(r_{dn})} = r_{dn} - \frac{r_{dn} + k_1 r_{dn}^3 + k_2 r_{dn}^5 - r_u}{1 + k_1 r_{dn}^2 + k_2 r_{dn}^4}, \quad r_{d0} = r_u \tag{2.21}$$

This iteration is performed until $n > N$ for a fixed $N \in \mathbb{N}$.

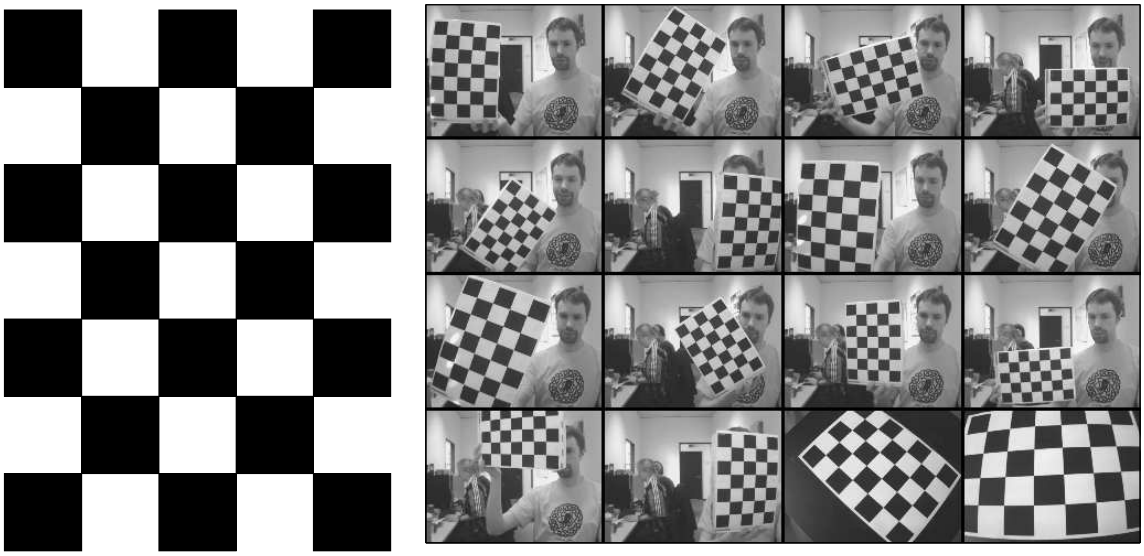
Of course other numerical methods could also obtain an approximation for r_d, but for this case Newton-Raphson is well suited, so that no further methods will be considered here.

2.3 Camera Calibration

This section will briefly describe how to obtain the parameters needed by the complete distortion model (see (2.11)) for a real camera, how to adapt these parameters to fit the distortion model used in [13,16,20] and show the results of camera calibration.

Since correctly calibrated cameras are a requirement for many applications there exists a large number of calibration tools, commercial or noncommercial. OPENCV provides its own method for camera calibration (see [7]). The OPENCV implementation is basically the `C` adaption of the Camera Calibration Toolbox for MATLABby Bouguet [6]. The intrinsic camera model for the calibration toolbox is similar to the model proposed by Heikkila [26], which largely corresponds to the "plumb bob model" of Brown [8]. The calibration process itself is inspired by [50].

To estimate the camera parameters using camera calibration a *calibration pattern* is needed. In the OPENCV implementation and the MATLAB toolbox the pattern consists of a flat chequered rectangle much like a chessboard. Commonly, but not necessarily, the black and white rectangles are squares and the whole pattern is rectangular (i.e. the number of columns differs from the number of rows) to better distinguish the orientation of the pattern. A typical pattern used for calibration is depicted in Figure 2.6a. Note that the single rectangles do not need to be black and white, but this coloring simplifies corner detection of the single rectangles, which is needed for the camera calibration. It is important that the pattern is really flat so it should be carefully attached to some rigid surface if printed out. Of course a real chessboard made of wood could also work perfectly. Furthermore the size of the black and white rectangles needs to be known and should be measured by hand, since a printer might scale its input and for a real chessboard the squares have to be measured anyway. After these preparations some images containing the calibration pattern have to be captured with the camera. Calibration will return better results, if the pattern is captured from different distances and at different locations on the image (i.e. center, left, right, top, etc...). Tilting the calibration pattern so that it can be observed from different angles and rotating the pattern can also improve calibration results, but in each image the whole pattern



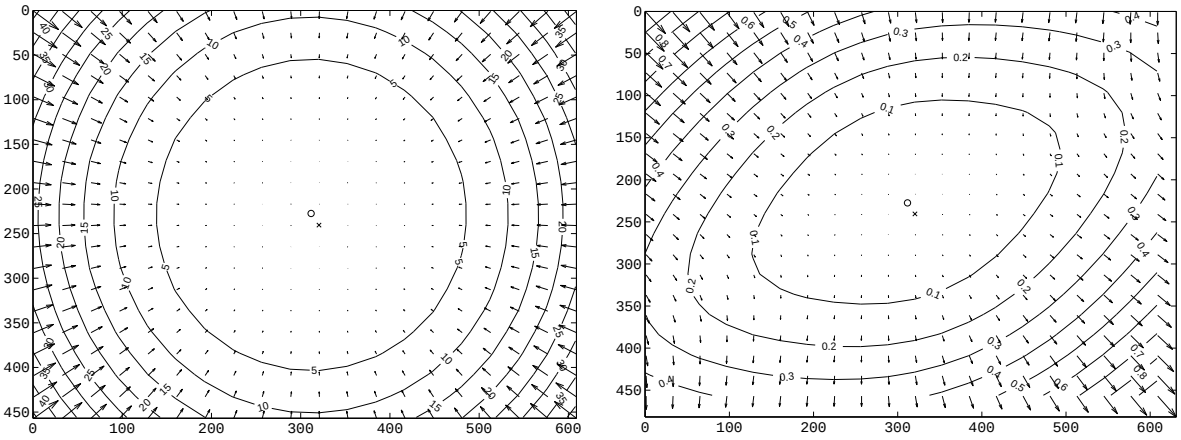
(a) Exemplary calibration pattern **(b)** Short calibration sequence

Figure 2.6: (a) Exemplary calibration pattern of size 4×6. Please note that the size is determined by the "inner corners" of the pattern. **(b)** Short calibration sequence. The calibration pattern of (a) is captured in different locations, orientations, at different distances and different angles. Please note that for visualization purposes only a part of a real calibration sequence is depicted.

has to be visible. Generally the more images are taken for camera calibration, the better, though 15–20 images usually provide quite acceptable results and even less than 15 images provide oftentimes noticeable improvements. An example for a short calibration sequence is provided in Figure 2.6b.

Once a sufficiently large sequence of suitable images for calibration is created, these images can be processed via OPENCV or MATLAB toolbox. The main advantage of the OPENCV implementation is automatic detection of all chessboard-corners.Especially with large calibration image sequences this becomes quite convenient. In the current version of the MATLAB toolbox the 4 corners defining the calibration pattern have to be marked by hand in every image. On the other hand the MATLAB toolbox provides much more options and information about the calibration. In cases of large distortion it is possible to adjust the corner detection region for single images, for example. It also offers a great visualization of the estimated distortion model (see Figure 2.7) or the estimated extrinsic parameters (that means the 3D positions of the calibration pattern in each image) can be shown. The influence of the tangential distortion (Figure 2.7b) compared to the complete distortion model (Figure 2.7a) is for the used HERCULES WebCam Classic near to non-existent, which justifies the simplified distortion model (see equation (2.13)) as opposed to the complete distortion model (equation(2.11)).

Image processing like feature detection (explained in 3.1) is done on the original distorted images in VISUAL MONO-SLAM and not on the undistorted images. The perceived image coordinates are then undistorted according to equation (2.13) stored in the VISUAL MONO-SLAM application and for prediction purposes distorted again by applying (2.17) (what is meant by "storing" and "prediction" will be explained in detail in chapter 4 so the reader should not worry about this now). This might seem a bit



(a) Complete distortion model **(b)** Tangential component of distortion model

Figure 2.7: (a) Influence of the complete distortion model (see equation (2.11)). The arrows show the displacement of the corresponding pixels induced by distortion. Notice that maximal distortion occurs at the top left and lower left corners, where points are displaced by more than 40 pixels. **(b)** The tangential component of the distortion model (equation (2.10)). Notice the maximal distortion occurs at the bottom right corner where points are displaced by more than 0.9 pixels.
Both results originate from calibrating a HERCULES WebCam Classic, providing images of 640×480 pixels. Calibration results and images obtained by MATLAB toolbox [6].

(a) Original image **(b)** Undistorted image

Figure 2.8: Example for image undistortion. **(a)** Original image captured by HERCULES WebCam Classic. Image distortion becomes visible near the edges of the image. The straight lines on the calibration pattern on the left and the corner between ceiling and wall on the top are appear curved due to radial distortion. **(b)** Undistorted image, using intrinsic camera parameters and distortion coefficients estimates provided by MATLAB toolbox [6]. Notice that calibration pattern does not appear distorted anymore, but the visible area is slightly smaller than in (a). The undistorted image was generated using OPENCV (see [7, 37]).

inconvenient at first glance and one might wonder why the undistorted images are not used. The main reason is that undistortion of a whole image takes quite a lot of time if the process time to perform all operations for one image is somewhere between 30 – 60 ms. Since the undistorted image is not composed as a direct mapping in terms of pixels. That means that usually a single pixel in the undistorted image is composed of a weighted sum of several pixels in the distorted image. Even though the weighted sums will stay the same, once the undistortion parameters are determined the whole process is still rather costly in terms of computation. Furthermore the weighted sums may induce a little blurring in some parts of the image which is usually not beneficial for the detection of repeatedly recognizable points in the image (as explained in 3.1). In addition the undistorted image will show less of the scene than the distorted image (see Figure 2.8). It is possible that some image region containing a stable feature may be cropped by the undistortion of the image.

2.4 Triangulation

Section 2.1 of this chapter introduced the pinhole model, which is used to emulate the projection properties of real cameras. In 2.2 a distortion model was introduced on top of the pinhole camera model so that radial distortion of real lenses can be compensated. However one fundamental problem is still unaddressed. Up until now the whole sensor information gathered by the camera is still 2 dimensional and does not provide any depth information about the projected 3 dimensional world. This becomes apparent if the basic projection equation of the pinhole model (2.2) is taken into account. Since the distortion model basically just describes, at which image coordinate $Q_{\mathrm{u},i} = (\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ the pixel at perceived image coordinate $Q_{\mathrm{d},i} = (\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ would be found on an ideal camera, a properly calibrated camera is not able to gather more information than the basic pinhole camera model.

In the following it is assumed that the camera is correctly calibrated and the each image coordinate $Q_i = (u_i, v_i)^T$ is not subject to any further distortion. To simplify things further, displacement of the image center R is neglected. In other words the basic pinhole model with the virtual image plane will be used (see equation (2.2)).

From the pinhole camera model a line on which an observed point P_i lies can be defined. If the projection $Q_i = (u_i, v_i)^T$ of P_i and the focal length f is known, P_i has to be somewhere on the line g:

$$g : \kappa \begin{pmatrix} u_i \\ v_i \\ f \end{pmatrix}, \kappa \in \mathbb{R}^+ \tag{2.22}$$

Remember that the optical center O is defined as the origin of the camera coordinate system. Therefore κ may be restricted to $\mathbb{R}^+$. However as long as no further information about κ is known, the above equation is not suited to determine the 3D position of P_i.

To gain 3D information from normal camera images, at least 2 images taken from a different positions are necessary. The technique introduced in the following is called *triangulation*. Assume two calibrated identical cameras observe the same scene. Both cameras are positioned in such a way that their pinhole planes are coplanar and their optical centers are apart by a known distance b, also referred to as *baseline* in *stereo vision* context. Also both cameras are orientated the same way (i.e. they look in the same direction). To distinguish the cameras they will be referred to as “left” and “right” camera and their corresponding variables will be marked with superscript l and r. Both cameras observe a point $P_i =$

$(x_i, y_i, z_i)^T$ which is projected at $Q_i^l = (u_i^l, v_i^l)^T$ and $Q_i^r = (u_i^r, v_i^r)^T$, as depicted in Figure 2.9. Without loss of generality it is also assumed that $v_i^l = v_i^r$ holds, but the equations below could be extended to the more general case with $v_i^l \neq v_i^r$. Observing the same point P_i in two different images allows for a depth estimation and subsequently for a estimation of the actual 3D position of P_i. According to the intercept theorem the following holds:

$$\frac{z_i - f}{z_i} = \frac{\overline{Q_i^l Q_i^r}}{b} = \frac{b - (u_i^l - u_i^r)}{b} \tag{2.23}$$

with $f = f^l = f^r$ (since both cameras are identical). From (2.23) follows directly

$$\frac{b - (u_i^l - u_i^r)}{z_i - f} = \frac{b}{z_i} \implies z_i = \frac{bf}{u_i^l - u_i^r} = \frac{bf}{\Delta u_i} \tag{2.24}$$

with $\Delta u_i = u_i^l - u_i^r$, baseline b. Distance d is also known as the *disparity*. For a visualization of (2.23) and (2.24) please refer to Figure 2.9. Equation (2.24) implies that the depth z_i of point P_i is inverse proportional to the disparity Δu_i. That means that if the disparity Δu_i is large, a small change in the disparity does not change the depth z_i much. However if Δu_i is near 0 a small disparity change evokes a large change in depth. This also implies that the depth resolution obtained by triangulation decreases the farther away an object is from the optical center of a camera. To illustrate this fact, please consider the following example: Assume two cameras with $b = 200\,\mathrm{mm}, f^l = f^r = f = 8\,\mathrm{mm}, d_u^\gamma = d_u = 8\,\mu\mathrm{m}, \gamma \in \{l, r\}$, where d_u^γ describes the physical size of one pixel in direction of the U-axis on the image sensor indicated by γ. Three different points, P_i, P_j and P_k are observed with their corresponding depth $z_i = 1\,\mathrm{m}, z_j = 10\,\mathrm{m}, z_k = 100\,\mathrm{m}$. Describing the depth z_i as a function h dependent from Δu_i according to (2.24) follows:

$$h(\Delta u_i) = z_i = \frac{bf}{\Delta u_i} \implies \Delta u_i = \frac{bf}{z_i} = \frac{1600}{1000}\,\mathrm{mm} = 1.6\,\mathrm{mm}$$

The smallest detectable change in the disparity corresponds with the physical pixel size d_u. This gives:

$$h(\Delta u_i + d_u) = \frac{bf}{\Delta u_i + d_u} = \frac{1600}{1.608}\,\mathrm{mm} \approx 995\,\mathrm{mm}$$

That means at a distance of $1\,\mathrm{m}$ the exemplary setup can distinguish between differences in depth of $5\,\mathrm{mm}$ or more. Analogously for P_j and P_k follows:

$$\begin{aligned}
\Delta u_j &= \frac{bf}{z_j} = \frac{1600}{10000}\,\mathrm{mm} = 0.16\,\mathrm{mm} \\
h(\Delta u_j + d_u) &= \frac{bf}{\Delta u_j + d_u} = \frac{1600}{0.168}\,\mathrm{mm} \approx 9523\,\mathrm{mm} = 9.523\,\mathrm{m} \\
\Delta u_k &= \frac{bf}{z_k} = \frac{1600}{100000}\,\mathrm{mm} = 0.016\,\mathrm{mm} \\
h(\Delta u_k + d_u) &= \frac{bf}{\Delta u_k + d_u} = \frac{1600}{0.024}\,\mathrm{mm} \approx 66666\,\mathrm{mm} = 66.666\,\mathrm{m}
\end{aligned}$$

Thus at a distance of $10\,\mathrm{m}$ the smallest difference in depth the exemplary setup is able to distinguish is $477\,\mathrm{mm} = 0.477\,\mathrm{m}$ and for far away objects like point P_k the depth resolution decreases dramatically to

$33.3\,\mathrm{m}$. This is an inherent problem of stereo vision. Though depth resolution for far away objects can be increased by increasing baseline b this does only help to a certain degree, since depth resolution does not decrease linearly (as shown above). Furthermore an increased baseline might inhibit depth estimation for close objects, since close objects might not be in the field of view of both cameras.

Triangulation can also be used to estimate x_i^γ and y_i^γ ($\gamma \in \{l, r\}$) of point P_i. If b, u_i^l and u_i^r are known the following equation holds:

$$\frac{x_i^\gamma}{z_i} = \frac{u_i^\gamma}{f} \implies x_i^\gamma = z_i \frac{u_i^\gamma}{f}, \quad \gamma \in \{l, r\} \tag{2.25}$$

Analogously y_i^γ can be estimated, if v_i^γ is known:

$$\frac{y_i^\gamma}{z_i} = \frac{v_i^\gamma}{f} \implies y_i^\gamma = z_i = \frac{v_i^\gamma}{f}, \quad \gamma \in \{l, r\} \tag{2.26}$$

Thinking of both cameras as one *stereo camera* gives a new optical center O_c at $\frac{\overline{O^l O^r}}{2}$. Since the camera coordinate systems of the "left" and "right" camera had the same orientation, they are simply substituted by one coordinate system with the same orientation and its origin in O_c. While equation (2.24) is not affected by this modification, (2.25) has to be slightly changed to

$$\frac{x_i + \frac{b}{2}}{z_i} = \frac{u_i^l}{f} \implies x_i = z_i \frac{u_i^l}{f} - \frac{b}{2}$$

Substituting z_i according to (2.24) gives

$$x_i = \frac{b f u_i^l}{f(u_i^l - u_i^r)} - \frac{b}{2} = \frac{b u_i^l}{u_i^l - u_i^r} - \frac{b}{2} = \frac{b}{2}\left(\frac{2u_i^l}{u_i^l - u_i^r} - \frac{u_i^l - u_i^r}{u_i^l - u_i^r}\right) = \frac{b}{2}\left(\frac{u_i^l + u_i^r}{\Delta u_i}\right) \tag{2.27}$$

$$y_i = z_i \frac{v_i^l}{f} = \frac{b f v_i^l}{f(u_i^l - u_i^r)} = \frac{b}{2}\frac{2v_i^l}{\Delta u_i} \tag{2.28}$$

Usually the camera setup is not as convenient as depicted in Figure 2.9. Even in off-the-shelf stereo cameras the pinhole planes are generally not perfectly coplanar and some other constraints might not be satisfied. In theory it should still be fairly easy to compute the depth z_i of point P_i observed by two cameras. As stated in equation (2.22) for each camera a line can be defined on which P_i has to lie. If the position of the optical centers of both cameras and their orientation in the world coordinate frame are known, point P_i should be at the intersection of lines g^l and g^r. Where g^l denotes the line from equation (2.22) of the "left" camera transformed in the world coordinate system and g^r the transformed line of the "right" camera. In reality however, chances for g^l and g^r to intersect a quite small, due to calibration errors and the discrete nature of the image sensor with its associated depth resolution. One approach to solve this problem constructs the line segment perpendicular to g^l and g^r, intersecting both. The center of this line segment is closest to both lines and therefore a meaningful estimation for P_i. Several other approaches exist as well, but are beyond the scope of this thesis. The interested reader is referred to [22] for more details.

Up until now a setup of two cameras (or one stereo camera) were considered. In a static environment the same methods can be applied with a single camera. Keep in mind that one requirement is that the

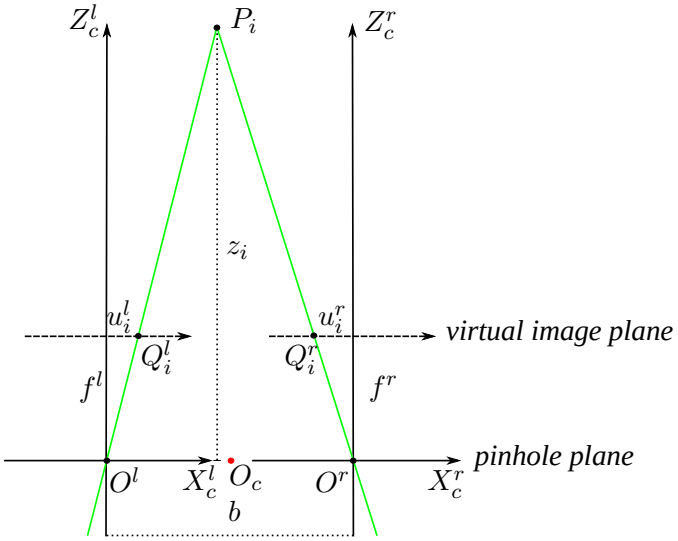


Figure 2.9: Triangulation from 2 cameras. Since the pinhole planes of both cameras are coplanar and their viewing direction is the same, in both camera coordinate systems the depth for point P_i is the same $z_i^l = z_i^r$ and therefore only labeled once as z_i in the image. Of course this does not hold necessarily for the other coordinates of P_i. In the depicted scene $x_i^l \neq x_i^r$ holds. The individual optical centers are denoted as O^l and O^r respectively. If both cameras are combined to a stereo camera the optical center O_c is in the middle of line $\overline{O^l O^r}$, marked in red. Please notice that the virtual image plane is depicted instead of the image plane to directly correspond with the used mathematical formulation.

position and orientation (or *pose*) of both cameras need to be known. Therefore in a static environment there is no difference between two cameras observing point P_i at the same time from given poses p^l and p^r or one camera observing P_i at two different times from poses p^l and p^r.

Above it was always assumed that for a 3 dimensional point $P_i = (x_i, y_i, z_i)^T$ the image coordinates $Q_i^l = (\mathrm{u}_i^l, \mathrm{v}_i^l)^T$ and $Q_i^r = (\mathrm{u}_i^r, \mathrm{v}_i^r)^T$ of its projection in two images I^l and I^r are known. In reality this is most often not the case. How to find a correspondence in image I^r to image coordinate $Q_i^l = (\mathrm{u}_i^l, \mathrm{v}_i^l)^T$ of image I^l will be shown in chapter 3.

Chapter 3

Image Processing

Chapter 2 dealt with camera models and how to estimate the 3D coordinates of a point $P_i = (x_i, y_i, z_i)^T$ observed in two images from two given different poses p^l, p^r. Up until now it was assumed that both projections $Q_i^l = (\mathrm{u}_i^l, \mathrm{v}_i^l)^T$ and $Q_i^r = (\mathrm{u}_i^r, \mathrm{v}_i^r)^T$ of point P_i on both image sensors are known. While that is true if we assume that P_i, p^l and p^r are known, the camera position and orientation and point positions are not known in the VISUAL MONO-SLAM-scenario, but need to be estimated. Establishing correspondences between two projections $Q_i^l = (\mathrm{u}_i^l, \mathrm{v}_i^l)^T$ and $Q_i^r = (\mathrm{u}_i^r, \mathrm{v}_i^r)^T$ of point P_i on the image sensor are crucial for depth estimation (see section 2.4) and therefore for the whole VISUAL MONO-SLAM approach.

This chapter will discuss how stable pairs of image coordinates Q_i^l and Q_i^r can be detected in two images observing the same scene from slightly different poses can be established. Furthermore some basic image processing techniques are introduced. First a brief overview of the used terminology and data structures in computer vision is given in the next paragraph.

The origin of an image coincides with the top left corner in computer vision, so that the top left pixel is at position $(0, 0)$. Accordingly if the image has a dimension of $\mathrm{width} \times \mathrm{height}$ the other corners are at positions $(\mathrm{width} - 1, 0)$, $(0, \mathrm{height} - 1)$ and $(\mathrm{width} - 1, \mathrm{height} - 1)$. Each pixel in turn is represented by an 8 bit value in case of gray-scale images, thus providing 256 different shades between black and white. Color images usually use three 8 bit values to represent one pixel. Their meaning depends on the used color model (HSV, YUV, CMY(K)...), among which the RGB color model is probably the most widely known and used. In the RGB model the first 8 bit value represents the amount and intensity of **r**ed for the current pixel, while the other two values represent **g**reen and **b**lue. For more information on color models, their advantages, disadvantages and methods of conversion, please refer to [22]. In the following only gray-scale images are considered, if not explicitly stated otherwise. This is in accordance with most image processing techniques, since gray-scale images are less noisy than color images and usually provide sufficient information about the environment.

Firstly it should be noted that in image processing almost never the complete raw data of an image is used. This has two main reasons. The first and more important reason is that raw image data tends to be relatively unstable and is as such quite infeasible for comparison between two images. For example in theory motion in a sequence of images taken from the same pose could be detected by just comparing each pixel with the pixel at the same position in the subsequent image. If the pixel in the subsequent image has not the same gray-scale value (or values in case of color images) one could assume that some

(a) **(b)** **(c)** difference of (a) and (b)

Figure 3.1: Image sensitivity to lighting conditions. In subfigures **(a)** and **(b)** are two images taken subsequently by HERCULES WebCam Classic. Though they appear the same, due to slight changes in lightning conditions they actually differ. **(c)** depicts the difference as binary images. Pixels in white indicate a difference between the pixels of (a) and (b) larger than a given threshold, while the difference of black pixels is below the threshold. Used thresholds are 1 (85.7% white pixel), 4 (35.9%), 7 (2.6%) and 10 (0.5%) from top left to bottom right.

movement in this area has occurred and thus a different object with a different gray-scale-value (RGB-value) is observed at this pixel. In an ideal environment this naive approach might work, in reality there will be too much noise in the images for this idea to be feasible. Cameras tend to be extremely sensitive to changes in lighting conditions (see Figure 3.1). Especially if cameras are used outdoors lightning conditions will change and vary constantly.

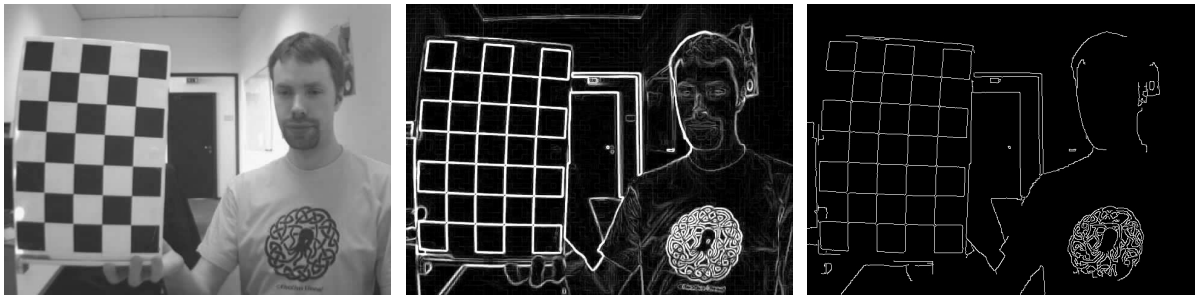
The second reason is just the amount of data which would be impractical in many applications. Consider that a common low-cost camera like the HERCULES WebCam Classic provides a resolution of $640 \times 480 = 307200$ pixels per image. Dependent on the frame rate of the camera 30 – 60 ms can be used to establish correspondences, estimate 3D positions for detected correspondences and what other operations might be necessary in an online application. That would (in a very naive implementation) leave up to $0.1953\,\mu$s to try to find a suitable correspondence in the remaining 307199 pixels for each single pixel, not taking into account that usually other computations have to take place as well.

Furthermore a single pixel does not contain much information, while an aggregation of adjacent pixels may indeed contain more information than its individual parts. Such an aggregation is commonly referred to as *feature*. Different feasible techniques exist how to select pixels to create such a feature, but mainly the common goal is to create more robust measures of comparison between different images or to establish correspondences between positions $Q_i^l = (\mathrm{u}_i^l, \mathrm{v}_i^l)^T$ and $Q_i^r = (\mathrm{u}_i^r, \mathrm{v}_i^r)^T$ that belong to the same 3D point P_i, observed in images I^l and I^r.

Section 3.1 will introduce some approaches to detect features in a given image and discuss their advantages and disadvantages. Afterwards section 3.2 will briefly discuss some general techniques related to image processing.

3.1 Image Features

As stated above robustness and repeatability are two major goals for features detected in an image. This section is divided into two further subsection, which will present two different approaches to tackle this problem. In subsection 3.1.1 early approaches will be presented, generally called corner or interest point



(a) Original Image **(b)** Sobel Filter **(c)** Canny Edge Detection

Figure 3.2: Edge Detection: To the original image **(a)** two different edge detectors were applied. **(b)** shows the result of the so called Sobel filter , while **(c)** depicts the result of the canny edge detector [10]. Notice that the edges in (b) are much thicker than in (c), which might not always be desired. Output however depends in both techniques heavily on the used parameters.

detectors / operators. Though not as powerful in themselves as later approaches presented in subsection 3.1.2, they still have their appeal nowadays, mainly because of their fast and easy computation.

Furthermore as a preface before the details of implementation are introduced, a short definition of terms is needed. In the literature the terms *corner*, *interest point* and *feature* are used somewhat ambiguously. In this thesis corner and interest point will both refer to a specific position in the image and the pixel present at this location. How such an interest point differs from normal pixels will be explained in 3.1.1. A feature is defined as an interest point with the addition of some comparison measure.

3.1.1 Corner Detectors

To achieve the robustness a common technique is to detect image regions with a high gradient (i.e. corresponding to visual edges). Several approaches for edge detection exist, among which the *Sobel operator* and *Canny edge detector* [10] rank among the most popular and their results are depicted in Figure 3.2.

Edges detected in one image usually have a good chance to be detected in subsequent images. However to establish correspondences for single 3D points needed for triangulation (see section 2.4) edges are not suitable. While edges show a large gradient in the image data perpendicular to the direction of the edge, responses to image filters moving along the direction of the edge are often very similar. Therefore it becomes difficult to establish pairs of image positions Q_i^l and Q_i^r, even if corresponding edges in two images are detected, as long as the camera movement is unknown. This is also known as the *aperture problem*.

To improve repeated detection and thus establish pairs of positions belonging to the projection of the same 3D point it was proposed to use corners instead of edges. Usually one would expect a corner to be at the intersection of two edges, but most so-called corner detectors actually find also edge endings, local intensity maxima or minima or points at local maximal curvature along a curve. Please keep this in mind if the term corner or corner detector is used in the following.

One of the fist algorithms to detect such interest points was proposed by Moravec [34]. Moravec defined a corner as an image point with low self-similarity. This is tested for each pixel in an image by

comparing a small squared image patch around the pixel with the image patches of adjacent points in vertical, horizontal and two diagonal directions. For patch comparison the *sum of squared distances* or SSD for short is calculated. The minimum of the calculated SSDs will be the interest measure for the current pixel and local maxima in the interest measure indicate an interest point at the current pixel. For uniform areas the interest measure will be close to zero, since the SSD should not differ significantly. If an edge is present in the direction perpendicular to the edge the SSD will be great, but in direction of the edge it should be small. This way edges should mostly be rejected, since the minimum of the calculated SSDs is used as interest measure. The main weakness of this detector is possible false classification of edges as interest points. Moravec proposed angles of $45°$between the directions used for SSD computation. So edges with angles of odd multiples of approximately $22.5°$might be detected as corners.

To create an isotropic detector Harris and Stephens [24] improved on the main idea presented in [34]. Their detector is widely known as the "Harris corner" operator / detector in computer vision literature. In order to make their detector isotropic, the weighted sum of squared distances (SSD_w) is used to compare image patches. The SSD_w over an image patch p with dimensions $(p_w + 1 \times p_h + 1)$ compared to a patch of the same size shifted by (x, y) in direction of the U and V axes of the image can be defined as:

$$\mathsf{SSD}_w(x,y) = \sum_{u=0}^{p_w}\sum_{v=0}^{p_h} w(u,v)\left(I(u,v) - I((u+x, v+y)\right)^2 \tag{3.1}$$

where $I(u,v) \in \mathbb{Z}^+ \leq 255$ denotes the gray-scale value of at image position (u,v) and $w(u,v)$ the corresponding weight. The term $I(u+x, v+y)$ of equation (3.1) can be approximated by Tailor expansion as

$$I(u+x, v+y) \approx I(u,v) + x\frac{\partial I}{\partial u} + y\frac{\partial I}{\partial v}$$

where $\frac{\partial I}{\partial u}$ and $\frac{\partial I}{\partial v}$ denote the partial derivatives of $I(u,v)$ in direction of the U and V axes of the image. The partial derivatives in turn are easily computed by

$$\frac{\partial I}{\partial u} = I \otimes \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$\frac{\partial I}{\partial v} = I \otimes \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{3.3}$$

where $I \otimes k$ indicates the convolution of the image with the specified kernel k. These approximations allow for (3.1) to be written as

$$\begin{aligned} \mathsf{SSD}_w(x,y) &\approx \sum_{u=0}^{p_w}\sum_{v=0}^{p_h} w(u,v)\left(x\frac{\partial I}{\partial u} + y\frac{\partial I}{\partial u}\right)^2 \\ &= \begin{pmatrix} x & y \end{pmatrix} \sum_{u=0}^{p_w}\sum_{v=0}^{p_h} w(u,v) \begin{bmatrix} \frac{\partial I}{\partial u}^2 & \frac{\partial I}{\partial u}\frac{\partial I}{\partial v} \\ \frac{\partial I}{\partial u}\frac{\partial I}{\partial v} & \frac{\partial I}{\partial v}^2 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ &= \begin{pmatrix} x & y \end{pmatrix} H \begin{pmatrix} x \\ y \end{pmatrix} \end{aligned} \tag{3.4}$$

where H is called "Harris" matrix. To actually achieve an isotropic detector, the weighting function $w(u, v)$ should be smooth and circular around the center of image patch p, like the Gaussian function:

$$w(u, v) = \exp\left(\frac{-\left(\left(u - \frac{p_w}{2}\right)^2 + \left(v - \frac{p_h}{2}\right)^2\right)}{2\sigma^2}\right)$$

where p_w and p_h denote width and height of patch p. To determine whether a corner is present at the center of the selected patch p, or an edge no distinguishable element is present the Harris matrix is examined more closely. Dependent on its *eigenvalues* λ_1 and λ_2 three different cases need to be considered:

- If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$, no interest point is present
- If $\lambda_1 \approx 0$ and $\lambda_2 \gg 0$, an edge is detected
- If $\lambda_1 \gg 0$ and $\lambda_2 \gg 0$, a corner is detected

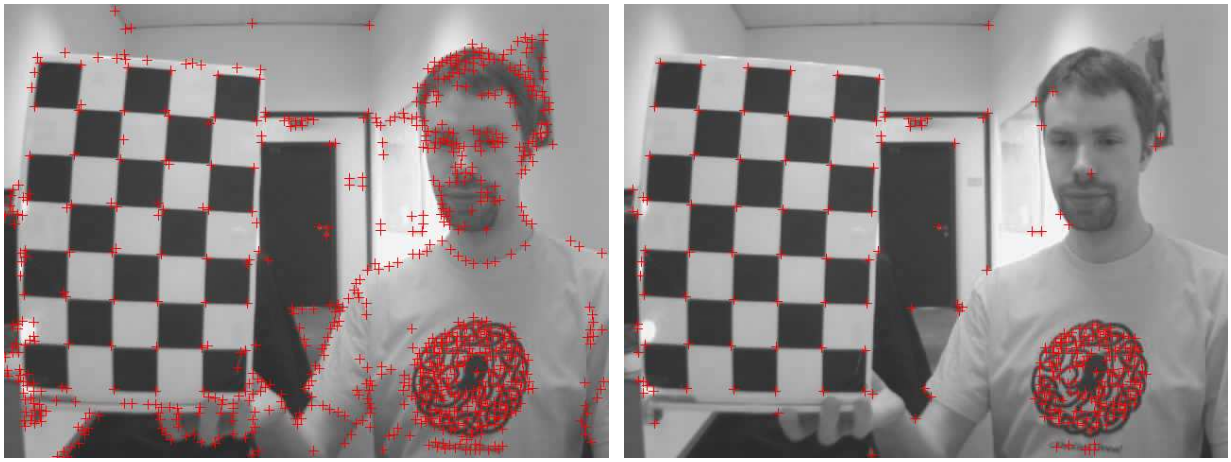
To avoid computation of eigenvalues in [24] the following measure for the *interest response* R is proposed:

$$R = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = \det(H) - \kappa \operatorname{trace}(H)^2 \tag{3.5}$$

where parameter κ is a constant that has to be determined empirically. For an interest response $R \gg 0$ a corner at the center of the patch may be assumed, $R \ll 0$ corresponds to an edge and $R \approx 0$ implies no distinguishable image region. Additionally only those points, where the interest response forms a local maximum are actually considered to be corners. Dependent on the application these points can be further filtered by applying a minimum interest response $R_{min} \in \mathbb{R}^+$ as an additional threshold. Results of the Harris operator depend heavily on the given parameters of κ, R_{min} and the size of the compared image patches, thus finding the right parameters for a specific application might take some fine tuning. OPENCV provides a ready- The default OPENCV settings are image patches of size 3×3 and $\kappa = 0.04$, but can be changed to adapt to the needs of different application scenarios. Exemplary results of the Harris operator are depicted in Figure 3.3.

Closely related to the Harris operator is the detector presented by Shi and Tomasi in [42]. The results of this detector are sometimes referred to as "Good Features to Track", named after the title of their first publication. Shi an Tomasi showed in their experiments that $\min(\lambda_1, \lambda_2)$ though more costly to compute provides a better measure for corner strength than interest response R (see (3.5)). Finding the appropriate parameters for the Shi Tomasi detector is crucial and might be somewhat tedious just like in case of the Harris detector. On the bright sight the Shi Tomasi detector has just 2 parameters to adapt, namely response threshold and patch size. An implementation of the algorithm can also be found in OPENCV. In [13, 16, 20] Davison et al. use this interest operator to find suitable candidates for features. An exemplary application of the Shi Tomasi detector on an image is depicted in Figure 3.4

The corner detectors discussed up until now are able to detect interest points in an image which can be recovered in a similar image with high probability. However means of comparison between the interest points of two different images are needed, since the interest response itself does not contain enough information to ensure an appropriate pairing. A common approach is to compare image patches with the



(a) low threshold **(b)** high threshold

Figure 3.3: Harris Corner Detector: **(a)** shows the result of the Harris detector for image patches of size 3×3, $\kappa = 0.04$ and a low threshold (dependent on the strongest interest response for the image). **(b)** depicts the results using a high threshold, while leaving κ and the patch size unchanged.

interest point at its center from the different images. Several comparison techniques of image patches are discussed in detail in 3.2.2. The interest points together with their associated image patches and an appropriate comparison mechanism compose a feature, according to the definition in the beginning of section3.1. Still these interest point operators suffer from some limitations. Namely they are not *scale invariant* or *rotational invariant*. Scale invariance means that an interest point is detected in images, depicting the same scene from different distances, which results in projections at different scales on the image sensor. The lack of this ability is inherent in the simple computation of the detectors described in [24, 34, 42]: Observed from a different distance (i.e. in a different scale) interest points may vanish if several pixels are combined to a single pixel (zooming out) or a previously single pixel might be represented by a number of pixels of nearly identical intensity (zooming in). A common technique to achieve scale invariance in image processing is the so called *image pyramid* of images where one image is scaled to different resolutions, thus simulating zooming in and out of a scene. However this is more of post-processing step, possibly including accumulation of several interest points detected at different scale into some combined information and would diminish the computational fast properties of the corner detectors. Details to the concept of image pyramids can be found in [1].

Rotational invariance in turn means that an interest point should be detected and successfully matched if the camera is rotated around its Z_c-axis. While the detection of interest points in a rotated view does not pose a problem to the Harris operator and the Shi Tomasi detector, since they are isotropic, establishing correspondences to interest points of an image under a different rotation poses a problem. If the rotation is not known and large, the comparison of two image patches (see section 3.2.2) will most likely fail.

In the following subsection some feature descriptors are introduced which gained popularity in recent years. Though computational costly, they are scale and rotational invariant, robust and provide their own means of comparison.

(a) low threshold **(b)** high threshold

Figure 3.4: Shi Tomasi detector: **(a)** depicts the results of the Shi Tomasi detector using a low threshold based on the strongest response in the image, while in **(b)** a high threshold is used. In both cases the size of the image patch is 3×3 pixels. Notice that the results of this detector are very similar to the Harris detector depicted in Figure 3.3, albeit not exactly the same.

3.1.2 Feature Descriptors

The algorithms introduced in this subsection detect interest points and compute for all suitable feature point candidates a *descriptor*. The descriptor is basically a vector which stores all the information needed to compare one feature with another one. The descriptor does not contain raw image data like an image patch, but higher level information calculated from the underlying image data and transformed in such a way that the contained information becomes scale and rotational invariant. The computations for a feature descriptors is, compared to the computations needed for corner detectors (see subsection 3.1.1) and retrieving an image patch surrounding the corner, quite costly, so that the calculation of the features presented in the following will be slower compared to corner detectors. Still many computer vision applications employ feature descriptors, especially in the domain of object recognition. Typically in this application scenario multiple descriptors are used to describe a single object. For real-time applications with high frame rates (30 – 60 ms) where computational speed becomes crucial usually simpler features based on corner detectors are used.

In the following the algorithms for two feature descriptors are sketched. Firstly the “Scale Invariant Feature Transform” (SIFT) which was introduced by Lowe [30–32] and can be considered as one of the most successful and widely used feature descriptors today. Secondly the “Speeded Up Robust Features” (SURF) by Bay et al. [4] which refined some ideas presented in [30–32]. Both algorithms will be introduced briefly only, since a detailed discussion would be beyond the scope of this thesis.

SIFT Scale invariant interest points for SIFT-features are found found by the calculation of *Difference of Gaussians* (DoG) over an image pyramid (see [1]). The DoG is defined as

$$\begin{aligned} D(u,v,\sigma) &= L(u,v,k\sigma) - L(u,v,\sigma) \\ &= (G(u,v,k\sigma) - G(u,v,\sigma)) \otimes I(u,v) \end{aligned} \tag{3.6}$$

with $L(u,v,\sigma) = G(u,v,\sigma) \otimes I(u,v)$ where $\otimes$ denotes the convolution of image $I(u,v)$ and Gaussian kernel $G(u,v,\sigma)$ defined as

$$G(u,v,\sigma) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-(u^2+v^2)}{2\sigma^2}\right)$$

To efficiently calculate (3.6) the image $I(u,v)$ is repeatedly convolved with the same Gaussian kernel $G(u,v,\sigma)$, until σ is doubled. The convolved images are stored and the DoG is computed by simply subtracting each convolved image from its predecessor. Once σ is doubled the last convolved image is rescaled to half its original size and the process begins anew (with much less computational cost). Please note that instead of rescaling the image, convolving the image further would yield the same result, but the convolution of a smaller image is more efficient in terms of computation. All images until σ is doubled (i.e. images of the same size) compose an *octave* in the terminology of [30–32], while the convolved images in the same octave are in different *scale*. Potential interest points are at local minimal or maximal in the DoGs. These are detected by comparing each pixel with its 8 neighbors on the same scale and the 9 neighbors on the scale above and below. A schematic of this algorithm is depicted in Figure 3.5.

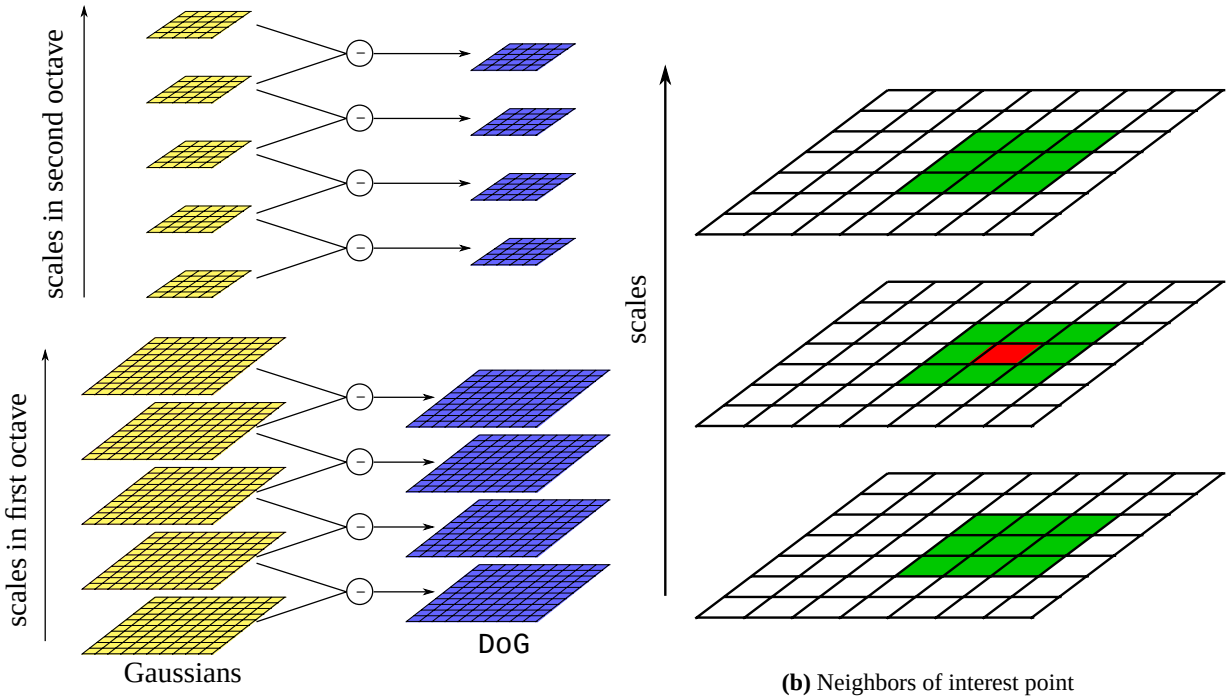
Please note that the DoG is a computational efficient approximation for the Laplacian of Gaussians. In [29] Lindeberg showed that the Gaussian function is the only suitable kernel for scale-space. Mathematical details why the DoG closely approximates the Laplacian of Gaussians can be found in [32].

After obtaining the potential key points, these are fitted with sub-pixel accuracy to their corresponding local extrema (i.e. the key point will usually not correspond to a single pixel, but refer to a position u_{sub}, v_{sub} with $u_{sub}, v_{sub} \in \mathbb{R}^+$ with $0 \leq u_{sub} < \text{width}$ and $0 \leq v_{sub} < \text{height}$, where width and height specify the dimensions of the image). How this is done, is explained in detail in [9]. After the correct position of the interest point is determined the function values at this point is compared with a threshold, rejecting low function values (i.e. local minima and maxima with a small absolute value) to ensure stable interest points.

Since the DoG also has strong responses along edges and edges make for poor localization due to aperture the remaining points are filtered once more. Similar to edges in 3.1.1 where the interest measure alongside the edge was small, the principal curvature along an actual edge would be much smaller than perpendicular to the edge. The principal curvature can be determined by obtaining the eigenvalues of the *Hessian* $\mathcal{H}$ of the DoGs defined as

$$\mathcal{H} = \begin{bmatrix} D_{uu} & D_{uv} \\ D_{uv} & D_{vv} \end{bmatrix}$$

where D denotes the DoG (see 3.6) and D_{uu}, D_{uv}, D_{vv} are the partial derivatives along the U- and V-axes. Partial derivatives are calculated by convolution of the image according to equations (3.2) and (3.3). Similar to the Harris detector [24] (see equation (3.5)) the actual computation of the eigenvalues can be avoided, since just their ratio is of importance. If the ratio of both eigenvalues differs strongly



(a) Octaves and scales in SIFT

(b) Neighbors of interest point

Figure 3.5: Image pyramid of SIFT-algorithm. **(a)** depicts the construction of the DoGs. To get from one scale image to the next scale (shown in yellow), the current scale is convoluted with $G(u, v, \sigma)$. The stored scales are used to calculate the DoGs. In **(b)** the neighbors for a potential interest point are shown. The interest point is displayed in red, while its neighbors are colored in green. Both figures (a) and (b) are based on figures shown in [32].

from 1 the interest point can be assumed to lie on an edge. To obtain the ratio r of the eigenvalues λ_1, λ_2 the following method is proposed. Without loss of generality it can be assumed that $\lambda_1 \geq \lambda_2$ with $\lambda_1 = r\lambda_2, r \in \mathbb{R}^+, 1 \leq r$. This assumption yields

$$\frac{\operatorname{trace}(\mathcal{H})^2}{\det(\mathcal{H})} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} = \frac{(r\lambda_2 + \lambda_2)^2}{r\lambda_2^2} = \frac{(r+1)^2}{r}$$

The term $\frac{(r+1)^2}{r}$ has its minimum for equal eigenvalues (i.e. $r = 1$) and increases for other valid values of r. In [32] a use of $r = 10$ is suggested. That means when the ratio between the eigenvalues becomes grater than 10 the interest point is assumed on an edge and subsequently discarded.

The remaining interest points will be assigned a orientation in order to ensure rotational invariance. This way the information contained in the actual descriptor (see below) can be represented in relation to the assigned orientation. Orientation for an interest point will be determined by calculation of a gradient histogram in a region around the interest point at its corresponding scale $D(u, v, k\sigma)$. According to their direction the gradients are arranged in the different bins of the histogram and weighted with their magnitude. Thus a gradient of great magnitude becomes more influential than a gradient of small magnitude. If resulting histogram does not clearly indicate one direction, an interest point may also be assigned more

than one orientation. For details of this process please refer to [32].

After suitable interest point locations were found by the previous steps, made scale and rotational invariant, the actual measure of feature comparison needs to be constructed, namely the feature descriptor. The information contained in the descriptor should make the feature more robust, taking into account different lighting conditions for example. To calculate the descriptor a vector is constructed by calculating gradients in a neighborhood around the interest point and weighting the gradients according to their magnitude and a Gaussian with the interest point as its mean. All in all 4×4 histograms are computed around the interest point location, each histogram containing 8 bins. The resulting information is written into a vector of size $4 \times 4 \times 8 = 128$. To make the feature descriptor robust to changing illumination conditions, the vector is normalized (further details in [32]).

SIFT-features are matched by nearest neighbor search: The descriptor of a feature in the first image is compared to all descriptors found in the second image (or in case of object recognition with descriptors in a database containing the information for specific objects). Distance between two descriptors is measured by the Euclidean distance. The closest distance d_1 between two descriptors is compared to the second closest distance d_2. If the ratio $\frac{d_1}{d_2}$ is above a certain threshold (in [32] a threshold of 0.8 is suggested) the match is discarded due to ambiguity, in all other cases a correspondence between the nearest neighbors is assumed. Please note that SIFT does approximate the nearest neighbor search by the so called *Best-Bin-First* algorithm which uses *kd*-trees. This especially becomes important for object recognition tasks with huge numbers of features, since a complete naive nearest neighbor search would be very time consuming.

SURF SURF-features, short for "Speeded Up Robust Features", introduced by Bay et al. in [4] seek to reduce computational effort, compared to SIFT, while still maintaining a high rate of correct matching. In the implementation of SURF-features another method to find initial interest points is used. Instead of the Laplacian of Gaussians (which was approximated by the DoG for SIFT-features) the SURF detector is based on the determinant of the Hessian Matrix of Gaussians reported in [29]. The Hessian detector is defined as $\det\left(\mathcal{H}\left(u,v,\sigma\right)\right)$ with

$$\mathcal{H}\left(u,v,\sigma\right) = \begin{bmatrix} L_{uu}\left(u,v,\sigma\right) & L_{uv}\left(u,v,\sigma\right) \\ L_{uv}\left(u,v,\sigma\right) & L_{vv}\left(u,v,\sigma\right) \end{bmatrix} \tag{3.7}$$

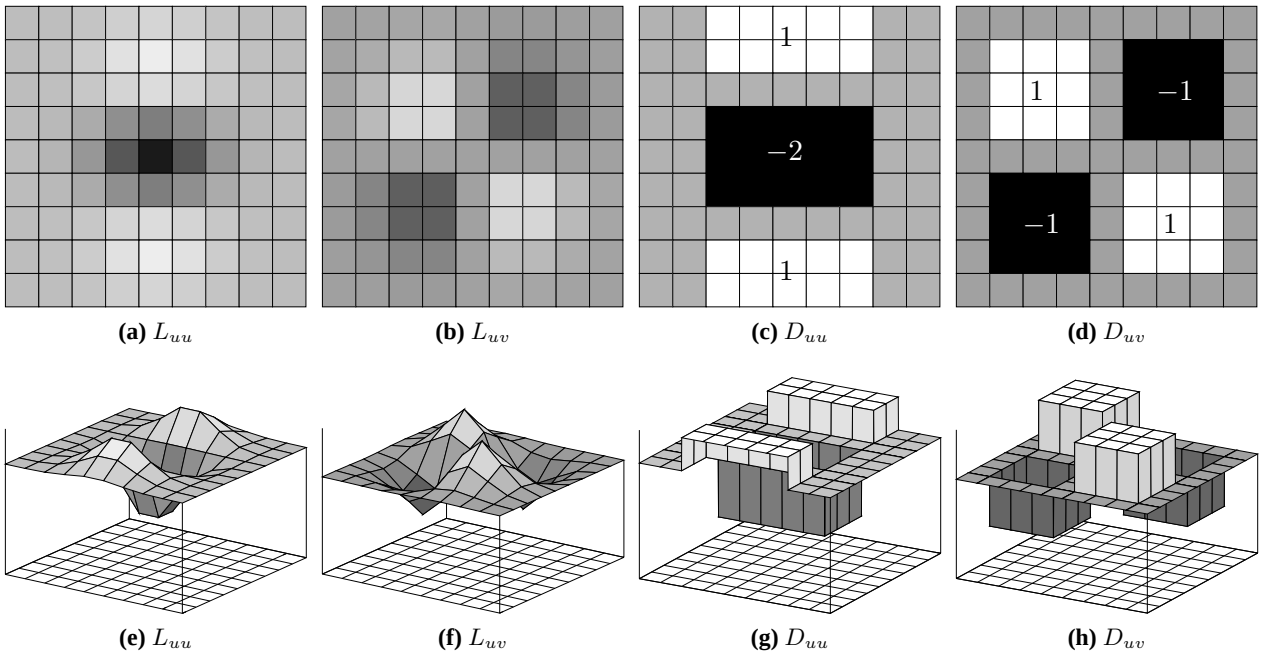
where $L_{uu}\left(u,v,\sigma\right) = \frac{\partial^2 L(u,v,\sigma)}{\partial u^2}$ denotes the second order derivative of the Gaussian convolution of the image (see 3.6). The Gaussian kernel is therefore convolved twice according to equation (3.2) before being applied to the image. L_{uv} and L_{vv} are defined analogously. The Gaussian convolution is in practice accomplished by using a discretized and cropped kernel of fixed size. In [4] L_{uu}, L_{uv} and L_{vv} are roughly approximated by box filters D_{uu}, D_{uv}, D_{vv} (see Figure 3.6).

Since the approximation with box filters is rather coarse just exchanging L_{uu}, L_{uv} and L_{vv} in equation (3.7) will yield different results from the original Hessian detector. To better emulate the Hessian detector Bay et al. propose

$$\det\left(\mathcal{H}\left(u,v,\sigma\right)\right) \approx D_{uu}D_{vv} - \left(0.9D_{uv}\right)^2 \tag{3.8}$$

This approximation is justified by observing

$$\frac{\|L_{uv}\|_{\mathrm{F}}\|D_{uu}\|_{\mathrm{F}}}{\|L_{uu}\|_{\mathrm{F}}\|D_{uv}\|_{\mathrm{F}}} = 0.912\ldots \approx 0.9 \tag{3.9}$$



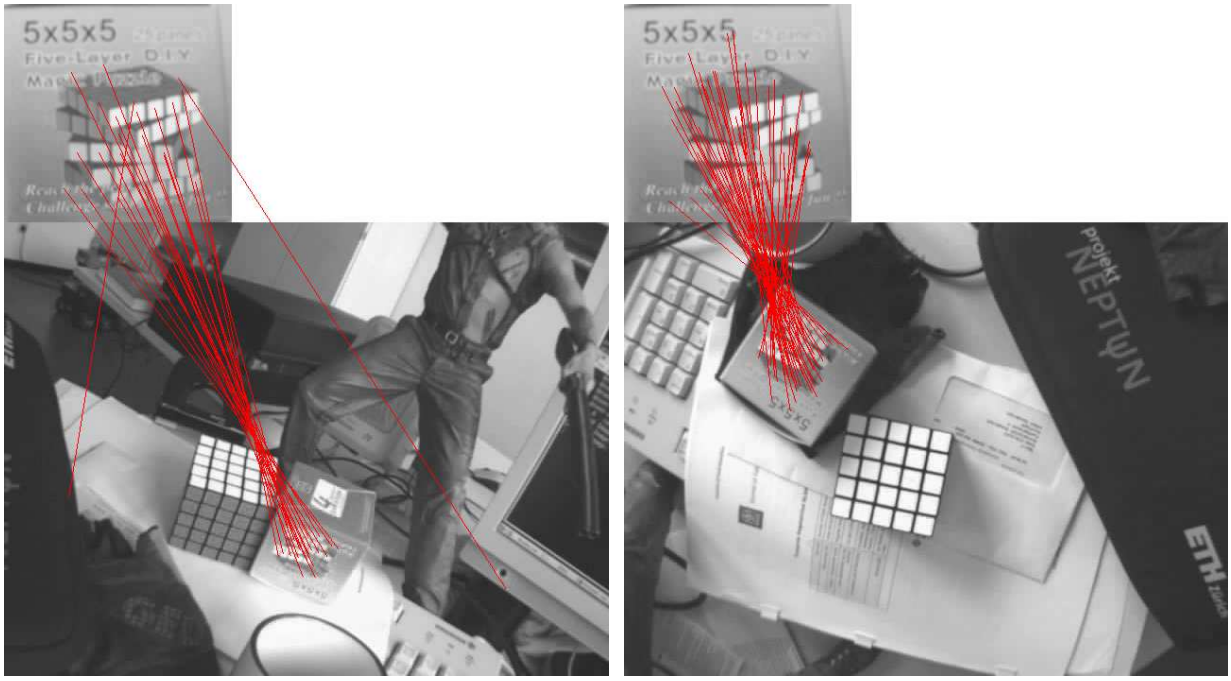
(a) L_{uu} **(b)** L_{uv} **(c)** D_{uu} **(d)** D_{uv}

(e) L_{uu} **(f)** L_{uv} **(g)** D_{uu} **(h)** D_{uv}

Figure 3.6: Gaussian Kernels and Box Filters. **(a)** shows a discretized kernel for $L_{uu}\,(u, v, \sigma)$ (see equation (3.7)), with $\sigma = 1.2$. Below in **(e)** the 3D plot of the kernel is depicted. Please note that $L_{vv}\,(u, v, \sigma)$ can be achieved by rotating (a) by 90°. In **(b)** the kernel for L_{uv} is shown and its corresponding 3D plot is depicted in **(f)**. **(c)** and **(g)** picture the box filter D_{uu} approximating to L_{uu}. Accordingly box filter D_{uv} displayed in **(d)** and **(h)** corresponds to (b) and (f). Note the different scales of the Gaussian kernels (e) and (f) compared to their box filter approximations (g) and (h) which results in the adaption of the Hessian detector in equation (3.8).

where $\|\ \|_{\mathrm{F}}$ denotes the Frobenius norm, L_{uu}, L_{uv} the second order of a Gaussian convolution with standard deviation $\sigma = 1.2$ and D_{uu}, D_{uv} the corresponding box filters of size 9×9 (see Figure 3.6). Equation (3.9) yields the the ratio of the resultant convolution of box filter approximations D_{uu} to D_{uv} compared with the ratio of the convolution of L_{uu} and L_{uv}. This in turn gives a valid measure how the determinant has to be modified, leading to equation (3.8). A variance of $\sigma = 1.2$ and box filter of size 9×9 are the initial values in the original SURF implementation by Bay et al. to detect scale invariant interest points.

Before the box filters are applied, the *integral image* (see subsection 3.2.1) of the input image is created. Integral images allow for a fast alternative to the image pyramid: Instead of iteratively applying a filter to the output of the previous layer and sub-sampling the image for the next octave the filter size is increased. This has basically the same effect, but is usually not done, because increased filter sizes usually greatly increase the needed computations. However since box filters are applied to an integral image, the computational effort to calculate the result is independent of the filter size. And since all filters are applied to the same integral image the whole process could easily be done in parallel, taking advantage of multi-processor platforms. Similar to SIFT, potential interest points are detected as local extrema in a neighborhood on their and the adjacent scales. Afterwards the location of the interest points is calculated to sub-pixel accuracy, also using the approach of [9].



(a) **(b)**

Figure 3.7: Object recognition with SURF-features in cluttered desk environment. The object is shown in the upper left corner of **(a)** and **(b)**. Estimated correspondences are indicated by red lines. Notice that both cases (a) and (b) contain false matchings (though more obvious in (a)). However the majority of features are matched correctly, therefore object position and orientation in the image can be estimated.

Next the orientation of the feature will be estimated. Please note that there also exists a version of SURF, called U-SURF where the step is omitted and orientation is assumed to be upright. Since the camera will not heavily rotate around its own Z_c-axis in many application scenarios this is a sensible measure to reduce computational effort in these cases. Instead of using histograms of gradients like SIFT, SURF determines the orientation of a feature by calculating several Haar-wavelet responses in U and V direction at different scales. Since Haar-wavelets are box filters, the already constructed integral image can be reused. The descriptor itself is also constructed of wavelet responses over a squared area, centered at the interest point and rotated accordingly to the previously determined orientation. In order to speed up the comparison process of features SURF-feature use a descriptor of size 64 (instead of SIFT's 128), which increases comparison speed. The actual comparison works just like for SIFT-features. Exemplary object matching, using SURF-features is depicted in Figure 3.7.

An interesting comparison between SIFT and SURF can be found in [48], where features in outdoor images taken during different seasons and thus very different lighting conditions are compared.

3.2 Basic Image Processing Techniques

Many applications in computer vision benefit from high frame rates. VISUAL MONO-SLAM makes no difference and update frequencies significantly below $\sim 60\,\mathrm{Hz}$ will seriously hamper the performance of the algorithm. High frame rates of course imply that all needed computations for one image need to be done before the next image is retrieved, which leaves a time window of 30 – 60 ms per image, dependent on the frame rate of the used camera. In this time window existing features need to be matched, eventually new features need to be acquired, all computations associated with the *Extended Kalman filter* (see section 4.1) need to be completed and results need to be visualized. On this account it is important to speed up computations as much as possible, even though this implies some inconvenience on the implementation of algorithms, since naive solutions which are easy to program and understand tend to perform rather slow.

To provide a short example for this remark consider the representation of image data. Since images are 2 dimensional it seems convenient and natural to represent an 8 bit gray-scale image of size $\mathrm{width} \times \mathrm{height}$ as a two dimensional array $\texttt{img}\,[\mathrm{width}]\,[\mathrm{height}]$ of `char` or any other 8 bit data type. This way the pixel at position $(\mathrm{u}, \mathrm{v})^T$ could simply be addressed by using $\texttt{img}\,[\mathrm{u}]\,[\mathrm{v}]$. During image processing oftentimes data needs to be temporarily copied, allocated, freed and addressed. Since all the above mentioned operations perform much faster on one dimensional arrays, usually images are internally represented as $\texttt{img}\,[\mathrm{width} \times \mathrm{height}]$. That means of course that addressing a pixel $(\mathrm{u}, \mathrm{v})^T$ becomes less convenient ($\texttt{img}\,[\mathrm{width} \cdot \mathrm{v} + \mathrm{u}]$).

In subsection 3.2.1 the concept of integral images will be presented, which are used in the SURF algorithm and prove also beneficial for other purposes. Thereafter the topic of image patch matching is discussed in subsection 3.2.2 which will also serve to exemplary illustrate the importance of efficient computation in image processing.

3.2.1 Integral Images

Integral images were first introduced in 1984 by Crow [15]. Each pixel in an integral image `Int` contains the sum of the intensity of all pixels to the left and above of this pixel in the original image and the intensity of the pixel itself. Formally this can be expressed as

$$\texttt{Int}\,(\mathrm{u}, \mathrm{v}) = \sum_{\mathrm{u}' \leq \mathrm{u}} \sum_{\mathrm{v}' \leq \mathrm{v}} I\left(\mathrm{u}', \mathrm{v}'\right) \tag{3.10}$$

where $I\,(\mathrm{u}, \mathrm{v})$ denotes the gray-scale value of the pixel at position (u, v) in image I. The computation of (3.10) can be done efficiently in a single pass over image I by employing

$$\texttt{Int}\,(\mathrm{u}, \mathrm{v}) = I\,(\mathrm{u}, \mathrm{v}) + \texttt{Int}\,(\mathrm{u} - 1, \mathrm{v}) + \texttt{Int}\,(\mathrm{u}, \mathrm{v} - 1) - \texttt{Int}\,(\mathrm{u} - 1, \mathrm{v} - 1) \tag{3.11}$$

To further speed up the computation oftentimes an additional column above the first column and an additional row left of the first row are added to resultant the integral image. These rows are filled with 0s, so that equation (3.11) can be employed without any special consideration of the boundaries. From (3.10) and (3.11) follows that computational cost for the integral image is in $\mathcal{O}\,(n)$ where n is the number of pixels contained in one image (per pixel 2 additions and 1 subtraction are needed).

Still the benefit of integral images might not be obvious at first. Once constructed, an integral image allows the summation of the image intensity in a rectangular area specified by its four corners in constant

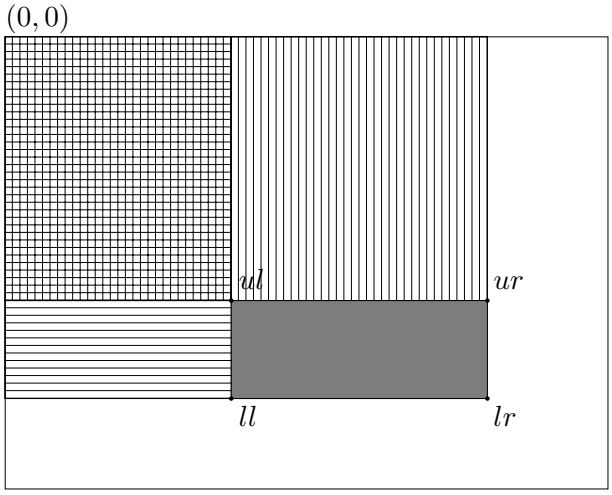


Figure 3.8: Summation over rectangle in integral image. To get the sum of the intensity of all pixels inside the rectangle given by ul and lr (colored in gray) the rectangle specified by $(0, 0)$ and lr is added to the rectangle specified by $(0, 0)$ and ul. The rectangles marked by vertical lines (defined by $(0, 0)$ and ur) and horizontal lines (defined by $(0, 0)$ and ll) are subtracted. Due to the nature of the integral image the needed sums can be accessed by 4 simple references (see equation (3.12)).

time. Consider the corners of a given rectangle as upper left $(\mathrm{u}_{ul}, \mathrm{v}_{ul})$, upper right $(\mathrm{u}_{ur}, \mathrm{v}_{ur})$, lower left $(\mathrm{u}_{ll}, \mathrm{v}_{ll})$ and lower right $(\mathrm{u}_{lr}, \mathrm{v}_{lr})$. The summation over all pixels in the defined rectangle is given by

$$\sum_{\mathrm{u}=\mathrm{u}_{ul}}^{\mathrm{u}_{lr}} \sum_{\mathrm{v}=\mathrm{v}_{ul}}^{\mathrm{v}_{lr}} I(\mathrm{u}, \mathrm{v}) = \mathtt{Int}(\mathrm{u}_{ul}, \mathrm{v}_{ul}) + \mathtt{Int}(\mathrm{u}_{lr}, \mathrm{v}_{lr}) - \mathtt{Int}(\mathrm{u}_{ur}, \mathrm{v}_{ur}) - \mathtt{Int}(\mathrm{u}_{ll}, \mathrm{v}_{ll}) \tag{3.12}$$

A graphical representation of equation (3.12) is provided by Figure 3.8. The property described in equation (3.12) allows for very fast calculation of box filters, simply by calculating the sum of the rectangle specified by the filter according to equation (3.12) and multiplying it by the weight of the box filter. This characteristic was first exploited by Viola and Jones [49] for object recognition, using cascades of Haar wavelets as classifiers. Since then it has been used in other applications, for example the approximation of the Hessian of Gaussians in scale space in SURF-features (see subsection 3.1.2 SURF). It also becomes very useful in the comparison of image patches, as shown in subsection 3.2.2. It should be noted that it is sometimes beneficial to also calculate the integral image of squared sums $\mathtt{Int}_{\mathtt{sq}}$.

$$\mathtt{Int}_{\mathtt{sq}}(\mathrm{u}, \mathrm{v}) = \sum_{\mathrm{u}' \leq \mathrm{u}} \sum_{\mathrm{v}' \leq \mathrm{v}} I(\mathrm{u}', \mathrm{v}')^2 \tag{3.13}$$

The integral image of squared sums can be calculated like the integral image in a single pass simply by exchanging $I(\mathrm{u}, \mathrm{v})$ in equation (3.11) with $I(\mathrm{u}, \mathrm{v})^2$. Therefore $\mathtt{Int}$ and $\mathtt{Int}_{\mathtt{sq}}$ are usually calculated together in one single pass over image I if required.

3.2.2 Image Patch Comparison

The following subsection is supposed to give a short overview how images or image patches can be compared efficiently. In the following the terms image and image patch will be used interchangeably,

since it makes no algorithmic difference if whole images or just parts of them are compared. The purpose of the comparison is to determine if in two given image patches the same scene is depicted. It is important to note, that both images need to have the same size ($\mathrm{width} \times \mathrm{height}$) and the discussed techniques are applicable to gray-scale or any other single channel images.

For this purpose a comparison function $C\left(I, I'\right)$ is needed which gets two images as input and should indicate if these images are similar or not. The perhaps most obvious idea to solve this task is to just calculate the sum of distances between single pixels in Images I and I':

$$C_{dist}\left(I, I'\right) = \sum_{\mathrm{u}} \sum_{\mathrm{v}} I\left(\mathrm{u}, \mathrm{v}\right) - I'\left(\mathrm{u}, \mathrm{v}\right) \tag{3.14}$$

While (3.14) would correctly determine a distance of 0 for two identical images it would fail in many other cases (a gray image would be considered similar to an image showing white and black in equal parts). Using the absolute distances instead of the subtraction would correct this problem. However it is usually desired to weight one great difference stronger than many small differences, since this reduces the influence of random noise. Therefore the SSD is often considered a good measure of comparison

$$C_{ssd}\left(I, I'\right) = \sum_{\mathrm{u}} \sum_{\mathrm{v}} \left(I\left(\mathrm{u}, \mathrm{v}\right) - I'\left(\mathrm{u}, \mathrm{v}\right)\right)^2 \tag{3.15}$$

Still (3.15) will have problems with images under different illumination. If $I'\left(\mathrm{u}, \mathrm{v}\right) = I\left(\mathrm{u}, \mathrm{v}\right) + k$ where $\forall \mathrm{u}, \mathrm{v}: \ 0 \leq \mathrm{u} < \mathrm{width}, 0 \leq \mathrm{v} < \mathrm{height}, k \in \mathbb{Z}$ holds, then both images show exactly the same image under different lighting conditions, but C_{ssd} will indicate a difference between I and I'. Another problem might be that the codomain of C_{ssd} is dependent on the size of the compared images. This inhibits the use of fixed thresholds to evaluate results different from perfect matches ($C_{ssd} = 0$). To address the problems of the aforementioned comparison functions the *normalized cross-correlation* (abbreviated NCC in the following) is a suitable function. It is defined as

$$C_{\mathtt{NCC}}\left(I, I'\right) = \frac{1}{n} \sum_{\mathrm{u}} \sum_{\mathrm{v}} \frac{\left(I\left(\mathrm{u}, \mathrm{v}\right) - \bar{I}\right)\left(I'\left(\mathrm{u}, \mathrm{v}\right) - \bar{I}'\right)}{\sigma_I \sigma_{I'}} \tag{3.16}$$

where $\bar{I}$ and $\bar{I}'$ are the mean values of image I and I' respectively, $n = \mathrm{width} \cdot \mathrm{height}$ denotes the number of pixels in the image patches and σ_I and $\sigma_{I'}$ are the standard deviation of images I and I'. Contrary to the (absolute) distance of the SSD small results for the NCC do not indicate good matches. The range of values for the NCC is defined as $C_{\mathtt{NCC}}\left(I, I'\right) \mapsto [-1, 1]$ where 1 indicates a perfect match and -1 would correspond to a perfect mismatch (i.e. the image is compared to its inverted image). The use of the standard deviation provides for robustness against changing illumination.

Obviously the normalized cross correlation is more costly to compute than (3.14) or (3.15). Naive implementation will not be fast enough to estimate a good correspondence in the available time window. Consider the scenario that a given image patch p is compared to a region of identical size in image I. The objective is to find the position where patch and image correspond best and if the score of the NCC is above a given threshold it may be assumed that the patch is depicted in this part of the image. Like kernels, patches in this scenario are typically of odd width and height (but not necessarily squared), so that there is one center pixel which determines the position of the patch in the image. The image patch's size is given by $p_w \times p_h$ and the search region in image I is denoted by S_I. S_I can take various shapes, most common are rectangles or circles around a given image coordinate (u, v), but also a set of listed

image coordinates would work. Since patch p will not change it is assumed, that standard deviation σ_p and mean $\bar{p}$ are previously determined, the part of the image compared with p around image coordinate (u, v) is denoted as $p_I(\mathrm{u}, \mathrm{v})$, its mean and standard deviation as $\bar{p}_I$ and σ_{p_I}. To simplify the expressions it is furthermore assumed that the image coordinates in p and p_I are the same, i.e. in both the upper left corner refers to $(0, 0)$ and the lower right to $(p_w - 1, p_h - 1)$. The naive implementation of `NCC` might look like Algorithm 3.1. Note that the operations in lines 3 and 4 of Algorithm 3.1 are also loops, going

Algorithm 3.1: Naive Implementation of `NCC`.
Please note that functions `getMean()` and `getStdDev()` include loops over all pixel in p_I.

input : image patch p, image I and a search region S_I
output: correlation score best_score and corresponding position best_pos

```
1  best_score ← −1
2  best_pos ← (−1, −1)
3  patch_size ← p_w * p_h
4  forall (i,j) in S_I do
5      p_I ← getPatch(i,j)
6      p̄_I ← getMean(p_I)            // loop over all pixels in p_I required
7      σ_pI ← getStdDev(p_I)         // loop over all pixels in p_I required
8      sum ← 0
9      forall (u,v) in p_I do
10         sum ← sum + (p(u,v) − p̄) * (p_I(u,v) − p̄_I)
11     end
12     sum ← sum / ( patch_size * σ_p * σ_pI)
13     if sum > best_score then
14         best_score ← sum
15         best_pos ← (i,j)
16     end
17 end
```

over all elements of p_I.

There exist however several methods to speed up the calculation of equation (3.16). Consider the definition of the standard deviation of an image patch p of size $n = p_w \times p_h$ with its upper left corner at position $(\mathrm{u}_p, \mathrm{v}_p)$ in image I:

$$\begin{aligned}\sigma_p &= \sqrt{\frac{1}{n}\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)-\bar{p}\right)^2} \\ &= \sqrt{\frac{1}{n}\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(I\left(\mathrm{u}+\mathrm{u}_p,\mathrm{v}+\mathrm{v}_p\right)-\frac{1}{n}\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}I\left(\mathrm{u}+\mathrm{u}_p,\mathrm{v}+\mathrm{v}_p\right)\right)^2}\end{aligned} \tag{3.17}$$

This expression can be simplified, if a closer look is taken to $\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)-\bar{p}\right)^2$ This can be

rewritten as

$$\begin{aligned}\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)-\bar{p}\right)^2 &= \sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)^2-2p\left(\mathrm{u},\mathrm{v}\right)\bar{p}+\bar{p}^2\right)\\ &= \sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)^2\right)-2\bar{p}\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)\right)+n\bar{p}^2\\ &= \sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)^2\right)-2\bar{p}n\bar{p}+n\bar{p}^2\\ &= \sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)^2\right)-n\bar{p}^2 \qquad (3.18)\end{aligned}$$

Combining equations (3.18) and (3.17) results in

$$\sigma_I=\sqrt{\frac{1}{n}\left(\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}p\left(\mathrm{u},\mathrm{v}\right)^2\right)-\bar{p}^2} \qquad (3.19)$$

If the integral image $\texttt{Int}$ (see equation (3.10)) and the integral image of squared sums $\texttt{Int}_{\texttt{sq}}$ (equation (3.13)) are calculated beforehand, $\bar{p}^2$ and $\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}p\left(\mathrm{u},\mathrm{v}\right)^2$ can be calculated very efficiently as:

$$\bar{p}=\frac{\texttt{Int}\left(\mathrm{u}_p,\mathrm{v}_p\right)+\texttt{Int}\left(\mathrm{u}_p+p_w-1,\mathrm{v}_p+p_h-1\right)}{n}-\frac{\texttt{Int}\left(\mathrm{u}_p,\mathrm{v}_p+p_h-1\right)+\texttt{Int}\left(\mathrm{u}_p+p_w-1,\mathrm{v}_p\right)}{n}$$

and

$$\begin{aligned}\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}p\left(\mathrm{u},\mathrm{v}\right)^2 = \texttt{Int}_{\texttt{sq}}\left(\mathrm{u}_p,\mathrm{v}_p\right)+\texttt{Int}_{\texttt{sq}}\left(\mathrm{u}_p+p_w-1,\mathrm{v}_p+p_h-1\right)-\\ \texttt{Int}_{\texttt{sq}}\left(\mathrm{u}_p,\mathrm{v}_p+p_h-1\right)-\texttt{Int}_{\texttt{sq}}\left(\mathrm{u}_p+p_w-1,\mathrm{v}_p\right)\end{aligned}$$

Since the computation of $\texttt{Int}$ and $\texttt{Int}_{\texttt{sq}}$ are in $\mathcal{O}\left(n\right)$ (n being the number of pixels) their computations pays of quickly especially if several image patches need to be compared in large search areas. However optimization can be taken another step further. First consider the numerator of equation (3.16) for the case of comparing a patch p with a region of image I of equal size and upper left corner $\left(\mathrm{u}_p,\mathrm{v}_p\right)$. Let $\bar{p}_I$ be the mean value of the defined region in I. With a similar approach as for the standard deviation the

numerator of equation (3.16) can be written as

$$\begin{aligned}
&\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)-\bar{p}\right)\left(I\left(\mathrm{u}+\mathrm{u}_p,\mathrm{v}+\mathrm{v}_p\right)-\bar{p}_I\right)\\
&=\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)I\left(\mathrm{u}+\mathrm{u}_p,\mathrm{v}+\mathrm{v}_p\right)-p\left(\mathrm{u},\mathrm{v}\right)\bar{p}_I-\bar{p}I\left(\mathrm{u}+\mathrm{u}_p,\mathrm{v}+\mathrm{v}_p\right)+\bar{p}\bar{p}_I\right)\\
&=\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)I\left(\mathrm{u}+\mathrm{u}_p,\mathrm{v}+\mathrm{v}_p\right)\right)-2n\bar{p}\bar{p}_I+n\bar{p}\bar{p}_I\\
&=\sum_{\mathrm{u}=0}^{p_w-1}\sum_{\mathrm{v}=0}^{p_h-1}\left(p\left(\mathrm{u},\mathrm{v}\right)I\left(\mathrm{u}+\mathrm{u}_p,\mathrm{v}+\mathrm{v}_p\right)\right)-n\bar{p}\bar{p}_I
\end{aligned}\tag{3.20}$$

Though the first part of the numerator with the double sum has to be computed in a loop, the second part can be obtained by the integral image (see above). Employing both equations (3.19) and (3.20) results in Algorithm 3.2.

Algorithm 3.2: Efficient implementation of `NCC`.
This algorithm combines the simplifications stated in (3.19) and (3.20).

input : image patch p, image I, search region S_I `Int` and $\texttt{Int}_{\texttt{sq}}$ of I
output: correlation score best_score and corresponding position best_pos

```
best_score ← −1
best_pos ← (−1, −1)
patch_size ← p_w * p_h
forall (i,j) in S_I do
    p_I ← getPatch(i,j)
    i_sum ← getPatchSum(Int, i, j, p_w, p_h)
    i_sum2 ← getSquaredPatchSum(Int_sq, i,j, p_w, p_h)
    sum_ppI ← 0
    forall (u,v) in p do
        sum_ppI ← sum_ppI + p(u,v) * p_I(u,v)
    end
    numerator ← sum_ppI − i_sum * p̄
    denom ← (i_sum2 − i_sum * i_sum / patch_size) * σ_p^2
    score ← numerator / sqrt(denom)
    if score > best_score then
        best_score ← score
        best_pos ← (i,j)
    end
end
```

The OPENCV function `cvMatchTemplate()` uses some additional low-level optimizations to Algorithm 3.2, but the basic principle is the same. Of course the overhead computation needed for

Specifications		Algorithm				
		3.1	3.2		`cvMatchTemplate()`	
Patch Size	Search Region	ms/patch	ms/patch	% of 3.1	ms/patch	% of 3.1
5×5	3×3	0.0433	0.0400	92	0.0457	105
5×5	5×5	0.1100	0.0766	70	0.0465	42
5×5	10×10	0.4239	0.2528	60	0.1067	25
7×7	3×3	0.0763	0.0400	52	0.0530	69
7×7	5×5	0.1947	0.0877	45	0.0503	26
7×7	10×10	0.7460	0.2778	37	0.1156	16
11×11	3×3	0.1643	0.0498	30	0.0492	30
11×11	5×5	0.4372	0.1015	23	0.0724	17
11×11	10×10	1.7474	0.3513	20	0.1151	7
21×21	3×3	0.5598	0.0925	17	0.1082	19
21×21	5×5	1.5366	0.2064	13	0.1100	7
21×21	10×10	6.0853	0.6708	11	0.2804	5

Table 3.1: Speed comparison for `NCC` computation. All run-times were determined experimental as the average of 153000 runs on a Intel Core2Duo 2.26 GHz processor (no multi-threading used here). Still it is doubtful, how reliable the measured run-times are, therefore run-time percentage compared to Algorithm 3.1 is rounded to integer values and should more be seen as a general indicator, than hard fact.

the integral image and the integral image of squared sums pays off more the larger search area and image patch become. In Table 3.1 a run-time comparison of the algorithms is depicted. Please note that patches of size 11×11 and 21×21 give stable results, smaller patch sizes may decrease correct matching. Search regions up to 10×10 pixels or even larger might temporary be needed in VISUAL MONO-SLAM. Usually around 10 image patches need to be matched in one frame. If not all necessary operations can be completed this can lead to a vicious circle, since missed frames induce larger search areas in the subsequent frame.

Chapter 4

VISUAL MONO-SLAM

After all the basics have been discussed in chapter 2 and 3 this chapter will explain how the discussed concepts and algorithms work together in VISUAL MONO-SLAM.

In the first section 4.1 the general concept of the Extended Kalman Filter is introduced from a theoretic standpoint. Afterwards it will be shown how an EKF can be used to model 3D positions of distinct feature points and the pose and velocities of the observing camera. Therefore in section 4.2 the state representation will be defined, while section 4.3 presents the transition function along its Jacobian. The function to obtain the measurement prediction and its Jacobian are discussed in section 4.4, followed by section 4.5 devoted to incorporate the actual measurements gained by feature matching. The description of the EKF algorithm for the VISUAL MONO-SLAM application is concluded by section 4.6 which contains the update step along all related operations. The remainder of this chapter covers various issues concerning the two different 3D point representations introduced in section 4.2. First section 4.7 provides a mathematical analysis of the linearity of the depth estimation of both 3D point representations, followed by section 4.8 which discusses how additional features can be incorporated in the existing EKF. Afterwards in section 4.9 conversion from one point representation to the other is presented and the chapter is closed by a brief explanation on the deletion of features in the EKF.

This chapter is supposed to fully illustrate the workings of an EKF at a very detailed level, since many descriptions of EKFs in the literature are either on a very superficial level where most of the details are omitted or the EKF is discussed in a mathematical context without any application background. Thus the reader not familiar with the EKF in the first place might sometimes get the impression that its a priori state estimation and its correction happen by “magic”. Hopefully this chapter will show that no “magic” but plain (and sometimes tedious) mathematics can be used to do all the work. Before the technical details will be explained in their appropriate sections the main idea of VISUAL MONO-SLAM will be presented first.

VISUAL MONO-SLAM will create a consistent 3 dimensional map of the environment by movement of a single low-cost camera. The environment will be represented by single 3D points. As explained in section 2.4 depth information can only be gathered if the same point is observed from at least two known poses. Therefore the pose of the camera needs to be estimated as well. The estimation is done via an EKF, assuming a static environment. Therefore changes in the received images are contributed to camera movement. Dependent on the amount of movement and the depth of a point its parallax will start to differ. Repeated observation of the same points and comparing the location of their projections with

the expected locations refines both camera pose estimation and point location estimation. The underlying method is triangulation, of course, but the whole triangulation process is implicitly handled by the EKF and does not need to be modeled explicitly. Another nice property of the EKF is that different points in the EKF will become connected via the covariance matrix. This means that the location estimation of points that were not observed during one frame may still be improved by the observation of other points. How this main idea is implemented will be shown in the following.

4.1 Extended Kalman Filter

This chapter will provide a short introduction to the *Extended Kalman Filter* (EKF) which is used as the underlying mechanism to estimate pose and feature positions in VISUAL MONO-SLAM.

The EKF is an extension of the Kalman Filter to model non-linear systems. Since the Kalman Filter and EKF are quite similar in respect to the underlying theory only the EKF will be discussed further. Please note that only a very brief and incomplete (in terms of proofs etc.) introduction to the EKF is given in this chapter. Extensive information on the topic of Kalman Filters and other probabilistic methods, including proofs, computational complexity analysis and application examples can be found in [47] which is highly recommended for more details.

The *Kalman Filter* is a well known and widely used recursive *Gaussian filter* to estimate the state of continuous linear systems under uncertainty. That means, that the state vector $\mathbf{x}_t$ of a system is modeled by a multivariate Gaussian distribution with mean μ_t and covariance Σ_t, at time t. What the state denotes is dependent on the application, of course. In VISUAL MONO-SLAM for example the state will encode the 3D position of the camera in the world coordinate system, its orientation and velocities and the estimated positions of all observed features. The system will be observed at discrete points in time, where the current time is always referred to as t, while the previous time steps are $t-1$, $t-2$ etc. The system can be influenced in each time step by a set of actions denoted as a_t. Furthermore it is assumed that some sort of sensor with measurement function h exists which can be used to gain (noisy) measurements z_t about the system at time t. Each time step is structured in two phases: First comes the so called *prediction step* and afterwards the *update step*. The basic idea of the prediction is to estimate into which state the system should be transferred to from estimated state $(\mu_{t-1}, \Sigma_{t-1})$ if actions a_t are executed. This is done by transition function $g(a_t, \mu_{t-1})$. Once the next state $(\bar{\mu}_t, \bar{\Sigma}_t)$ is predicted, a measurement prediction has to be made. In order to do this, a measurement model with a function $h(\bar{\mu}_t)$ is needed, where $h(\bar{\mu}_t)$ returns the measurements which are expected, if the system would in fact be in state $(\bar{\mu}_t, \bar{\Sigma}_t)$. After the execution a measurement z_t is taken and the actual state of the system is compared with the predicted state. Both prediction and measurement influence the new estimated state (μ_t, Σ_t). Algorithm 4.1 shows the the previous description in a more compact way. Note that while the measurements z_t and the actions a_t can be directly observed, the rest of the system, especially state $\mathbf{x}_t$ can not be observed directly, but is estimated through sensor measurements and actions. If the true state could be observed there would be no need for an estimation and thus an EKF. This distinction is depicted in Figure 4.1.

In the remainder a closer look is taken to Algorithm 4.1 to properly explain the meaning of all involved terms. As stated above, the Kalman Filter can only be used for linear systems, which is a rather severe limitation, since many "interesting" systems do not behave linearly. Since only linear systems are considered in the Kalman Filter the prediction of the mean $\bar{\mu}_t$ (Algorithm 4.1, line 1) is done by

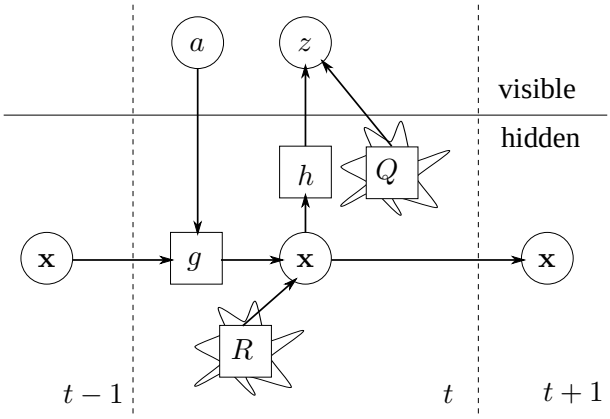


Figure 4.1: Schematic EKF Sequence.
Legend: g – transition function, h – measurement function, R – transition noise, Q – sensor noise, a – actions, t – time and $\mathbf{x}$ – state. Arrows indicate influences. Keep in mind that $\mathbf{x}$ denotes the real state and not its estimation. Therefore there is no arrow from the measurement z to state $\mathbf{x}$, since the measurement should not change the actual state of the system (while it is likely to change the estimation of the state).

Algorithm 4.1: Extended Kalman Filter

input : prev. mean μ_{t-1}, prev. covariance Σ_{t-1}, actions a_t, measurements z_t
output: mean μ_t, covariance Σ_t

1 $\bar{\mu}_t = g\left(a_t, \mu_{t-1}\right)$
2 $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3 $K_t = \bar{\Sigma}_t H_t^T \left(H_t \bar{\Sigma}_t H_t^T + Q_t\right)^{-1}$
4 $\mu_t = \bar{\mu}_t + K_t \left(z_t - h\left(\bar{\mu}_t\right)\right)$
5 $\Sigma_t = \left(\mathbb{1} - K_t H_t\right) \bar{\Sigma}_t$ // $\mathbb{1}$ *is identity matrix*

multiplying the actions a_t with an appropriate matrix, encoding the state transition. The EKF uses *linearization* of to enable estimation of systems, featuring non-linear behavior. In a non-linear system the state transition cannot be expressed by a matrix (since the system would be linear otherwise), but is expressed by some function g. To linearize g first order Taylor expansion is used, which creates a linear approximation of g, dependent on the properties of the first order derivative g'. Since Taylor expansion approximates a function from a single point and the value of the derivatives at that point, for a Gaussian functions it is reasonable to use the point of the largest likelihood as that point. This is given by the last mean μ_{t-1}. The linearization for g is therefore given by

$$\begin{aligned} g\left(a_t, \mathbf{x}_{t-1}\right) &\approx g\left(a_t, \mu_{t-1}\right) + g'\left(a_t, \mu_{t-1}\right)\left(\mathbf{x}_{t-1} - \mu_{t-1}\right) \\ &= g\left(a_t, \mu_{t-1}\right) + G_t\left(\mathbf{x}_{t-1} - \mu_{t-1}\right) \end{aligned} \tag{4.1}$$

where $\mathbf{x}_t$ denotes the actual state at time $t-1$ and G_t the *Jacobian* of $g\left(a_t, \mu_{t-1}\right)$. The Jacobian G_t plays an important part in the estimation of the covariance (see Algorithm 4.1, line 2). To model noise in the transition function a Gaussian random variable with mean 0 and covariance R_t is incorporated in

the state prediction. This is reflected in the additive term R_t in line 2 of Algorithm 4.1 while line 1 is not influenced, since the mean of the noise is 0. The same linearization method used on g is applied to measurement function h, resulting in its Jacobian H_t (Algorithm 4.1, line 3). In the same line matrix K_t is calculated, which is referred to as *Kalman gain* in the literature. The Kalman gain can be interpreted as a weight how strongly the actual measurement z_t can influence the predicted mean $\bar{\mu}_t$, resulting in μ_t (see line 4). Finally the Kalman gain and the Jacobian of the measurement function are used to update the estimation of the covariance (line 5).

Please note that the EKF after the linearization for g and h according to equation (4.1) basically corresponds to the well-studied Kalman Filter. Due to this fact and its simplicity and efficiency compared to other methods to estimate non-linear systems the EKF is currently one of the most popular approaches in this field. The computational most expensive part of the EKF is the matrix inversion in line 3. According to [47] the EKF is in $\mathcal{O}\left(k^{2.4} + n\right)$, where k is the dimension of measurement vector z_t and n denotes the dimension of state $\mathbf{x}_t$. Compared to other approaches like particle filters, which can be exponential in n this is quite fast. Still for high frequency applications like VISUAL MONO-SLAM the matrix inversion limits the feasible size of the state vector $\mathbf{x}_t$. What this means in practice will be discussed in chapter 4 alongside the practical implementation of an EKF as the core of VISUAL MONO-SLAM.

4.2 State Representation

As stated in section 4.1 the state vector $\mathbf{x}_t$ is not directly observable but can only indirectly be estimated by prediction, sensor measurements and subsequent fusion of prediction and measurement. All information important for the system needs to be encoded in the state, which implies for VISUAL MONO-SLAM that the current position and orientation of the camera need to be included in the state vector as well as the estimated positions of all present features in the map. In practice that means that $\mathbf{x}_t$ is composed of two parts. The first will represent the camera state and this part will not vary in its dimension. The second part of $\mathbf{x}_t$ will contain the features and their estimated positions, thus making up the map. This part will vary in dimension, since the map is initially empty and will grow (and eventually shrink) over time.

Before the different parts contributing to the state vector are considered first some short notes on the notation used in the following are given: In the following two coordinate systems will be considered, namely the world coordinate system W and the camera coordinate system C. These coordinate system may be denoted as superscripts in the remainder to indicate relative to which coordinate system a variable is defined (for example ω^C). To better distinguish vectors from scalars vectors will be printed bold and in non-italics (so $\mathbf{x}$ denotes a vector while x would denote a scalar).

It should be noted that VISUAL MONO-SLAM as presented in the following assumes a right-hand coordinate system, for both world coordinates and camera coordinates. This is later illustrated among other things in Figure 4.3.

4.2.1 Camera Representation

Information concerning the camera is encoded in a 13 dimensional vector $\mathbf{x}_v$, which embodies the first entries in the complete state vector $\mathbf{x}_t$. Vector $\mathbf{x}_v$ has the following appearance:

$$\mathbf{x}_v = \begin{pmatrix} \mathbf{r}^{WC} \\ \mathbf{q}^{WC} \\ \mathbf{v}^{W} \\ \omega^{C} \end{pmatrix} \tag{4.2}$$

where $\mathbf{r}^{WC} = (x_c \ y_c \ z_c)^T$ denotes the 3D position of the camera optical center in the world coordinate system, $\mathbf{q}^{WC}$ the unit *quaternion* specifying the camera orientation relative to the world frame, $\mathbf{v}^{W}$ encodes the linear velocities of the camera along the coordinate axes of W and ω^{C} the angular velocities relative to the camera coordinate system C. Thus the camera state will be represented by a 13 dimensional vector. Initially the camera will be positioned in the origin of the world coordinate system ($\mathbf{r}^{WC} = (0 \ 0 \ 0)^T$), looking into the direction of the positive Z_W-axis ($\mathbf{q}^{WC} = (1 \ 0 \ 0 \ 0)^T$) and the camera is assumed to be unmoving ($\mathbf{v}^{W} = \omega^{C} = (0 \ 0 \ 0)^T$).

The 13×13 covariance matrix $\mathbf{P}$ is initialized as

$$\mathbf{P} = \begin{pmatrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & V \end{pmatrix}, V = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{pmatrix}$$

If the reader should be unfamiliar with quaternions and how they can be used to represent rotations and their advantages compared to other rotation notations [3, 43] provides useful information on this topic.

4.2.2 XYZ Feature Representation

The parametrization of a 3 dimensional feature $\mathbf{x}_i$ in the EKF seems straightforward by simply denoting $\mathbf{x}_i$ as

$$\mathbf{x}_i = \begin{pmatrix} x_i & y_i & z_i \end{pmatrix}^T \tag{4.3}$$

referred to as XYZ encoding. As discussed in section 2.4 cameras are not able to measure depth immediately, but can only calculate depth by triangulation for given correspondences and two or more known camera positions. At the first observation of a newly initialized feature no information about its depth can be deduced. While the EKF is able to cope with non-linear transition and measurement functions by linearization, its state estimation estimation modeled as Gaussian (μ_t, Σ_t) has to be linear. Unfortunately if a feature is not initialized with its guessed depth close to its real depth (which is quite unlikely) the depth estimation will not behave linear and thus cannot be correctly modeled in an EKF.

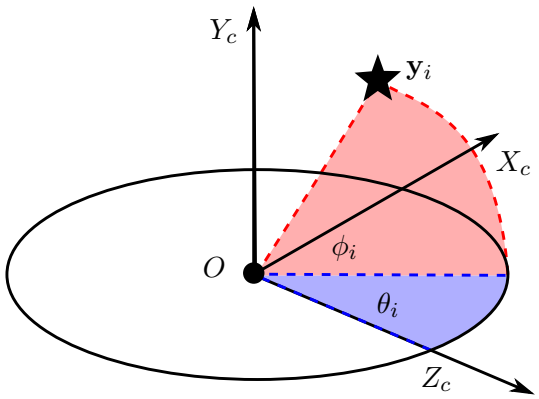


Figure 4.2: Azimuth and elevation in camera coordinate system. The camera optical center is denoted as O, coordinate axes are labeled X_c, Y_c and Z_c. For the observed feature $\mathbf{y}_i$, marked by a ★, azimuth θ_i and elevation ϕ_i are depicted. The azimuth is shaded in blue, while the elevation is shaded red.

4.2.3 Inverse Depth Feature Representation

To address the problems caused by initialization of features in XYZ encoding the authors of [13, 16, 20] propose an *inverse depth* encoding of features. Since the inverse depth is linear in contrast to depth this is a feasible approach. A feature $\mathbf{y}_i$ in inverse depth encoding is comprised by the following 6 dimensional vector:

$$\mathbf{y}_i = \begin{pmatrix} x_{c,i} & y_{c,i} & z_{c,i} & \theta_i & \phi_i & \rho_i \end{pmatrix}^T \tag{4.4}$$

where $x_{c,i}$, $y_{c,i}$, $z_{c,i}$ specify the 3D position of the camera's optical center at the first observation of feature $\mathbf{y}_i$. θ_i and ϕ_i are *azimuth* and *elevation* (see Figure 4.2) of the feature in reference to the camera coordinate system and ρ_i is the inverse depth of $\mathbf{y}_i$.

The 3D point modeled by (4.4) is given by

$$\mathbf{x}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} x_{c,i} \\ y_{c,i} \\ z_{c,i} \end{pmatrix} + \frac{1}{\rho_i} m(\theta_i, \phi_i) \tag{4.5}$$

$$m(\theta_i, \phi_i) = \begin{pmatrix} \sin\theta_i \cos\phi_i & -\sin\phi_i & \cos\theta_i \cos\phi_i \end{pmatrix}^T \tag{4.6}$$

Function $m(\theta_i, \phi_i)$ in equation (4.6) yields a unit vector pointing from the camera's optical center to feature $\mathbf{y}_i$. Multiplying this vector with the depth $d_i = \frac{1}{\rho_i}$ and adding it to the position of the first observation $(x_{c,i} \;\; y_{c,i} \;\; z_{c,i})^T$ results in the concurrent 3D position (equation (4.5)).

Both XYZ encoded features and inverse depth features consist of more than just the description of a 3 dimensional point by equations (4.3) and (4.4) respectively. In addition some comparison mechanism is needed. Whether this is done by comparison of an image patch, as presented in subsection 3.2.2 or by comparison of high level descriptors (see subsection 3.1.2) depends on the actual implementation and has no influence on the underlying mathematical modulation of the EKF.

The full state vector $\mathbf{x}_t$ for a map composed of n features is therefore composed of

$$\mathbf{x}_t = \begin{pmatrix} \mathbf{x}_v^T & \mathbf{f}_1^T & \mathbf{f}_2^T & \dots & \mathbf{f}_n^T \end{pmatrix}^T \tag{4.7}$$

where $\mathbf{f}_i \in \{\mathbf{x}_i, \mathbf{y}_i\}$ denotes a feature either in XYZ or inverse depth encoding.

4.3 State Transition

The state transition function g_v for the camera in VISUAL MONO-SLAM is quite simple. As stated in equation (4.2) the camera is defined by the 3D position of its optical center $\mathbf{r}^{WC}$ its orientation as a quaternion $\mathbf{q}^{WC}$ and linear and angular velocities $\mathbf{v}^W$ and ω^C. In section 4.1 the transition function $g\left(u_t, \mu_{t-1}\right)$ was dependent on the current actions u_t and the previous mean μ_{t-1}. In the case of a free moving camera no observable explicit commands are given, thus the transition function in this case will solely depend on μ_{t-1} and is defined as

$$g_v\left(\mu_{t-1}\right) = \begin{pmatrix} \mathbf{r}_t^{WC} \\ \mathbf{q}_t^{WC} \\ \mathbf{v}_t^W \\ \omega_t^C \end{pmatrix} = \begin{pmatrix} \mathbf{r}_{t-1}^{WC} + \mathbf{v}_{t-1}^W \Delta t \\ \mathbf{q}_{t-1}^{WC} \times \operatorname{quat}\left(\omega_{t-1}^C \Delta t\right) \\ \mathbf{v}_{t-1}^W \\ \omega_{t-1}^C \end{pmatrix} \tag{4.8}$$

where Δt is defined as the difference of t and $t-1$ and $\operatorname{quat}\left(\omega_{t-1}^C \Delta t\right)$ is the quaternion corresponding to the rotation of $\omega_{t-1}^C \Delta t$. To compute the quaternion from the given angular velocities and time difference Δt two operations are necessary. First the the angle-axis $a = \langle \mathbf{a}, \alpha \rangle$ representation of the rotation is calculated. For given angular velocity $\omega_t^C = \begin{pmatrix} \omega_{t,X}^C & \omega_{t,Y}^C & \omega_{t,Z}^C \end{pmatrix}^T$ and Δt the equivalent angle-axis representation is given by

$$a = \langle \mathbf{a}, \alpha \rangle = \langle \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}, \alpha \rangle = \langle \begin{pmatrix} \frac{\omega_{t,X}^C \Delta t}{\|\omega_t^C \Delta t\|} \\ \frac{\omega_{t,Y}^C \Delta t}{\|\omega_t^C \Delta t\|} \\ \frac{\omega_{t,Z}^C \Delta t}{\|\omega_t^C \Delta t\|} \end{pmatrix}, \|\omega_t^C \Delta t\| \rangle \tag{4.9}$$

where $\omega_{t,\gamma}^C,\ \gamma \in \{X, Y, Z\}$ refers to the angular velocity around the indicated coordinate axis. The result of (4.9) is then transferred to the quaternion $\mathbf{q}$ denoting the same rotation:

$$\mathbf{q} = \begin{pmatrix} \cos\frac{\alpha}{2} & \frac{a_x}{\|\mathbf{a}\|}\sin\frac{\alpha}{2} & \frac{a_y}{\|\mathbf{a}\|}\sin\frac{\alpha}{2} & \frac{a_z}{\|\mathbf{a}\|}\sin\frac{\alpha}{2} \end{pmatrix}^T \tag{4.10}$$

Since no information about any actions like accelerations of linear or angular velocity are available both velocity vectors are predicted to be the same in $\bar{\mu}_t$ as in μ_{t-1}. If during measurement contradicting information concerning this assumption is gathered, this will be incorporated in the update step. As all features in the map are assumed to be static their estimations are not changed by the transition function. This leads to the following complete transition function:

$$g\left(\mu_{t-1}\right) = \begin{pmatrix} g_v\left(\mu_{t-1}\right) \\ \mathbf{0} \end{pmatrix} \tag{4.11}$$

where $\mathbf{0}$ denotes the 0 vector and $\dim(\mathbf{0}) = n_{\dim} - 13$ if $\dim(\mu_{t-1}) = n_{\dim}$.

Please note that for similar applications of VISUAL MONO-SLAM it might be possible to directly observe issued commands. If for example a single camera is mounted on top of a robot, its steering commands would definitely have an impact on the appearance of the transition function.

With the complete transition function $g(\mu_{t-1})$ defined in equation (4.11) for the VISUAL MONO-SLAM scenario its Jacobian G_t should be considered, to complete the prediction step of the EKF (lines 1 and 2 in algorithm 4.1). According to (4.11) for a n-dimensional mean vector μ_{t-1} and belonging $n_{\dim} \times n_{\dim}$ covariance Σ_{t-1} Jacobian G_t is of the following form

$$G_t = \begin{pmatrix} F_t & 0 \\ 0 & 0 \end{pmatrix}, \ \dim(F_t) = 13 \times 13, \ \dim(G_t) = n_{\dim} \times n_{\dim} \tag{4.12}$$

In the remainder of 4.3 the appearance of F_t will be examined more closely. Matrix F_t is the Jacobian of $g_v(\mu_{t-1})$, defined in equation (4.8). Judging from (4.8) the structure of F_t is

$$F_t = \begin{pmatrix} \frac{\partial \mathbf{r}_t^{WC}}{\partial \mathbf{r}_{t-1}^{WC}} & 0 & \frac{\partial \mathbf{r}_t^{WC}}{\partial \mathbf{v}_{t-1}^{W}} & 0 \\ 0 & \frac{\partial \mathbf{q}_t^{WC}}{\partial \mathbf{q}_{t-1}^{WC}} & 0 & \frac{\partial \mathbf{q}_t^{WC}}{\partial \omega_{t-1}^{C}} \\ 0 & 0 & \frac{\partial \mathbf{v}_t^{W}}{\partial \mathbf{v}_{t-1}^{W}} & 0 \\ 0 & 0 & 0 & \frac{\partial \omega_t^{W}}{\partial \omega_{t-1}^{W}} \end{pmatrix} \tag{4.13}$$

where 0 is already inserted in place of the Jacobians submatrices where no influence is present. Of the remaining 6 non-zero Jacobians 4 can be dealt with easily, namely it holds

$$\frac{\partial \mathbf{r}_t^{WC}}{\partial \mathbf{r}_{t-1}^{WC}} = \frac{\partial \mathbf{v}_t^{W}}{\partial \mathbf{v}_{t-1}^{W}} = \frac{\partial \omega_t^{W}}{\partial \omega_{t-1}^{W}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

and

$$\frac{\partial \mathbf{r}_t^{WC}}{\partial \mathbf{v}_{t-1}^{W}} = \Delta t \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

which follows immediately from (4.8). To distinguish the real and the three imaginary parts a quaternion $\mathbf{q}$ will be denotes as $\mathbf{q} = (q_{t,r} \ q_{t,i} \ q_{t,j} \ q_{t,k})^T$. The quaternion $\mathbf{a}$ refers to the quaternion representing the rotation given by ω_{t-1}^C and Δt, thus $\mathbf{a}_{t-1} = \text{quat}(\omega_{t-1}^C \Delta t)$. Using these notations the 2 remaining

Jacobians $\frac{\partial \mathbf{q}_t^{WC}}{\partial \mathbf{q}_{t-1}^{WC}}$ and $\frac{\partial \mathbf{q}_t^{WC}}{\partial \omega_{t-1}^{C}}$ are defined as

$$\frac{\partial \mathbf{q}_t^{WC}}{\partial \mathbf{q}_{t-1}^{WC}} = \begin{pmatrix} a_{t-1,r} & -a_{t-1,i} & -a_{t-1,j} & -a_{t-1,k} \\ a_{t-1,i} & a_{t-1,r} & a_{t-1,k} & -a_{t-1,j} \\ a_{t-1,j} & -a_{t-1,k} & a_{t-1,r} & a_{t-1,i} \\ a_{t-1,k} & a_{t-1,j} & -a_{t-1,i} & a_{t-1,r} \end{pmatrix} \tag{4.14}$$

$$\begin{aligned} \frac{\partial \mathbf{q}_t^{WC}}{\partial \omega_{t-1}^{C}} &= \frac{\partial \mathbf{q}_t^{WC}}{\partial \mathrm{quat}\left(\omega_{t-1}^{C}\Delta t\right)} \frac{\partial \mathrm{quat}\left(\omega_{t-1}^{C}\Delta t\right)}{\partial \omega_{t-1}^{C}\Delta t} \\ &= \begin{pmatrix} q_{t-1,r} & -q_{t-1,i} & -q_{t-1,j} & -q_{t-1,k} \\ q_{t-1,i} & q_{t-1,r} & -q_{t-1,k} & q_{t-1,j} \\ q_{t-1,j} & q_{t-1,k} & q_{t-1,r} & -q_{t-1,i} \\ q_{t-1,k} & -q_{t-1,j} & -q_{t-1,i} & q_{t-1,r} \end{pmatrix} \begin{pmatrix} \frac{\partial a_{t-1,r}}{\partial \omega_{t-1,X}^{C}\Delta t} & \frac{\partial a_{t-1,r}}{\partial \omega_{t-1,Y}^{C}\Delta t} & \frac{\partial a_{t-1,r}}{\partial \omega_{t-1,Z}^{C}\Delta t} \\ \frac{\partial a_{t-1,i}}{\partial \omega_{t-1,X}^{C}\Delta t} & \frac{\partial a_{t-1,i}}{\partial \omega_{t-1,Y}^{C}\Delta t} & \frac{\partial a_{t-1,i}}{\partial \omega_{t-1,Z}^{C}\Delta t} \\ \frac{\partial a_{t-1,j}}{\partial \omega_{t-1,X}^{C}\Delta t} & \frac{\partial a_{t-1,j}}{\partial \omega_{t-1,Y}^{C}\Delta t} & \frac{\partial a_{t-1,j}}{\partial \omega_{t-1,Z}^{C}\Delta t} \\ \frac{\partial a_{t-1,k}}{\partial \omega_{t-1,X}^{C}\Delta t} & \frac{\partial a_{t-1,k}}{\partial \omega_{t-1,Y}^{C}\Delta t} & \frac{\partial a_{t-1,k}}{\partial \omega_{t-1,Z}^{C}\Delta t} \end{pmatrix} \end{aligned} \tag{4.15}$$

To solve $\frac{\partial \mathrm{quat}\left(\omega_{t-1}^{C}\Delta t\right)}{\partial \omega_{t-1}^{C}\Delta t}$ a closer look at the partial derivatives in the right-hand matrix in (4.15) is needed. First consider the partial derivatives of the real part of the quaternion. These are calculated by

$$\frac{\partial a_{t-1,r}}{\partial \omega_{t-1,\gamma}^{C}\Delta t} = -\sin\left(\|\omega_{t-1}^{C}\|\frac{\Delta t}{2}\right)\frac{\omega_{t-1,\gamma}^{C}}{\|\omega_{t-1}^{C}\|}\frac{\Delta t}{2}, \quad \gamma \in \{X, Y, Z\} \tag{4.16}$$

The partial derivatives in the remaining 3×3 submatrix of the right-hand matrix in (4.15) can be distinguished in two cases: Those were the imaginary part of quaternion $\mathbf{a}$ is derived by its corresponding angular velocity (found on the main diagonal of the 3×3 submatrix) and the other partial derivatives. The main diagonal partial derivatives are

$$\frac{\partial a_{t-1,\eta}}{\partial \omega_{t-1,\gamma}^{C}\Delta t} = \cos\left(\|\omega_{t-1}^{C}\|\frac{\Delta t}{2}\right)\frac{{\omega_{t-1,\gamma}^{C}}^2}{\|\omega_{t-1}^{C}\|^2}\frac{\Delta t}{2} + \sin\left(\|\omega_{t-1}^{C}\|\frac{\Delta t}{2}\right)\frac{1}{\|\omega_{t-1}^{C}\|}\left(1 - \frac{{\omega_{t-1,\gamma}^{C}}^2}{\|\omega_{t-1}^{C}\|^2}\right) \tag{4.17}$$

and $(\eta, \gamma) \in \{(i, X), (j, Y), (k, Z)\}$ The partial derivatives still left are of the form

$$\frac{\partial a_{t-1,\eta}}{\partial \omega_{t-1,\gamma}^{C}\Delta t} = \frac{\omega_{t-1,\zeta}^{C}\omega_{t-1,\gamma}^{C}}{\|\omega_{t-1}^{C}\|^2}\left(\cos\left(\|\omega_{t-1}^{C}\|\frac{\Delta t}{2}\right)\frac{\Delta t}{2} - \left(\frac{1}{\|\omega_{t-1}^{C}\|}\sin\left(\|\omega_{t-1}^{C}\|\frac{\Delta t}{2}\right)\right)\right) \tag{4.18}$$

with $(\eta, \gamma, \zeta) \in \{(i, Y, X), (i, Z, X), (j, X, Y), (j, Z, Y), (k, X, Z), (k, Y, Z)\}$. Though the partial derivatives (4.16), (4.17) and (4.18) may seem complicated at first glance they can be obtained by simple but somewhat tedious application of the common rules of derivation on function $\mathrm{quat}(\omega_{t-q}^{C}\Delta t)$, defined by equations (4.9) and (4.10).

In order to calculate the prediction of the covariance $\bar{\Sigma}_t$ there is still one piece missing, namely the additional noise R_t (see algorithm 4.1, line 2). R_t has an appearance similar to G_t:

$$R_t = \begin{pmatrix} R_t' & 0 \\ 0 & 0 \end{pmatrix}, \quad \dim\left(R_t'\right) = 13 \times 13, \quad \dim\left(R_t\right) = n_{\dim} \times n_{\dim} \tag{4.19}$$

where matrix R'_t is defined as

$$R'_t = \tilde{F}_t \, \mathbf{V}_{\max,t} \, \tilde{F}_t^T = \begin{pmatrix} \frac{\partial \mathbf{r}_t^{WC}}{\partial \mathbf{v}_{t-1}^W} & 0 \\ 0 & \frac{\partial \mathbf{q}_t^{WC}}{\partial \omega_{t-1}^C} \\ \frac{\partial \mathbf{v}_t^W}{\partial \mathbf{v}_{t-1}^W} & 0 \\ 0 & \frac{\partial \omega_t^W}{\partial \omega_{t-1}^W} \end{pmatrix} \mathbf{V}_{\max,t} \begin{pmatrix} \frac{\partial \mathbf{r}_t^{WC}}{\partial \mathbf{v}_{t-1}^W} & 0 \\ 0 & \frac{\partial \mathbf{q}_t^{WC}}{\partial \omega_{t-1}^C} \\ \frac{\partial \mathbf{v}_t^W}{\partial \mathbf{v}_{t-1}^W} & 0 \\ 0 & \frac{\partial \omega_t^W}{\partial \omega_{t-1}^W} \end{pmatrix}^T \tag{4.20}$$

with

$$\dim\left(\tilde{F}_t\right) = 13 \times 6, \quad \dim\left(\mathbf{V}_{\max,t}\right) = 6 \times 6$$

Where $\tilde{F}_t$ is the 13×6 submatrix, formed by columns 8 – 13 of matrix F_t (see equation (4.13)). If the maximal linear velocity is denoted by $v_{\max}^W$ and maximal angular velocity as $\omega_{\max}^C$ matrix $\mathbf{V}_{\max,t}$ is defined as

$$\mathbf{V}_{\max,t} = \begin{pmatrix} V_{\max,t} & 0 \\ 0 & \Omega_{\max,t} \end{pmatrix} \tag{4.21}$$

where $V_{\max,t}$ and $\Omega_{max,t}$ in turn are specified by

$$V_{\max,t} = \left(v_{\max}^W \Delta t\right)^2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \Omega_{\max,t} = \left(\omega_{\max}^C \Delta t\right)^2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Having completed the prediction of $\bar{\mu}_t$ and $\bar{\Sigma}_t$ next the measurement function needs to be considered.

4.4 Measurement Function

In this subsection the measurement function h and its Jacobian H_t will be presented. Both play an integral part in the EKF and therefore in VISUAL MONO-SLAM, since the H_t influences the Kalman gain K_t and the difference of the actual measurements z_t and the measurement prediction $h\left(\bar{\mu}_t\right)$ is used to eventually correct the predicted pose according to received sensor data.

According to the pinhole camera model (see chapter 2) an observed point on the image sensor defines a directional vector $\mathbf{h}^C = (h_x \;\; h_y \;\; h_z)^T$ in the camera coordinate system C. For better readability the the subscripts denoting time t will be omitted for the time being. For a point in XYZ encoding $\mathbf{x}_i$ directional vector $\mathbf{h}^C$ is specified by

$$\mathbf{h}_i^C = \mathbf{h}_{\mathrm{XYZ},i}^C = \mathbf{R}^{CW} \left(\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - \mathbf{r}^{WC} \right) \tag{4.22}$$

where $\mathbf{r}^{WC}$ denotes the position of the camera optical center in the world coordinate system and $\mathbf{R}^{CW}$ is the rotational matrix to transform vector $(\mathbf{x}_i - \mathbf{r}^{WC})$ into the camera coordinate system. $\mathbf{R}^{CW}$ can

be obtained by the inverse of quaternion $\mathbf{q}^{WC}$ which denotes orientation of the camera with respect to the world coordinate system. How a unit quaternion $\mathbf{q} = (q_r \;\; q_i \;\; q_j \;\; q_k)$ can be converted by function $\mathrm{q2r}\,(\mathbf{q})$ into a rotation matrix $\mathbf{R}$ denoting the same rotation is shown in the following equation:

$$\mathbf{R} = \mathrm{q2r}\,(\mathbf{q}) = \begin{pmatrix} q_r^2 + q_i^2 - q_j^2 - q_k^2 & -2q_rq_k + 2q_iq_j & 2q_rq_j + 2q_iq_k \\ 2q_rq_k + 2q_iq_j & q_r^2 - q_i^2 + q_j^2 - q_k^2 & -2q_rq_i + 2q_jq_k \\ -2qr_qj + 2q_iq_k & 2q_rq_i + 2q_jq_k & q_r^2 - q_i^2 - q_j^2 + q_k^2 \end{pmatrix} \tag{4.23}$$

Please note that in the special case of unit quaternions the inverse $\mathbf{q}^{-1}$ of a quaternion $\mathbf{q}$ is the same as the conjugate $\bar{\mathbf{q}}$ of the quaternion. Furthermore it makes no difference in the resulting matrix if first the the quaternion $\mathbf{q}^{WC}$ is conjugated and R^{CW} is constructed as $\mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)$ or if $\mathbf{q}^{WC}$ is converted by $\mathrm{q2r}\left(\mathbf{q}^{WC}\right)$ to R^{WC} first and R^{WC} is subsequently inverted to obtain R^{CW}. However in terms of computational efficiency first calculating the conjugate $\bar{\mathbf{q}}^{WC}$ is preferable, since this can be done by switching the sign of three double values which is less costly than matrix inversion.

The directional vector for a point $\mathbf{y}_i$ in inverse depth encoding is given using equation (4.5) in (4.22):

$$\mathbf{h}_i^C = \mathbf{h}_{\rho,i}^C = \mathbf{R}^{CW}\left(\rho_i\left(\begin{pmatrix} x_{c,i} \\ y_{c,i} \\ z_{c,i} \end{pmatrix} - \mathbf{r}^{WC}\right) + m\,(\theta_i, \phi_i)\right) \tag{4.24}$$

where $m\,(\theta_i, \phi_i)$ is defined in equation (4.6). It is noteworthy that even for points at infinity ($\rho_i = 0$) equation (4.24) can be evaluated without problems. In scenes observed by a camera points which show no parallax despite movements of the camera are considered to be at infinity or close to infinity. For these kind of points where $\rho_i \approx 0$ holds it follows by equation (4.24) that $\mathbf{h}^C \approx \mathbf{R}^{CW} m\,(\theta_i, \phi_i)$. That means while points at or close to infinity will not contribute to the estimation of the camera position $\mathbf{r}^{WC}$, they still can provide valuable information on the camera's orientation $\mathbf{q}^{WC}$ and the directional vector $m\,(\theta_i, \phi_i)$ of the associated point $\mathbf{y}_i$ in inverse depth coding. A scene with different point encoding is depicted in Figure 4.3.

Furthermore it should be mentioned that for a given point $\mathbf{y}_i$ in inverse depth and a XYZ-coded point $\mathbf{x}_j$ obtained by evaluating $\mathbf{y}_i$ according to equation (4.5) generally $h_{\mathrm{XYZ},i}^C \neq h_{\rho,j}^C$ will hold. This can be explained by the different length $h_{\mathrm{XYZ},i}^C$ and $h_{\rho,j}^C$ will have in the general case. However both vectors will specify the same direction (i.e. $\mathbf{h}_\rho^C / \|\mathbf{h}_\rho^C\| = \mathbf{h}_{\mathrm{XYZ}}^C / \|\mathbf{h}_{\mathrm{XYZ}}^C\|$).

Of course $\mathbf{h}^C$ is neither for points in XYZ encoding nor for points in inverse depth directly observed, but only the points projection on the image sensor. For a given point $\mathbf{f}_i,\ \mathbf{f} \in \{\mathbf{x}, \mathbf{y}\}$ and its associated directional vector $\mathbf{h}_i^C$ the undistorted image coordinates $(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ expected projection to be measured is given by

$$h_p\,(\mathbf{f}_i) = \begin{pmatrix} \mathrm{u}_{\mathrm{u},i} \\ \mathrm{v}_{\mathrm{u},i} \end{pmatrix} = \begin{pmatrix} u_0 - \frac{f}{d_u}\frac{h_{x,i}}{h_{z,i}} \\ v_0 - \frac{f}{d_v}\frac{h_{y,i}}{h_{z,i}} \end{pmatrix} \tag{4.25}$$

where (u_0, v_0) denotes the principal point, $d_u \times d_v$ is the physical size of a pixel and f the focal length of the camera.

To resemble the actual coordinates obtained by the camera the undistorted image coordinate $(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ needs to be distorted to be comparable to the actually received image. The distortion of image coordinate

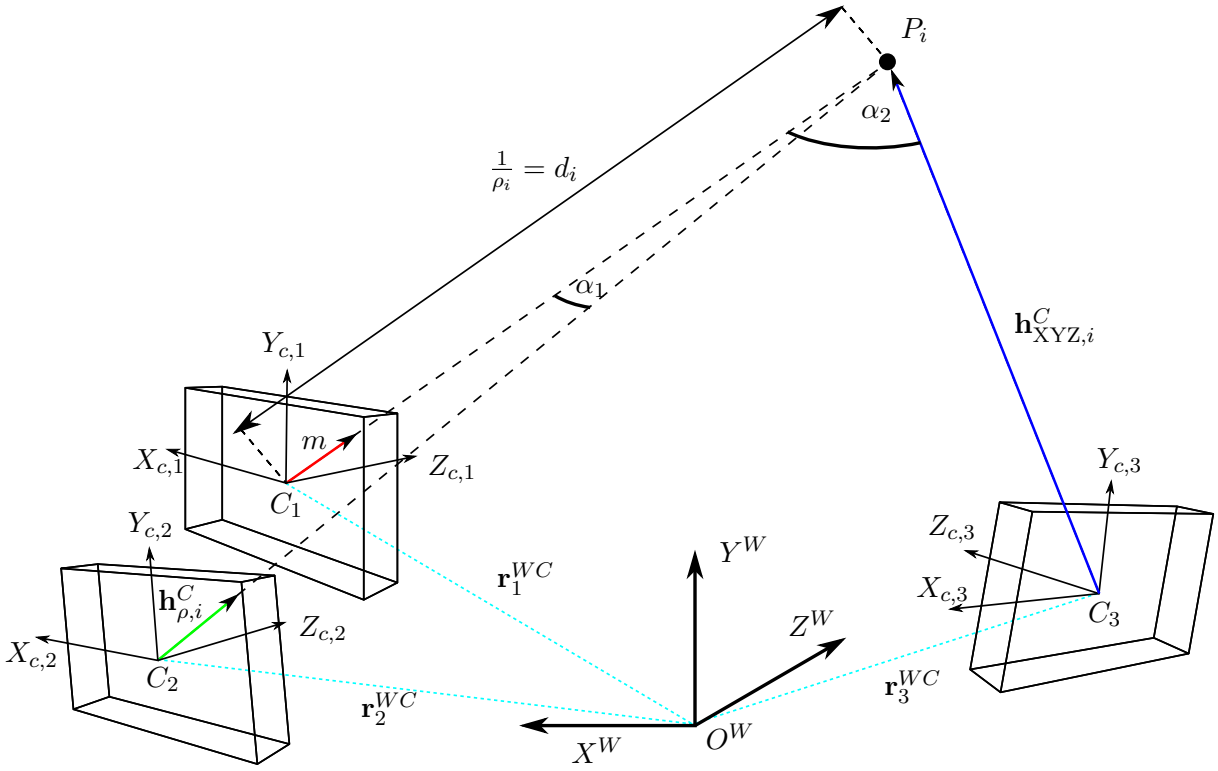


Figure 4.3: VISUAL MONO-SLAM coordinate system and feature parametrization. The origin of the right-hand world coordinate system is indicated by O^W. Three camera positions at times 1, 2 and 3 are depicted with optical centers $C_1 - C_3$. The corresponding translations $\mathbf{r}_t^{WC}$ are depicted in cyan. Different orientations $\mathbf{q}_t^{WC}$ are indicated by the wire frames symbolizing the camera case. Point $P_i = (x_i, y_i, z_i)^T$ is first observed from camera position C_1. Directional vector $m(\theta_i, \phi_i)$ is shown in red and along with the depth d_i of P_i. Between camera position C_1 and C_2 little parallax occurred (angle α_1 is small), thus P_i is in inverse depth coding and described by $(x_{c,i}, y_{c,i}, z_{c,i})^T + \frac{1}{\rho_i} m(\theta_i, \phi_i)$, where $(x_{c,i}, y_{c,i}, z_{c,i})^T$ denotes the position of C_1. Directional vector $\mathbf{h}^C_{\rho,i}$ (see equation (4.24)) is depicted in green. For larger parallax (α_2 and position C_3) coding is switched to XYZ. Directional vector $\mathbf{h}^C_{\mathrm{XYZ},i}$ (see (4.22)) is depicted in blue.

$(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ can be done according to the distortion function $h_d(h_p(\mathbf{f}_i))$ (see equation (2.17)), gaining distorted image coordinates $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$.

With the steps described above for each point in the state vector, whether in XYZ encoding or inverse depth, an expected measurement can be computed. These measurements need to be checked if they are expected to be inside the next image (i.e. $0 \leq \mathrm{u}_{\mathrm{d},i} < \mathrm{width}$ and $0 \leq \mathrm{v}_{\mathrm{d},i} < \mathrm{height}$ have to be fulfilled). All expected measurements inside the image compose $h(\bar{\mu}_t)$ of algorithm 4.1, line 4.

Next the Jacobian H_t of $h(\bar{\mu}_t)$ needs to be considered. In the remainder of this subsection the construction of a matrix H'_t will be discussed, that is closely related to H_t. In fact H_t will sometimes be identical with H'_t while sometimes H_t will consist of H'_t missing some rows. How H_t is actually constructed from H'_t will be shown in section 4.5. The dimension of H'_t depends on the number of measurements expected to be inside the next image. If the the set of all points in $\mathbf{x}_t$ is denoted as $\mathcal{F}_t = \{\mathbf{f}_{t,1}, \mathbf{f}_{t,2}, \ldots, \mathbf{f}_{t,n}\}$ with $n_{\dim} = 13 + \sum_{\mathbf{f}_{t,i}} \dim(\mathbf{f}_{t,i})$, $\mathbf{f}_{t,i} \in \mathcal{F}_t$ and let $\mathcal{M}_t \subseteq \mathcal{F}_t$, $|\mathcal{M}_t| = m_{\dim}$

be the subset containing all $\mathbf{f}_{t,i} \in \mathcal{F}_t$ which are expected to be on the image sensor, then the dimension of H'_t will be $2m_{\dim} \times n_{\dim}$. With the given notations Jacobian H'_t will have the following structure

$$H'_t = \begin{pmatrix} \frac{\partial h(\mathbf{g}_{t,1})}{\partial \bar{\mu}_t} \\ \frac{\partial h(\mathbf{g}_{t,2})}{\partial \bar{\mu}_t} \\ \vdots \\ \frac{\partial h\left(\mathbf{g}_{t,m_{\dim}}\right)}{\partial \bar{\mu}_t} \end{pmatrix}, \quad \dim\left(\frac{\partial h\left(\mathbf{g}_{t,i}\right)}{\partial \bar{\mu}_t}\right) = 2 \times n_{\dim}, \quad \mathbf{g}_{t,i} \in \mathcal{M}_t \tag{4.26}$$

where $\frac{\partial h(\mathbf{g}_{t,j})}{\partial \bar{\mu}_t} = \frac{\partial h(\mathbf{f}_{t,i})}{\partial \bar{\mu}_t}$ is defined as

$$\frac{\partial h\left(\mathbf{f}_{t,i}\right)}{\partial \bar{\mu}_t} = \begin{pmatrix} \frac{\partial h(\mathbf{f}_{t,i})}{\partial \mathbf{x}_{t,v}} & \underbrace{0 \cdots 0}_{\sum_{k<i} \dim\left(\mathbf{f}_{t,k}\right)} & \frac{\partial h(\mathbf{f}_{t,i})}{\partial \mathbf{f}_{t,i}} & \underbrace{0 \cdots 0}_{\sum_{i<l<n} \dim\left(\mathbf{f}_{t,l}\right)} \end{pmatrix} \tag{4.27}$$

where

$$\dim\left(\frac{\partial h\left(\mathbf{f}_{t,i}\right)}{\partial \mathbf{x}_{t,v}}\right) = 2 \times 13, \ \dim\left(\frac{\partial h\left(\mathbf{f}_{t,i}\right)}{\partial \mathbf{f}_{t,i}}\right) = 2 \times \dim\left(\mathbf{f}_{t,i}\right)$$

In the following $\frac{\partial h(\mathbf{f}_{t,i})}{\partial \mathbf{x}_{t,v}}$ and $\frac{\partial h(\mathbf{f}_{t,i})}{\partial \mathbf{f}_{t,i}}$ will be analyzed. To simplify notation the subscript t will be omitted again, but keep in mind that the values will nevertheless be dependent on t.

First a closer look at $\frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{x}_v}$ is taken. This matrix can be interpreted how the camera state $\mathbf{x}_v$ influences the outcome of the measurement function $h\left(\mathbf{f}_i\right)$ for a given point $\mathbf{f}_i$ in XYZ or inverse depth encoding. Since $\dim\left(\mathbf{x}_v\right) = 13$, the Jacobian of $h\left(\mathbf{f}_i\right)$ with respect to camera state $\mathbf{x}_v$ will be a 2×13 matrix. This matrix can be partitioned as

$$\frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{x}_v} = \begin{pmatrix} \frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{r}^{WC}} & \frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{q}^{WC}} & 0 \end{pmatrix} \tag{4.28}$$

where $\dim\left(\frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{r}^{WC}}\right) = 2 \times 3$ and $\dim\left(\frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{q}^{WC}}\right) = 2 \times 4$. The 2×6 zero matrix in $\frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{x}_v}$ just shows that $h\left(\mathbf{f}_i\right)$ is not dependent on velocities $\mathbf{v}^W$ and ω^C of camera state $\mathbf{x}_v$. For $\frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{r}^{WC}}$ and $\frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{q}^{WC}}$ in turn it holds that

$$\frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{r}^{WC}} = \frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{h}_i^C} \frac{\partial \mathbf{h}_i^C}{\partial \mathbf{r}^{WC}} \tag{4.29}$$

$$\frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{q}^{WC}} = \frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{h}_i^C} \frac{\partial \mathbf{h}_i^C}{\partial \mathbf{q}^{WC}} \tag{4.30}$$

Similarly $\frac{\partial h(\mathbf{f}_i)}{\partial \mathbf{f}_i}$ can be expressed as

$$\frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{f}_i} = \frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{h}_i^C} \frac{\partial \mathbf{h}_i^C}{\partial \mathbf{f}_i} \tag{4.31}$$

Of these Jacobians first measurement function $h\left(\mathbf{f}_i\right)$ with respect to the direction vector $\mathbf{h}_i^C$ of $\mathbf{f}_i$ in the camera coordinate will be examined. For undistorted image coordinates $\left(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i}\right)^T$ the measurement

function is defined by equation (4.25), but the final result of the measurement estimation is distorted by $h_d\left(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i}\right)^T$ (see equation (2.17)). Consequently this needs to be considered in the Jacobian, thus resulting in

$$\frac{\partial h\left(\mathbf{f}_i\right)}{\partial \mathbf{h}_i^C} = \frac{\partial h\left(\mathbf{f}_i\right)}{\partial h_p\left(\mathbf{f}_i\right)} \frac{\partial h_p\left(\mathbf{f}_i\right)}{\partial \mathbf{h}_i^C} \tag{4.32}$$

The first Jacobian $\frac{\partial h(\mathbf{f}_i)}{\partial h_p(\mathbf{f}_i)}$ can also be seen as $\frac{\partial h_d}{\partial(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})}$ which contains the derivatives of the distorted image coordinates $\left(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i}\right)^T$ with respect to the undistorted image coordinates $\left(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i}\right)^T$. The distorted image coordinates however cannot be directly computed from the undistorted coordinates, but are obtained by Newton-Raphson method (see equations (2.17), (2.18) and (2.21)). Analogously the Jacobian is not calculated directly but by inversion of Jacobian $\frac{\partial h_u}{\partial\left(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i}\right)}$, which is given by

$$\frac{\partial h_u}{\partial\left(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i}\right)} = \left(\begin{array}{c|c} \begin{array}{c}\left(1 + k_1 r_d^2 + k_2 r_d^4\right) + \\ \left(k_1 + 2k_2 r_d^2\right) 2\left(\left(\mathrm{u}_{\mathrm{d},i} - u_0\right) d_u\right)^2\end{array} & \begin{array}{c}\left(\mathrm{u}_{\mathrm{d},i} - u_0\right)\left(k_1 + 2k_2 r_d^2\right) \\ \left(2\left(\mathrm{v}_{\mathrm{d},i} - v_0\right) d_v^2\right)\end{array} \\ \hline \begin{array}{c}\left(\mathrm{v}_{\mathrm{d},i} - v_0\right)\left(k_1 + 2k_2 r_d^2\right) \\ \left(2\left(\mathrm{u}_{\mathrm{d},i} - u_0\right) d_u^2\right)\end{array} & \begin{array}{c}\left(1 + k_1 r_d^2 + k_2 r_d^4\right) + \\ \left(k_1 + 2k_2 r_d^2\right) 2\left(\left(\mathrm{v}_{\mathrm{d},i} - v_0\right) d_v\right)^2\end{array} \end{array}\right) \tag{4.33}$$

with r_d as defined in equation (2.18). From equation (4.25) the second Jacobian $\frac{\partial h_p(\mathbf{f}_i)}{\partial \mathbf{h}_i^C}$ can be directly calculated as

$$\frac{\partial h_p\left(\mathbf{f}_i\right)}{\partial \mathbf{h}_i^C} = \begin{pmatrix} -\frac{f}{d_u}\frac{1}{h_{z,i}} & 0 & \frac{f}{d_u}\frac{h_{x,i}}{h_{z,i}^2} \\ 0 & -\frac{f}{d_v}\frac{1}{h_{z,i}} & \frac{f}{d_v}\frac{h_{y,i}}{h_{z,i}^2} \end{pmatrix} \tag{4.34}$$

Computation of the remaining Jacobians $\frac{\partial \mathbf{h}_i^C}{\partial \mathbf{r}^{WC}}$, $\frac{\partial \mathbf{h}_i^C}{\partial \mathbf{q}^{WC}}$ and $\frac{\partial \mathbf{h}_i^C}{\partial \mathbf{f}_i}$ has to take into account whether $\mathbf{f}_i$ is in XYZ or inverse depth encoding. The encoding determines of which equation the Jacobian has to be calculated. For points in XYZ coding the partial derivatives of equation (4.22) have to be calculated while points in inverse depth require the partial derivatives of equation (4.24). First consider the dependency of the directional vector in the camera coordinate system $\mathbf{h}_i^C$ with respect to the position of the camera's optical center in the world coordinate system. This is given by

$$\frac{\partial \mathbf{h}_{\mathrm{XYZ},i}^C}{\partial \mathbf{r}^{WC}} = -\mathbf{R}^{CW} \tag{4.35}$$

for points $\mathbf{f}_i$ in XYZ coding, while for points in inverse depth

$$\frac{\partial \mathbf{h}_{\rho,i}^C}{\partial \mathbf{r}^{WC}} = -\rho_i \mathbf{R}^{CW} \tag{4.36}$$

needs to be computed. How the rotation matrix $\mathbf{R}^{CW}$ can be obtained from quaternion $\bar{\mathbf{q}}^{WC}$is shown in equation (4.23).

Next the Jacobian of the directional vector with respect to the rotation of the camera $\mathbf{q}^{WC}$ is considered. This proves to be a bit more complicated than (4.35) and (4.36). The rotation $\mathbf{q}^{WC}$ influences

$\mathbf{h}_i$ via rotation matrix $\mathbf{R}^{CW}$ (see (4.22) and (4.24)). Matrix $\mathbf{R}^{CW}$ in turn is constructed by function $\mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)$. This has to be considered for Jacobian $\frac{\partial \mathbf{h}_i^C}{\partial \mathbf{q}^{WC}}$ resulting in:

$$\frac{\partial \mathbf{h}_i^C}{\partial \mathbf{q}^{WC}} = \frac{\partial \mathbf{h}_i^C}{\partial \bar{\mathbf{q}}^{WC}} \frac{\partial \bar{\mathbf{q}}^{WC}}{\partial \mathbf{q}^{WC}} \tag{4.37}$$

To calculate $\frac{\partial \mathbf{h}_i^C}{\partial \bar{\mathbf{q}}^{WC}}$ the construction of $\mathbf{R}^{CW}$ by $\mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)$ (see equation (4.23)) and its partial derivatives with respect to real and the three imaginary parts of quaternion $\bar{\mathbf{q}}^{WC}$ need to be calculated. This results in matrix

$$\frac{\partial \mathbf{h}_i^C}{\partial \bar{\mathbf{q}}^{WC}} = \left(\frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_r^{WC}} \mathbf{d}_i \quad \frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_i^{WC}} \mathbf{d}_i \quad \frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_j^{WC}} \mathbf{d}_i \quad \frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_k^{WC}} \mathbf{d}_i \right) \tag{4.38}$$

where $\mathbf{d}_i$ denotes the direction vector $\mathbf{h}_i$ before it is rotated into the camera frame (see (4.22) and (4.24)). Hence $\mathbf{d}_i$ depends on whether $\mathbf{f}_i$ is in XZY or inverse depth coding and is defined as:

$$\mathbf{d}_{\mathrm{XYZ},i} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - \mathbf{r}^{WC} \tag{4.39}$$

for points encoded in XYZ. For points in inverse depth coding $\mathbf{d}_i$ is given by

$$\mathbf{d}_{\rho,i} = \rho_i \left(\begin{pmatrix} x_{c,i} \\ y_{c,i} \\ z_{c,i} \end{pmatrix} - \mathbf{r}^{WC} \right) + m\left(\theta_i, \phi_i\right) \tag{4.40}$$

According to equation (4.23) the partial derivatives of $\mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)$ with respects to the different parts of quaternion $\bar{\mathbf{q}}^{WC}$ are specified as

$$\frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_r^{WC}} = 2 \begin{pmatrix} \bar{q}_r^{WC} & -\bar{q}_k^{WC} & \bar{q}_j^{WC} \\ \bar{q}_k^{WC} & \bar{q}_r^{WC} & -\bar{q}_i^{WC} \\ -\bar{q}_j^{WC} & \bar{q}_i^{WC} & \bar{q}_r^{WC} \end{pmatrix} \tag{4.41}$$

$$\frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_i^{WC}} = 2 \begin{pmatrix} \bar{q}_i^{WC} & \bar{q}_j^{WC} & \bar{q}_k^{WC} \\ \bar{q}_j^{WC} & -\bar{q}_i^{WC} & -\bar{q}_r^{WC} \\ \bar{q}_k^{WC} & \bar{q}_r^{WC} & -\bar{q}_i^{WC} \end{pmatrix} \tag{4.42}$$

$$\frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_j^{WC}} = 2 \begin{pmatrix} -\bar{q}_j^{WC} & \bar{q}_i^{WC} & \bar{q}_r^{WC} \\ \bar{q}_i^{WC} & \bar{q}_j^{WC} & \bar{q}_k^{WC} \\ -\bar{q}_r^{WC} & \bar{q}_k^{WC} & -\bar{q}_j^{WC} \end{pmatrix} \tag{4.43}$$

$$\frac{\partial \mathrm{q2r}\left(\bar{\mathbf{q}}^{WC}\right)}{\partial \bar{q}_k^{WC}} = 2 \begin{pmatrix} -\bar{q}_k^{WC} & -\bar{q}_r^{WC} & \bar{q}_i^{WC} \\ \bar{q}_r^{WC} & -\bar{q}_k^{WC} & \bar{q}_j^{WC} \\ \bar{q}_i^{WC} & \bar{q}_j^{WC} & \bar{q}_k^{WC} \end{pmatrix} \tag{4.44}$$

The missing Jacobian $\frac{\partial \bar{\mathbf{q}}^{WC}}{\partial \mathbf{q}^{WC}}$ of the conjugate with respect to the quaternion is luckily quite simple:

$$\frac{\partial \bar{\mathbf{q}}^{WC}}{\partial \mathbf{q}^{WC}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{4.45}$$

Finally the Jacobian of the directional vector $\mathbf{h}_i^C$ with respect to the estimation of point $\mathbf{f}_i$ is needed to complete the description of H_t'. For points in XYZ encoding this is is given by

$$\frac{\partial \mathbf{h}_{\mathrm{XYZ},i}^C}{\partial \mathbf{f}_i} = \mathbf{R}^{CW}, \ \dim\left(\frac{\partial \mathbf{h}_{\mathrm{XYZ},i}^C}{\partial \mathbf{f}_i}\right) = 3 \times 3 \tag{4.46}$$

while the Jacobian for inverse depth encoded points is defined as

$$\frac{\partial \mathbf{h}_{\rho,i}^C}{\partial \mathbf{f}_i} = \begin{pmatrix} \frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial x_{c,i}, y_{c,i}, z_{c,i}} & \frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial \theta_i} & \frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial \phi_i} & \frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial \rho_i} \end{pmatrix}, \ \dim\left(\frac{\partial \mathbf{h}_{\rho,i}^C}{\partial \mathbf{f}_i}\right) = 3 \times 6 \tag{4.47}$$

with

$$\frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial x_{c,i}, y_{c,i}, z_{c,i}} = \rho_i \mathbf{R}^{CW}, \qquad \frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial \theta_i} = \mathbf{R}^{CW} \begin{pmatrix} \cos\theta_i \cos\phi_i \\ 0 \\ -\sin\theta_i \cos\phi_i \end{pmatrix}$$

$$\frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial \phi_i} = \mathbf{R}^{CW} \begin{pmatrix} -\sin\theta_i \sin\phi_i \\ -\cos\phi_i \\ -\cos\theta_i \sin\phi_i \end{pmatrix}, \qquad \frac{\partial \mathbf{h}_{\rho,\mathbf{i}}^{\mathbf{C}}}{\partial \rho_i} = \mathbf{R}^{CW} \left(\begin{pmatrix} x_{c,i} \\ y_{c,i} \\ z_{c,i} \end{pmatrix} - \mathbf{r}^{WC} \right)$$

The different dimensionality of $\frac{\partial \mathbf{h}_{\mathrm{XYZ},i}^C}{\partial \mathbf{f}_i}$ and $\frac{\partial \mathbf{h}_{\rho,i}^C}{\partial \mathbf{f}_i}$ is caused by the different dimension of the encoding (3 dimensions for XYZ and 6 for inverse depth) and is also reflected in the structure of $\frac{\partial h(\mathbf{f}_{t,i})}{\partial \bar{\mu}_t}$ (see equation (4.27)).

Keep in mind that the above computations are necessary for each point $\mathbf{f}_{t,i} \in \mathcal{M}_t$ (i.e. all points that are predicted to be on the image sensor at time t). While some Jacobians on the lower level (like (4.41) – (4.44), (4.45)) can be calculated once and be reused for all points $\mathbf{f}_{t,i} \in \mathcal{M}_t$, most Jacobians are dependent on properties of $\mathbf{f}_i$ and need to be computed for each point individually. Furthermore for each iteration of the EKF the Jacobians, apart from equation (4.45), need to be calculated anew, since they are dependent on time t.

4.5 Feature Matching

Having discussed measurement function $h(\bar{\mu}_t)$ to gain expected measurements in section 4.4 the actual matching of features and the construction of H_t from H_t' will be discussed next. How the actual matching of features is done depends whether image patches or feature descriptors are used to characterize the

appearance of a feature in an image (see subsections 3.1.1, 3.1.2 and 3.2.2). However in either case the knowledge inherent in the EKF can be exploited to reduce computational effort. The basic idea is quite simple and intuitive: From measurement function $h(\bar{\mu}_t)$ an expected position $(\bar{\mathrm{u}}_{\mathrm{d},i}, \bar{\mathrm{v}}_{\mathrm{d},i})^T$ for feature $\mathbf{f}_i$ is known. Instead of trying to match features $\mathbf{f}_i$ with every possible location of the whole image, it is much more useful to search in an area around image coordinate $(\bar{\mathrm{u}}_{\mathrm{d},i}, \bar{\mathrm{v}}_{\mathrm{d},i})^T$. Since the covariance in the EKF encodes information about the uncertainty of a feature position, this information should be regarded and influence the size of the search area: If the feature's 3D position (see (4.3) or (4.5)) is well known (i.e. the uncertainty indicated by $\bar{\Sigma}_t$ for $\mathbf{f}_i$ is small, the resulting search area should be small as well, thus reducing computational load and saving time). Consequently for features where the uncertainty for the estimated location is high the size of the search region should increase. In case of a successful detection of feature $\mathbf{f}_i$ in its specific search region, the actually measured position $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ in the image will be part of the measurement z_t (see algorithm 4.1, line 4). If the feature could not be matched in its search region, it will not contribute to the correction of the pose estimation (i.e. it will not be part of z_t and its estimation will be removed from $h(\bar{\mu}_t)$. Analogously the 2 rows $\frac{\partial h(\mathbf{f}_{t,i})}{\partial \bar{\mu}_t}$ are removed from H'_t). Now that the basic idea has been outlined a closer look at the implementing mathematics is taken:

To get an idea about the uncertainty of the estimated position of feature $\mathbf{f}_i$ in the image and subsequently about the size of its search region the *innovation covariance* S_t is calculated. The innovation covariance is defined as

$$S_t = H'_t \bar{\Sigma}_t {H'_t}^T + Q'_t \tag{4.48}$$

where $\bar{\Sigma}_t$ is the predicted covariance (see algorithm 4.1, line 2 and subsection 4.3), H'_t is calculated according to equation (4.26) and Q'_t denotes a matrix to modulate sensor noise in pixels. The dimensions of the matrices in equation (4.48) are

$$\begin{aligned} \dim(S_t) = 2m'_{\dim} \times 2m'_{\dim},\ \dim\left(H'_t\right) = 2m'_{\dim} \times n_{\dim} \\ \dim\left(\bar{\Sigma}_t\right) = n_{\dim} \times n_{\dim},\ \dim\left(Q'_t\right) = 2m'_{\dim} \times 2m'_{\dim} \end{aligned}$$

where $n_{\dim}$ denotes the dimension of the current state $\mathbf{x}_t$ and $m'_{\dim}$ is the number of features predicted to be on the image sensor at time t. Apart from Q'_t all terms in (4.48) have been previously discussed, so a closer look at Q'_t is sufficient to completely determine innovation covariance S_t. Noise matrix Q'_t is given as

$$Q'_t = \sigma_R^2 \mathbb{1} \tag{4.49}$$

where $\mathbb{1}$ denotes an identity matrix with $\dim(\mathbb{1}) = 2m'_{\dim} \times 2m'_{\dim}$ and σ_R^2 specifies the squared standard deviation of image noise (i.e. σ_R models that a point might be displayed at another than its ideal position). For many cameras $\sigma_R = 1$ may be assumed, but there for some cameras other standard deviations need to be used.

Taking a closer look at S_t one might notice that for all points $\mathbf{g}_i \in \mathcal{M}_t$ the uncertainties regarding their image coordinates are found at the main diagonal (defining $\mathcal{M}_t$ as in equation (4.26)). Considering the appearance of the predicted covariance $\bar{\Sigma}_t$ and of H'_t the main diagonal of S_t contains the variance in U and V direction for each feature $\mathbf{g}_i \in \mathcal{M}_t$. This can be employed to define the boundaries $(b_{u,i}, b_{v,i})$ of the search region for each feature $\mathbf{g}_i \in \mathcal{M}_t$ in U and V direction as

$$\begin{pmatrix} b_{u,i} \\ b_{v,i} \end{pmatrix} = 2 \begin{pmatrix} \sqrt{s_{t,(2i-1,2i-1)}} \\ \sqrt{s_{t,(2i,2i)}} \end{pmatrix} \tag{4.50}$$

where $s_{t,(i,j)}$ denotes the element at position (i,j) of matrix S_t. Since in an EKF everything is modeled as Gaussian, 2 times the standard deviation constitutes the 95% confidence interval of the around the estimated mean. By equation (4.50) we get just these confidence intervals. The search region can be either modeled as an ellipse with the estimated points location $(\bar{\mathrm{u}}_{\mathrm{d},i}, \bar{\mathrm{v}}_{\mathrm{d},i})^T$ as its center, radius $b_{u,i}$ in U direction and radius $b_{v,i}$ in V direction. Or the more simple approach would be a rectangle with upper left corner $(\bar{\mathrm{u}}_{\mathrm{d},i} - b_{u,i}, \bar{\mathrm{v}}_{\mathrm{d},i} - b_{v,i})^T$ and lower right corner $(\bar{\mathrm{u}}_{\mathrm{d},i} + b_{u,i}, \bar{\mathrm{v}}_{\mathrm{d},i} + b_{v,i})^T$.

If the feature can be matched inside this search region using an appropriate comparison the feature is considered to be successfully matched. The image coordinates $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ denoting the position of the match are appended to measurement vector z_t and the feature will contribute towards the correction of the estimates mean $\bar{\mu}_t$ and covariance $\bar{\Sigma}_t$. Otherwise the predicted measurement for $\mathbf{g}_i$ will be removed from $h(\bar{\mu}_t)$, since no corresponding measurement in z_t is present and subsequently rows $2i-1$ and $2i$ are deleted from matrix H'_t. If all features $\mathbf{g}_i \in \mathcal{M}_t$ were successfully or unsuccessfully matched the remains of H'_t coincides with the final Jacobian H_t of the predicted measurements $h(\bar{\mu}_t)$. Having completed the calculations discussed in this subsection and previously in 4.3 and 4.4 the computation of the Kalman gain and the update step can be addressed next.

Note that the innovation covariance is not needed for the basic EKF algorithm, but is frequently used in applications of the EKF like EKF-SLAM.

4.6 Update Step

Having done all the tedious preparatory work in sections 4.3, 4.4 and 4.5 the actual calculation of the Kalman gain K_t and the update can be kept rather simple. First the Kalman gain (see algorithm 4.1, line 3) will be discussed. To recap that line of the algorithm the Kalman gain is stated as

$$K_t = \bar{\Sigma}_t H_t^T \left(H_t \bar{\Sigma}_t H_t^T + Q_t\right)^{-1} \tag{4.51}$$

with

$$\dim(K_t) = n_{\dim} \times 2m_{\dim},\ \dim(H_t) = 2m_{\dim} \times n_{\dim}$$
$$\dim(\Sigma_t) = n_{\dim} \times n_{\dim},\ \dim(Q_t) = 2m_{\dim} \times 2m_{\dim}$$

where $n_{\dim} = \dim(\mathbf{x}_t)$ denotes the dimension of the current state vector, and $m_{\dim}$ denotes the number of matched features.

Apart from the noise matrix Q_t all terms in equation (4.51) were introduced and discussed in previous chapters. Since matrix Q_t denotes image noise and is apart from the dimensions identical with matrix Q'_t (see (4.49)) no further description of Q_t is needed and thus the Kalman gain is complete.

The two update steps can now be computed straight forward according to algorithm 4.1, lines 4 and 5. Afterwards the resulting mean μ_t and covariance Σ_t need to be tweaked a bit which is not part of the EKF algorithm. So strictly speaking one could argue that in VISUAL MONO-SLAM no pure EKF is used. The need for a post-processing of μ_t and Σ_t arises from the usage of a quaternion $\mathbf{q}^{WC}$ to denote the orientation of the camera. Orientations are only represented by unit quaternions and after the update step it is not guaranteed that the estimated mean of $\mathbf{q}^{WC}$ still denotes a unit quaternion. Therefore $\mathbf{q}^{WC}$ will be normalized according to

$$\mathrm{norm}(\mathbf{q}) = \left(\frac{q_r}{\sqrt{q_r^2+q_i^2+q_j^2+q_k^2}} \quad \frac{q_i}{\sqrt{q_r^2+q_i^2+q_j^2+q_k^2}} \quad \frac{q_j}{\sqrt{q_r^2+q_i^2+q_j^2+q_k^2}} \quad \frac{q_k}{\sqrt{q_r^2+q_i^2+q_j^2+q_k^2}}\right) \tag{4.52}$$

The mean with the normalized quaternion will be referred to as μ_t'.

Consequently the covariance matrix Σ_t needs to be adopted as well. If the current dimension of the covariance is $\dim(\Sigma_t) = n_{\dim} \times n_{\dim}$, then the adopted covariance is given by

$$\Sigma_t' = \begin{pmatrix} \mathbb{1}_a & 0 & 0 \\ 0 & \frac{\partial \text{norm}(\mathbf{q}^{WC})}{\partial \mathbf{q}^{WC}} & 0 \\ 0 & 0 & \mathbb{1}_b \end{pmatrix} \Sigma_t \begin{pmatrix} \mathbb{1}_a & 0 & 0 \\ 0 & \frac{\partial \text{norm}(\mathbf{q}^{WC})}{\partial \mathbf{q}^{WC}} & 0 \\ 0 & 0 & \mathbb{1}_b \end{pmatrix}^T \tag{4.53}$$

where the covariance incorporating the normalization of $\mathbf{q}^{WC}$ is denoted as Σ_t', $\mathbb{1}_a$ and $\mathbb{1}_b$ label unit matrices of dimensions $\dim(\mathbb{1}_a) = 3 \times 3$ and $\dim(\mathbb{1}_b) = n_{\dim} - 7 \times n_{\dim} - 7$ and 0 indicates zero matrices of appropriate size.

The Jacobian $\frac{\partial \text{norm}(\mathbf{q}^{WC})}{\partial \mathbf{q}^{WC}}$ used in equation (4.53) can be determined from equation (4.52) as

$$\frac{\partial \text{norm}(\mathbf{q})}{\partial \mathbf{q}} = \left(q_r^2 + q_i^2 + q_j^2 + q_z^2\right)^{-\frac{3}{2}} \mathbf{Q} \tag{4.54}$$

with

$$\mathbf{Q} = \begin{pmatrix} q_i^2 + q_j^2 + q_z^2 & -q_r q_i & -q_r q_j & -q_r q_k \\ -q_r q_i & q_r^2 + q_j^2 + q_k^2 & -q_i q_j & -q_i q_k \\ -q_r q_j & -q_i q_j & q_r^2 + q_i^2 + q_k^2 & -q_j q_k \\ -q_r q_k & -q_i q_k & -q_j q_k & q_r^2 + q_i^2 + q_j^2 \end{pmatrix}$$

The modified mean μ_t' and covariance Σ_t' will be used as the old estimation in the next EKF step.

4.7 Feature Linearity

As mentioned in previous sections features in inverse depth encoding are linear in their depth estimation if initialized without prior knowledge, while XYZ encoded features are not. This makes the former encoding suitable for newly initialized features while the latter should not be used to represent points of unknown depth in an EKF. This section will substantiate these allegations with mathematical proof. The analysis presented here elaborates on the one found in [12, 13].

The approach used by Civera et al. in [12, 13] shows some similarity to the derivation of the EKF from the standard Kalman filter (see section 4.1). Civera et al. examine the behaviour of a Gaussian random variable $\chi \sim G\left(\mu_\chi, \sigma_\chi^2\right)$ through some function g. The image of χ will be a random variable denoted as ψ. If function g is linear ψ can also be approximated as Gaussian: $\psi \sim G\left(\mu_\psi, \sigma_\psi^2\right)$ with

$$\begin{aligned} \mu_\psi &= g\left(\mu_\chi\right) \\ \sigma_\psi^2 &= \left.\frac{\partial g}{\partial \chi}\right|_{\mu_\chi} \sigma_\chi^2 \left.\frac{\partial g}{\partial \chi}\right|_{\mu_\chi}^T \end{aligned}$$

where $\left.\frac{\partial f}{\partial \chi}\right|_{\mu_\chi}$ denotes the Jacobian of g with respect to χ, evaluated at mean μ_χ. The interval in which function g has to be linear in order to allow for this approximation to be correct depends on the size of

σ_χ, of course. The larger σ_χ, the larger the linear interval of g around mean μ_χ needs to be. To determine if Gaussians are properly mapped by function g it is sensible to analyse the behaviour of g in the 95% confidence region around μ_χ, given by interval $[\mu_\chi - 2\sigma_\chi, \mu_\chi + 2\sigma_\chi]$.

Linearity of a function may be assumed if the first order derivative of that function is constant. As in the linearization of the transition function of the EKF (see equation (4.1)) first order Tailor approximation is used to determine the first order derivative of function g:

$$\frac{\partial g}{\partial \chi}(\mu_\chi + \Delta\chi) \approx \left.\frac{\partial g}{\partial \chi}\right|_{\mu_\chi} + \left.\frac{\partial^2 g}{\partial \chi^2}\right|_{\mu_\chi} \Delta\chi \tag{4.55}$$

where $\left.\frac{\partial^2 g}{\partial \chi^2}\right|_{\mu_\chi}$ denotes the second order derivative. To analyze the linearity of function g Civera et al. propose to compare the derivative at the center of the confidence region, namely at μ_χ with the derivative at the extrema ($\mu_\chi \pm 2\sigma_\chi$) of the interval. The first is simply given by

$$\left.\frac{\partial g}{\partial \chi}\right|_{\mu_\chi} \tag{4.56}$$

while the derivative at the extrema can be expressed as

$$\left.\frac{\partial g}{\partial \chi}\right|_{\mu_\chi} + \left.\frac{\partial^2 g}{\partial \chi^2}\right|_{\mu_\chi} 2\sigma_\chi \tag{4.57}$$

according to the approximation in (4.55).

Combining (4.56) and (4.57) a dimensionless linearity index L is proposed in [12, 13]. Linearity index L can be used as a measure of the linearity of a function in interval $[\mu_\chi - 2\sigma_\chi, \mu_\chi + 2\sigma_\chi]$ and is defined as

$$L = \left| \frac{\left.\frac{\partial^2 g}{\partial \chi^2}\right|_{\mu_\chi} 2\sigma_\chi}{\left.\frac{\partial g}{\partial \chi}\right|_{\mu_\chi}} \right| \tag{4.58}$$

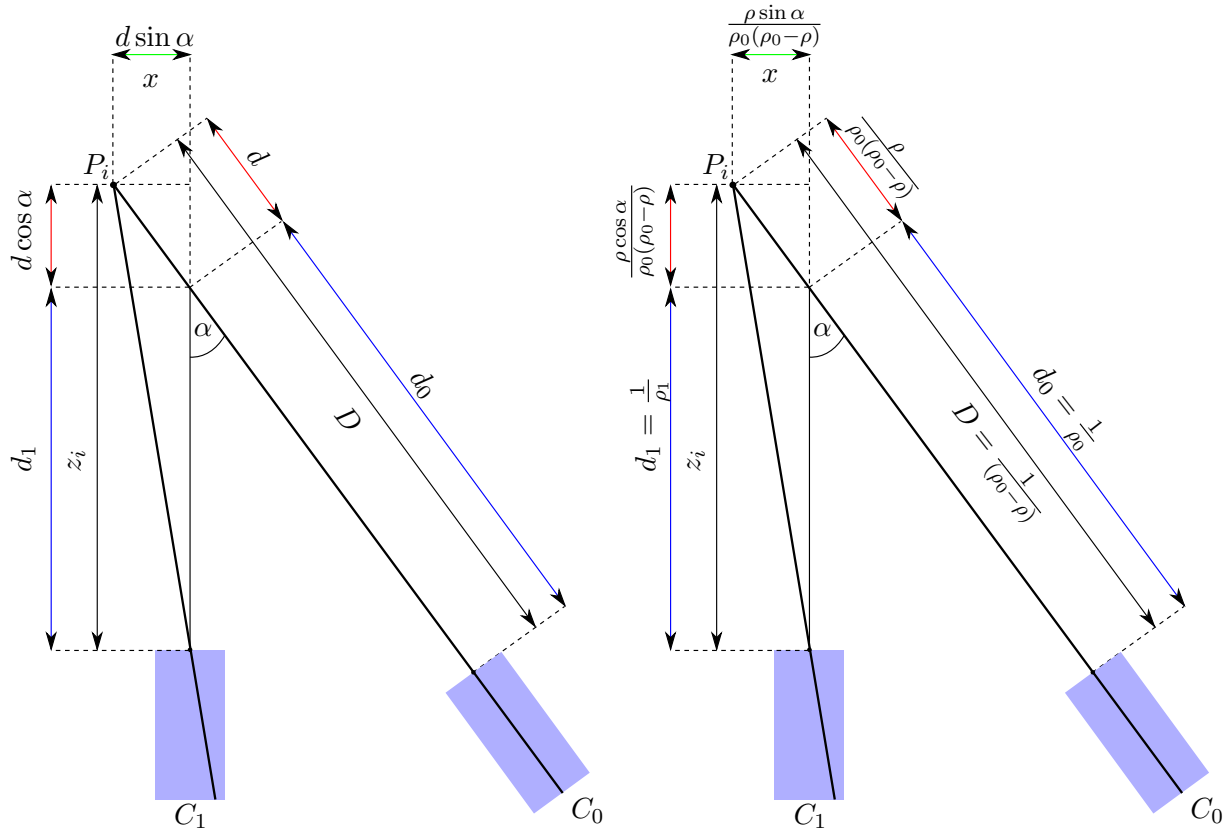
where the numerator consists of the absolute value of the difference between equations (4.56) and (4.57). To gain a dimensionless normalized measure, the denominator consists of the derivative evaluated at the mean (equation (4.56)). Linearity of a function may be assumed if $L \approx 0$ holds, since this implies that $\left.\frac{\partial^2 g}{\partial \chi^2}\right|_{\mu_\chi} 2\sigma_\chi \approx 0$ which in turn means that the first order derivative at mean μ_χ does not significantly differ from the derivative at the endpoints $\mu_\chi \pm 2\sigma_\chi$ of the 95% confidence region.

Now that a measure of linearity for a given function g is available by linearity index L next it will be shown, how L can be used to analyze linearity of XYZ and inverse depth encoding.

Remembering the pinhole camera model (see section 2.1) the location of a point $P_i = (x_i, y_i, z_i)$ is projected according to

$$u = \frac{x_i}{z_i} f \tag{4.59}$$

where f denotes the focal length. Without loss of generality it may be assumed $f = 1$, since other values of f would only scale the following considerations. To analyze the behaviour of XYZ and inverse



(a) XYZ coding **(b)** inverse depth coding

Figure 4.4: Uncertainty propagation from scene point P_i to the image. **(a)** shows the scene for XYZ encoding, while **(b)** displays the same scenario using inverse depth coding.

depth coding two cameras C_0 and C_1 with focal length 1 at different positions are used. Both cameras observe the same point P_i. Camera C_0 will observe the point for the first time and initialize the estimated depth with a default value either d_0 or $\rho_0 = \frac{1}{d_0}$ dependent on the encoding method. The parallax angle α between the rays from the optical centers of C_0 and C_1 to point P_i is approximated by the angle between the optical axes of the cameras. The estimated distance d_1 for camera C_1 is therefore given by the intersection of the two optical axes. Figure 4.4 depicts this setup. In the following the linearity of the measurement equation (4.59) will be analyzed by the linearity index defined in equation (4.58) for both types of point encoding.

First consider P_i in XYZ coding. Depth will be initialized with value d_0 and the depth error will be labeled d (see Figure 4.4a). The location error is assumed to be Gaussian with mean 0 (i.e. $d \sim G\left(0, \sigma_d^2\right)$) and the actual depth is given by $D = d_0 + d$. From Figure 4.4a it can easily be deduced that

$$x_i = d \sin \alpha$$
$$z_i = d_1 + d \cos \alpha$$

hold. This allows for

$$u = \frac{d \sin \alpha}{d_1 + \cos \alpha} \tag{4.60}$$

so that u can be interpreted as a function dependent on Gaussian d. Subsequently the linearity index for XYZ coding can be calculated. First the first and second order derivatives of u as defined in (4.60) with respect to d are given by

$$\frac{\partial u}{\partial d} = \frac{d_1 \sin \alpha}{(d_1 + d \cos \alpha)^2} \tag{4.61}$$

$$\frac{\partial^2 u}{\partial d^2} = \frac{-2 d_1 \sin \alpha \cos \alpha}{(d_1 + d \cos \alpha)^3} \tag{4.62}$$

Applying equations (4.61) and (4.62) to definition of the linearity index L (see equation (4.58)) the linearity index for XYZ coding L_d is given as:

$$L_d = \left| \frac{\frac{\partial^2 u}{\partial d^2}\Big|_{d=0} 2\sigma_d}{\frac{\partial u}{\partial d}\Big|_{d=0}} \right| = \frac{4\sigma_d}{d_1} \left|\cos(\alpha)\right| \tag{4.63}$$

Accordingly inverse depth coding can be analyzed. In this case the initial depth estimation will be $d_0 = \frac{1}{\rho_0}$, while the actual depth will be denoted as $D = \frac{1}{\rho_0 - \rho}$, where ρ is assumed to be Gaussian with mean 0 ($\rho \sim G\left(0, \sigma_\rho^2\right)$). It is also assumed that $D = d_0 + d \Leftrightarrow d = D - d_0$ holds for depth location error d. Therefore d is specified as

$$d = D - d_0 = \frac{\rho}{\rho_0 (\rho_0 - \rho)}$$

Similarly to XYZ coding x_i and z_i can be expressed by ρ, ρ_0, α and d_1 (see Figure 4.4b):

$$x_i = d \sin \alpha = \frac{\rho \sin \alpha}{\rho_0 (\rho_0 - \rho)}$$
$$z_i = d_1 + d \cos \alpha = d_1 + \frac{\rho \cos \alpha}{\rho_0 (\rho_0 - \rho)}$$

Now u can be expressed as a function dependent on Gaussian ρ and is given by

$$u = \frac{\rho \sin \alpha}{\rho_0 d_1 (\rho_0 - \rho) + \rho \cos \alpha} \tag{4.64}$$

First and second order derivative of u in equation (4.64) yields

$$\frac{\partial u}{\partial \rho} = \frac{\rho_0^2 d_1 \sin \alpha}{(\rho_0 d_1 (\rho_0 - \rho) + \rho \cos \alpha)^2} \tag{4.65}$$

$$\frac{\partial^2 u}{\partial \rho^2} = \frac{-2\rho_0^2 d_1 (\cos \alpha - d_1 \rho_0)}{(\rho_0 d_1 (\rho_0 - \rho) + \rho \cos \alpha)^3} \tag{4.66}$$

From equations (4.65) and (4.66) the linearity index L_ρ for inverse depth is defined as:

$$L_\rho = \left| \frac{\frac{\partial^2 u}{\partial \rho^2}\Big|_{\rho=0} 2\sigma_\rho}{\frac{\partial u}{\partial \rho}\Big|_{\rho=0}} \right| = \frac{4\sigma_\rho}{\rho_0} \left| 1 - \frac{\rho_1}{\rho_0} \cos\alpha \right| = \frac{4\sigma_\rho}{\rho_0} \left| 1 - \frac{d_0}{d_1} \cos\alpha \right| \tag{4.67}$$

Now that linearity indices are available for both XYZ and inverse depth coding a closer look at their implications should be taken. For XYZ coding the 95% confidence region for the depth is defined by the initial depth estimation d_0 and the standard deviation of the depth error σ_d as $[d_0 - 2\sigma_d, d_0 + 2\sigma_d]$. Since this region should cover a large interval of depth values σ_d needs to be quite large. Note that 0 depth may be included into this confidence region, but infinity cannot be included. For inverse depth the 95% confidence interval is specified as $[\frac{1}{\rho_0+2\sigma_\rho}, \frac{1}{\rho_0-2\sigma_\rho}]$, where ρ_0 labels the initial depth estimation in inverse depth and σ_ρ is the standard deviation of inverse depth error ρ. Since σ_ρ appears in the denominator small values are sufficient to express a large confidence region. Note that while 0 depth cannot be included in this confidence region infinity is included if $0 \in [\rho_0 - 2\sigma_\rho, \rho_0 - 2\sigma_\rho]$ holds.

If the VISUAL MONO-SLAM application is considered it is reasonable to assume that the observed parallax is small, which implies $\alpha \approx 0 \Rightarrow \cos\alpha \approx 1$ and $\frac{d_0}{d_1} \approx 1$. For these valid assumptions linearity indices L_d and L_ρ are consulted. According to equation (4.63) in this case $L_d \approx \frac{4\sigma_d}{d_1}$ holds. Since σ_d needs to be large for the confidence region to cover a large interval of depth values L_d in turn will also be large which indicates no linearity in the specified interval. For inverse depth equation (4.67) can be approximated as $L_\rho \approx 0$, since $1 - \frac{d_0}{d_1}\cos\alpha \approx 0$ holds under the given assumptions. Thus for inverse depth coding the measurement equation (4.59) may be assumed linear.

For repeatedly observed points where parallax angle α increases, depth estimation becomes more accurate, which means that σ_d and σ_ρ respectively may become smaller. Large parallax angles and small standard deviation σ_d mean that L_d will get smaller. That implies that points with low depth uncertainty at high parallax may be safely encoded by XYZ representation, since for these points the measurement equation may be assumed linear. On the other hand L_ρ will still be small for such features, since the increase of term $\left|1 - \frac{d_0}{d_1}\cos\alpha\right|$ will be compensated by the decrease of $\frac{4\sigma_\rho}{\rho_0}$ (keep in mind that ρ_0 is constant and the small values of σ_ρ may will further decrease for small depth uncertainty). So inverse depth coding is suitable for both, newly initialized features at low parallax and features with low depth uncertainty at high parallax.

4.8 Feature Initialization

As discussed in section 4.7 the XYZ parametrization lacks linearity for low parallax feature with large depth uncertainty. Inference on the depth of a feature is not possible from one single observation, but can only be gained by multiple observations if the parallax is large enough.

This however would mean that potential features would have to be observed over a certain time until the uncertainty concerning their depth is reasonably low, before they can be added into the EKF. Such an approach is somewhat undesirable, since the potential features need to be observed like real features

before they are added to the state vector $\mathbf{x}_t$ or discarded. Thus while needing about as much computational effort as features added to the EKF these feature candidates to not contribute to the estimation of the camera state or the estimation of other features. Secondly one could argue that such an initialization phase is not part of "pure" EKF and should be avoided, but the validity of this argument is doubtful, since in the update step of VISUAL MONO-SLAM the EKF has already been tinkered with (see section 4.6). While it was shown in section 4.7 that XYZ parametrization is not suitable to initialize features without prior knowledge it also proved that the inverse depth is linear at both low and high parallax. Therefore features in inverse depth coding can be initialized from just one observation and be directly added into the EKF. This way they are able to immediately contribute to the estimation of the camera properties, even if they are at low parallax. From equation (4.24) follows that features at low parallax (i.e. $d_i = \frac{1}{\rho_i}$ is large) may not contribute much to the estimation of the camera's position $\mathbf{r}^{WC}$, but will nevertheless provide information about the camera's orientation $\mathbf{q}^{WC}$.

In the absence of further knowledge the initial inverse depth with its confidence interval for a new initialized feature should include 0 (meaning infinite depth) even though this means that negative depth values will be included int the confidence interval. Infinite depth in terms of a camera just means that no parallax of the feature will be observed. However if the camera translates and enough parallax is produced the features depth estimation (via inverse depth) will gradually improve and the feature will start to contribute more to the estimation of the camera position.

Having stated that new features should be initialized in inverse depth encoding with a depth estimation that includes infinity a closer look at the actual initialization will be taken in the following. It is assumed that by means of a corner detector or a feature descriptor (see subsections 3.1.1 and 3.1.2) the location $(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T$ of the new feature was already detected. A new feature $\mathbf{y}_i$ will be initialized by function

$$\mathbf{y}_i = y\left(\mathbf{r}^{WC}, \mathbf{q}^{WC}, (\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i})^T, \rho_0\right) = \begin{pmatrix} x_{c,i} & y_{c,i} & z_{c,i} & \theta_i & \phi_i & \rho_i \end{pmatrix}^T \tag{4.68}$$

where $\mathbf{r}^{WC}$ denotes the position of the optical center of the camera, $\mathbf{q}^{WC}$ its orientation and ρ_0 indicates the initial inverse depth estimation.

As already stated in subsection 4.2.3 the first three coordinates of a feature in inverse depth encoding specify the camera center at its first observation so the initialization of $x_{c,i}$, $y_{c,i}$ and $z_{c,i}$ is straight forward:

$$\begin{pmatrix} x_{c,i} & y_{c,i} & z_{c,i} \end{pmatrix}^T = \mathbf{r}^{WC} \tag{4.69}$$

Acquiring azimuth θ_i and elevation ϕ_i involves a bit more mathematics. First the undistorted image coordinates $(\mathrm{u}_{\mathrm{u},i}, \mathrm{v}_{\mathrm{u},i})^T$ are calculated via $h_u\left(\mathrm{u}_{\mathrm{d},i}, \mathrm{v}_{\mathrm{d},i}\right)^T$ (see equation (2.13)). Afterwards the directional vector $\mathbf{h}^W$ is calculated, pointing from the cameras optical center towards the features location in the world coordinate frame W. $\mathbf{h}^W$ is defined as

$$\mathbf{h}^W = \begin{pmatrix} h_x^W \\ h_y^W \\ h_z^W \end{pmatrix} = \mathrm{q2r}\left(\mathbf{q}^{WC}\right) \begin{pmatrix} (u_0 - \mathrm{u}_{\mathrm{u},i}) \frac{f}{d_u} \\ (v_0 - \mathrm{v}_{\mathrm{u},i}) \frac{f}{d_v} \\ 1 \end{pmatrix} \tag{4.70}$$

where $\mathrm{q2r}\left(\mathbf{q}^{WC}\right)$ is the rotational matrix constructed from quaternion $\mathbf{q}^{WC}$ (see equation (4.23)). Though $\mathbf{h}^W$ is no unit vector, according to the pinhole camera model it still points in the direction

of the feature which will be at location $\kappa \mathbf{h}^W$ for some (unknown) $\kappa \in \mathbb{R}^+$. From the directional vector $\mathbf{h}^W$ azimuth θ_i and elevation ϕ_i can be deduced, since it holds

$$\begin{pmatrix} \theta_i \\ \phi_i \end{pmatrix} = \begin{pmatrix} \arctan\left(h_x^W, h_y^W\right) \\ \arctan\left(-h_y^W, \sqrt{\left(h_x^W\right)^2 + \left(h_z^W\right)^2}\right) \end{pmatrix} \tag{4.71}$$

That leaves the estimated inverse depth ρ_i as the sole not yet defined parameter of equation (4.68). This will be simply set to the predefined initial depth estimation, i.e. $\rho_i = \rho_0$. In [13] Davison et al. report that an initial depth of $\rho_0 = 0.1$ works well.

Thus by equation (4.68) the initial values for feature $\mathbf{y}_i$ are well defined and can be appended to the state mean μ_t. Subsequently the covariance Σ_t needs to be adapted to the new feature as well. For the moment the covariance before the addition of the new feature will be referred to as Σ_t^{old}, while Σ_t will denote the updated covariance, already incorporating information about newly added feature $\mathbf{y}_i$. If the dimensions of Σ_t^{old} were $n_{\text{dim}} \times n_{\text{dim}}$ the dimension of Σ_t will be $n_{\text{dim}} + 6 \times n_{\text{dim}} + 6$, since the associated mean μ_t of the covariance was extended by 6 dimensions. The addition of feature $\mathbf{y}_i$ to covariance Σ_t^{old} can be described by

$$\Sigma_t = J \begin{pmatrix} \Sigma_t^{\text{old}} & 0 & 0 \\ 0 & Q_I & 0 \\ 0 & 0 & \sigma_\rho^2 \end{pmatrix} J^T \tag{4.72}$$

with

$$\begin{aligned} \dim\left(\Sigma_t\right) = n_{\text{dim}} + 6 \times n_{\text{dim}} + 6,\ \dim\left(\Sigma_t^{\text{old}}\right) = n_{\text{dim}} \times n_{\text{dim}}, \\ \dim\left(J\right) = n_{\text{dim}} + 6 \times n_{\text{dim}} + 3,\ \dim\left(Q_I\right) = 2 \times 2 \end{aligned}$$

where matrix J is constructed by a $n_{\text{dim}} \times n_{\text{dim}}$ identity matrix and the partial derivatives of function y (see equation (4.68)) and will be discussed in detail in equation (4.73). Matrix Q_I is a 2×2 matrix, containing the variance of the image measurement noise and is constructed like matrix Q_t' (see equation (4.49)). The entry σ_ρ^2 is the squared standard deviation of the estimated inverse depth, so σ_ρ influences the confidence interval of the inverse depth. The authors of [13] mention repeatedly that 0 should be included in the confidence interval of the inverse depth, thus including features at infinity. Therefore they propose for $\rho_0 = 0.1$ a standard deviation of $\sigma_\rho = 0.5$, resulting in 95% confidence interval of $[-0.9; 1.1]$ for inverse depth.

Next a closer look at the structure of matrix J will be taken.

$$J = \left(\begin{array}{ccc|c} \multicolumn{3}{c|}{\mathbb{1}} & 0 \\ \hline \frac{\partial y}{\partial \mathbf{r}^{WC}} & \frac{\partial y}{\partial \mathbf{q}^{WC}} & 0 \cdots 0 & \frac{\partial y}{\partial (u_d, v_d)} \quad \frac{\partial y}{\partial \rho} \end{array}\right) \tag{4.73}$$

where $\mathbb{1}$ denotes a $n_{\text{dim}} \times n_{\text{dim}}$ identity matrix. Next the Jacobians in the second row of J will be analyzed. From equation (4.69) follows directly

$$\frac{\partial y}{\partial \mathbf{r}^{WC}} = \begin{pmatrix} \mathbb{1} \\ 0 \end{pmatrix},\ \dim\left(\mathbb{1}\right) = 3 \times 3,\ \dim\left(0\right) = 3 \times 3 \tag{4.74}$$

Unfortunately Jacobian $\frac{\partial y}{\partial \mathbf{q}^{WC}}$ is a bit more complicated and its structure is given by

$$\frac{\partial y}{\partial \mathbf{q}^{WC}} = \begin{pmatrix} 0 & 0 & 0 & \frac{\partial \theta_i}{\partial \mathbf{q}^{WC}} & \frac{\partial \phi_i}{\partial \mathbf{q}^{WC}} & 0 \end{pmatrix}^T, \; \dim\left(\frac{\partial y}{\partial \mathbf{q}^{WC}}\right) = 6 \times 4 \tag{4.75}$$

where the zero matrices correspond to the Jacobians of $x_{c,i}$, $y_{c,i}$, $z_{c,i}$ and ρ_i with respect to $\mathbf{q}^{WC}$, indicating that these components of feature $\mathbf{y}_i$ are not dependent on the camera's rotation. That leaves $\frac{\partial \theta_i}{\partial \mathbf{q}^{WC}}$ and $\frac{\partial \phi_i}{\partial \mathbf{q}^{WC}}$ to be determined. These can be stated as:

$$\frac{\partial \theta_i}{\partial \mathbf{q}^{WC}} = \frac{\partial \theta_i}{\partial \mathbf{h}^W} \frac{\partial \mathbf{h}^W}{\partial \mathbf{q}^{WC}} \tag{4.76}$$

$$\frac{\partial \phi_i}{\partial \mathbf{q}^{WC}} = \frac{\partial \phi_i}{\partial \mathbf{h}^W} \frac{\partial \mathbf{h}^W}{\partial \mathbf{q}^{WC}} \tag{4.77}$$

where $\frac{\partial \mathbf{h}^W}{\partial \mathbf{q}^{WC}}$ denotes the Jacobian of directional vector $\mathbf{h}^W$ from camera to feature in world coordinates with respect to the orientation of the camera. While $\frac{\partial \mathbf{h}^W}{\partial \mathbf{q}^{WC}}$ is derived from equation (4.70), both $\frac{\partial \theta_i}{\partial \mathbf{h}^W}$ and $\frac{\partial \phi_i}{\partial \mathbf{h}^W}$ can be deduced from equation (4.71). The resulting Jacobians are specified as

$$\frac{\partial \theta_i}{\partial \mathbf{h}^W} = \begin{pmatrix} \frac{h_z^W}{(h_x^W)^2+(h_z^W)^2} \\ 0 \\ \frac{-h_x^W}{(h_x^W)^2+(h_z^W)^2} \end{pmatrix} \tag{4.78}$$

$$\frac{\partial \phi_i}{\partial \mathbf{h}^W} = \begin{pmatrix} \left(\frac{h_x^W h_y^W}{\left((h_x^W)^2+(h_y^W)^2+(h_z^W)^2\right)\sqrt{(h_x^W)^2+(h_z^W)^2}} \right) \\ \left(-\frac{\sqrt{(h_x^W)^2+(h_z^W)^2}}{(h_x^W)^2+(h_y^W)^2+(h_z^W)^2} \right) \\ \left(\frac{h_y^W h_z^W}{\left((h_x^W)^2+(h_y^W)^2+(h_z^W)^2\right)\sqrt{(h_x^W)^2+(h_z^W)^2}} \right) \end{pmatrix} \tag{4.79}$$

$$\frac{\partial \mathbf{h}^W}{\partial \mathbf{q}^{WC}} = \begin{pmatrix} \frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_r^{WC}} \mathbf{h}^W & \frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_i^{WC}} \mathbf{h}^W & \frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_j^{WC}} \mathbf{h}^W & \frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_k^{WC}} \mathbf{h}^W \end{pmatrix} \tag{4.80}$$

The appearance of equation (4.80) is quite similar to equation (4.38) and the Jacobians $\frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_r^{WC}}$, $\frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_i^{WC}}$, $\frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_j^{WC}}$ and $\frac{\partial \mathrm{q2r}(\mathbf{q}^{WC})}{\partial q_k^{WC}}$ are in fact calculated according to equations (4.41) – (4.44).

A very interesting property compared to other EKF based visual SLAM algorithms is the fact, that the inverse depth parametrization used in this approach enables the EKF to work without any prior knowledge about the scene. Usually the initial state of an EKF includes a number of given features with known 3D position, to allow for camera state estimation and to better estimate the location of features initialized later. The VISUAL MONO-SLAM algorithm presented here works just as well without such information. It is impossible to correctly infer the positions of features fixed to a given unit scale like

meters for example without any additional knowledge, though. This is due to the ambiguity of a small movement of a single camera observing a close object and a large camera movement while observing a distant object in terms of the information gathered by the camera. However the map created by VISUAL MONO-SLAM without any additional knowledge will be consistent in itself. If for one of the features in the map the exact position in an existing coordinate system is known, a scale factor can be calculated. If all features in the map are scaled accordingly by this factor the resultant map will closely correspond to a map created by algorithms with prior knowledge.

Since the estimated positions of the features in the created map will settle to a scale of some value the algorithm proves quite robust to different values of the initial inverse depth ρ_0 (given in the same meaningless scale). A crucial point however seems to be the inclusion of infinity in the confidence interval (i.e. 0 should be contained in the confidence interval of the inverse depth).

4.9 Feature Conversion

As discussed in section 4.8 a new feature will be initialized in inverse depth coding. In principle inverse depth coding is suitable for low parallax features at large distances close to infinity as well as for close features showing high parallax. This is reflected by equation (4.24) that can cope with distant features (i.e. $\rho_i \approx 0$) and close features features. Please note that features at 0 depth ($0 = d_i = \frac{1}{\rho_i} \Leftrightarrow \rho_i = \infty$) can not be modeled. However in practice such a feature will never be included in the EKF since a depth of 0 would imply that the feature's location corresponds to the camera's optical center (which would result either in a broken camera or a feature location outside the camera's field of view). Still features in XYZ encoding have one property that makes them preferable compared to features in inverse depth. A feature in XYZ coding simply needs 3 dimensions less to be represented. While this might not seem much at first glance the run-time of the EKF is specified as $\mathcal{O}\left(k^{2.4} + n^2\right)$ (see [47]) where k denotes the dimension of the measurement vector z_t and n denotes the dimension of the current state vector. Ultimately the size of a feasible map in VISUAL MONO-SLAM in terms of features is bounded by the time available between the capture of two camera images. Of this time one share will be used mainly for image related operations like feature matching and another share will be used by the visualization of the current estimations. The remaining time has to suffice to perform all operation necessary for the several EKF steps (see algorithm 4.1). Therefore whenever it is safe to convert a feature from its inverse depth encoding to the more compact XYZ representation this should be done. The remainder of this section will be split in two parts: Firstly subsection 4.9.1 presents a simple mechanism, adapting the linearity index introduced in section 4.7 to determine if a point in inverse depth coding may safely be converted to XYZ. Afterwards subsection 4.9.2 shows how the actual conversion is accomplished.

4.9.1 Linearity Threshold

As discussed in section 4.7 the linearity index L_d (equation (4.63)) provides a measure to estimate linearity of the measurement function for a feature in XYZ encoding. Therefore a sensible method to determine if a feature may be converted from inverse depth to XYZ is to consult linearity index L_d for this feature and compare L_d with a given threshold. If L_d is smaller than the threshold the feature can be converted safely using the methods described in subsection 4.9.2, otherwise it will stay in inverse depth. To calculate L_d three variables are needed, namely the estimated depth d_1, the standard deviation σ_d of

the depth confidence interval and the cosine of parallax angle α. How these values can be obtained by a point $\mathbf{y}_i$ in inverse depth coding will be shown step by step:

First point $\mathbf{y}_i$ in inverse depth is converted to $\mathbf{x}_i$ in XYZ via equation (4.5). The ray $\mathbf{d}^W_{\mathrm{XYZ}}$ from camera to the point can be calculated according to equation (4.39). The estimated depth d_i of the feature of $\mathbf{x}_i$ is the euclidean norm of the vector from camera to point ($d_i = \|\mathbf{d}^W_{\mathrm{XYZ}}\|$). With the help of ρ_i and $\sigma_{\rho,i}$ the needed standard deviation $\sigma_{d,i}$ can be obtained. If the mean of the inverse depth ρ_i of $\mathbf{y}_i$ is stored at position n in mean vector μ_t, then the standard deviation is defined as

$$\sigma_{\rho,i} = \sqrt{\sigma^2_{t,(n,n)}}\,,$$

where $\sigma^2_{t,(n,m)}$ refers to the element at position (n, m) of covariance Σ_t. Finally $\cos\alpha$ can be calculated from directional vector $m(\theta_i, \phi_i)$ (see equation (4.6)) and $\mathbf{d}^W_{\mathrm{XYZ}}$ as $\cos\alpha = m(\theta_i, \phi_i)^T \mathbf{d}^W_{\mathrm{XYZ}} \|\mathbf{d}^W_{\mathrm{XYZ}}\|^{-1}$. Thus L_d can be computed by

$$L_d = 4\frac{\sqrt{\sigma^2_{t,(n,n)}}}{\rho_i \|\mathbf{d}^W_{\mathrm{XYZ}}\|} m(\theta_i, \phi_i)^T \frac{\mathbf{d}^W_{\mathrm{XYZ}}}{\|\mathbf{d}^W_{\mathrm{XYZ}}\|}$$

By computing linearity index L_d for each point in inverse depth coding the linearity for the corresponding XYZ point is obtained. If L_d is below a specified linearity index, then the point should be converted to reduce computational load for this point in future iterations. Point conversion is described in subsection 4.9.2. The authors of [12,13] recommend a linearity threshold L_t for conversion of $L_t \leq 0.1$. This value was experimentally determined by a simulation for using different values for α, d_i and σ_d. The details of the simulation are omitted here. For details please refer to [12, 13].

4.9.2 Conversion Mechanism

In order to switch a point from inverse depth encoding to XYZ representation the current mean μ_t and covariance Σ_t of the EKF have to be modified. The former is fairly simple and can be done by using equation (4.5), but the latter requires the computation of the Jacobian of (4.5), namely $\frac{\partial \mathbf{x}_i}{\partial \mathbf{y}_i}$. This can be obtained from equations (4.5) and (4.6) as

$$\frac{\partial \mathbf{x}_i}{\partial \mathbf{y}_i} = \begin{pmatrix} 1 & 0 & 0 & \frac{1}{\rho_i}\cos(\theta_i)\cos(\phi_i) & -\frac{1}{\rho_i}\sin(\theta_i)\sin(\phi_i) & -\frac{1}{\rho_i^2}\sin(\theta_i)\cos(\phi_i) \\ 0 & 1 & 0 & 0 & -\frac{1}{\rho_i}\cos(\phi_i) & \frac{1}{\rho_i^2}\sin(\phi_i) \\ 0 & 0 & 1 & -\frac{1}{\rho_i}\sin(\theta_i)\cos(\phi_i) & -\frac{1}{\rho_i}\cos(\theta_i)\sin(\phi_i) & -\frac{1}{\rho_i^2}\cos(\theta_i)\cos(\phi_i) \end{pmatrix} \tag{4.81}$$

If the mean before the point conversion is denoted as $\mu_t^{\mathrm{old}} = (\mathbf{x}_v, \mathbf{f}_1, \ldots, \mathbf{y}_i, \ldots, \mathbf{f}_n)^T$ with dimension $\dim\left(\mu_t^{\mathrm{old}}\right) = n_{\dim}$, mean μ_t after the conversion will be $\mu_t = (\mathbf{x}_v, \mathbf{f}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{f}_n)^T$ and its dimension $\dim\left(\mu_t^{\mathrm{old}}\right) = n_{\dim} - 3$. Consequently covariance Σ_t^{old} has to be modified to create the new covariance Σ_t incorporating the converted point. If $a_{\dim} = \sum_{j=0}^{i-1} \dim(\mathbf{f}_{t,i}) + 13$ denotes the dimensionality of the camera and all points before point i and $b_{\dim} = \sum_{j=i+1}^{n} \dim(\mathbf{f}_{t,i})$ gives the dimensionality of all points behind point i in the state vector, then covariance Σ_t can be expressed as:

$$\Sigma_t = J\,\Sigma_t^{\mathrm{old}} J^T \tag{4.82}$$

with

$$J = \begin{pmatrix} \mathbb{1}_a & 0 & 0 \\ 0 & \frac{\partial \mathbf{x}_i}{\partial \mathbf{y}_i} & 0 \\ 0 & 0 & \mathbb{1}_b \end{pmatrix}, \ \dim(J) = n_{\dim} - 3 \times n_{\dim} \tag{4.83}$$

where $\mathbb{1}_a$ and $\mathbb{1}_b$ denote identity matrices of dimension $a_{\dim} \times a_{\dim}$ and $b_{\dim} \times b_{\dim}$ respectively and 0 indicate zero matrices of appropriate dimensions.

4.10 Point Deletion

While not playing a part in theoretical descriptions of EKF and its applications for SLAM, the deletion of features plays an important part in practice. Without any deletion mechanism state vector $\mathbf{x}_t$ would be ever-growing which slows down the performance of the EKF. Reliable features which are matched repeatedly should of course not be deleted, but there may be features that will repeatedly not be matched after their initialization. Such features will not contribute to the state estimation in any way, but result in additional computational effort. If such features are removed from the state vector, new promising features may be added without endangering run-time constraints imposed by high frame rate of image retrieval.

A sensible mechanism to detect if a feature should be deleted is to measure the ratio of successful matches to the number of match attempts. If this ratio is below a given threshold (for example 50%), the feature will be deleted. Though this mechanism is quite simple it preserves stable features outside of the current field of view. To add to robustness the matching ratio should only be considered, after an initial number of matches was attempted. Of course this basic idea can be extended in various manners. For example if a certain number of successful matches are recorded for a feature its match ratio may be reduced (since the high number of matches implies that for certain camera positions the feature is quite stable and thus improves the overall estimation).

The deletion from the EKF itself is much more simple than the addition of a new feature. If point $\mathbf{f}_{t,i}$ is to be deleted it just needs to be removed from the current mean and the covariance. If the index of the first entry of feature $\mathbf{f}_{t,i}$ is j and its last entry is $j + \dim(\mathbf{f}_{t,i})$ these dimensions will just be removed from μ_t and columns and rows $j - j + \dim(\mathbf{f}_{t,i})$ will be removed from covariance Σ_t.

Although not explicitly designed to do so the deletion mechanism may introduce some robustness to a non static environment. If some distinct features are detected on a moving object they will be added to the EKF like any other feature. However if the estimation of the camera movement is stable enough a feature on a moving object will repeatedly not be matched in its predicted search region and thus be deleted again after a short time.

Chapter 5

Evaluation

While chapter 4 and the previous chapters covered the theoretical background for VISUAL MONO-SLAM, this chapter is devoted to a practical evaluation of the discussed methods. First a simulation of VISUAL MONO-SLAM is briefly introduced in section 5.1 along with the created GUI to visualize the results. Afterwards section 5.2 shows the experiments conducted with real images as input.

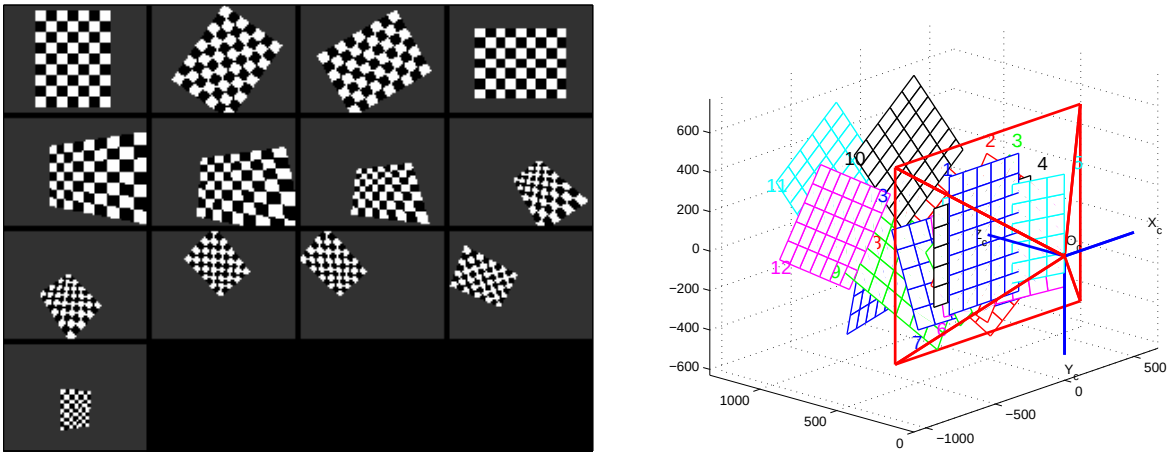
All code produced for the evaluation was written in `C/C++`. To keep things simple, visualization was done in OPENGL, mainly using GLUT and GLUI ([44] provides a good starting point for more information concerning OPENGL and GLUT). Grabbing images from cameras and most of the image processing tasks were done by or based on OPENCV.

5.1 Simulation

Like for many other applications a simulation of VISUAL MONO-SLAM can become a valuable evaluation tool, since it provides a closed environment without any unknown parameters. From the rather lengthy description of the principle description of VISUAL MONO-SLAM in chapter 4 the reader might have already guessed that the actual implementation of VISUAL MONO-SLAM also requires a lengthy amount of source code. Usually the larger a program becomes the larger the possibility for bugs and unforeseen side effects becomes. Considering that image processing often has to handle noise (see chapter 3) and the finding the correct thresholds often requires both time and fine tuning it may be hard too determine if and in which part of the source code an error might be located. In a simulation environment it becomes possible to disregard image processing and various noise induced by image processing to evaluate the basic VISUAL MONO-SLAM algorithm.

5.1.1 Simulation Setup

To avoid processing of real camera images and leave as much source code identical for both simulation and real application a virtual camera was constructed. Since VISUAL MONO-SLAM was implemented using `C/C++`, visualization and virtual camera were implemented in OPENGL. The virtual camera basically consists of a vector $\mathbf{x}_v$ to define its position, orientation and velocities (see equation (4.2)) which enables the simulation to compare the actual camera state with the camera state estimated by the EKF. Furthermore the virtual camera depicts the 2 dimensional projection of a scene rendered in OPENGL that



(a) Calibration sequence

(b) Estimated extrinsics

Figure 5.1: Virtual camera calibration. **(a)** depicts a short calibration sequence for a virtual camera. Note that the calibration sequence used for a virtual camera might be quite short, since distortion coefficients k_1 and k_2 will be 0. **(b)** shows the estimated extrinsics for the calibration sequence depicted in (a). Estimated extrinsics were obtained by the MATLAB toolbox [6].

corresponds to the image perceived by a camera defined according to vector $\mathbf{x}_v$. Surprisingly at it may seem at first, a virtual camera also has to be calibrated. Remember that the main goal of the simulation is to evaluate the methods used in the real world application in a determined environment. Therefore the measurement function discussed in section 4.4 used to predict expected measurements should not be modified. Of course a virtual camera defined in OPENGL will not exhibit lens imperfections like a real camera. Therefore the distortion coefficients k_1 and k_2 (see equations (2.13) and (2.17) may safely be assumed to be 0 and the image center (u_0, v_0) is located at $\left(\frac{width}{2}, \frac{height}{2}\right)$. However a virtual camera will still have one property that cannot be deduced easily or directly corresponds to the parameters used in OPENGL to define the projection properties. To predict the measurement for a given point in the state vector the focal length f is needed (see equation (4.25)). Similar to a real camera the virtual camera can be calibrated by observation of a chessboard, with the only difference that the chessboard is now a virtual chessboard rendered in OPENGL and projected from different viewing positions. An example for the calibration of a virtual camera is depicted in Figure 5.1.

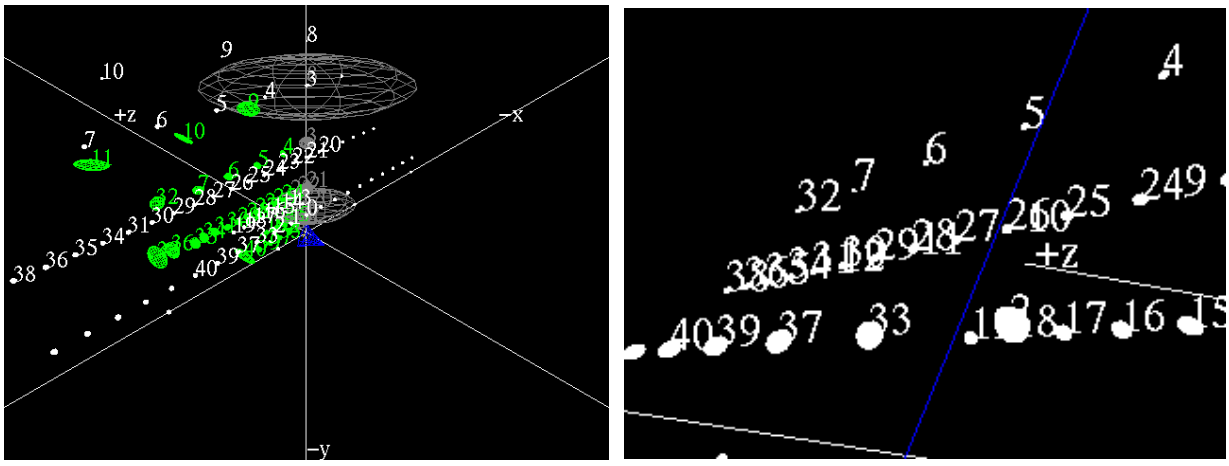
In the simulation the virtual camera will observe a scene containing virtual landmarks. Virtual landmarks are specified by their 3D coordinates in the world coordinate frame. In the visualization they are shown as solid white sphere. As perceived coordinates of a virtual landmark the projection of its 3D coordinates for the scene depicted by the virtual camera is used. These coordinates can be easily obtained by OPENGL. If the received coordinates are inside the search region of the associated feature, the feature resembling the virtual landmark in VISUAL MONO-SLAM is matched. Otherwise the feature is not successfully matched. Thus the need for a matching mechanism needed in the real application can be avoided in the simulation. Otherwise the simulation works just like the real application, newly observed landmarks may be initialized as new features, features can be deleted and if the linearity index for a feature in inverse depth is below the given threshold, the feature will be converted. The estimated

map and virtual camera perspective for the simulation are depicted in Figure 5.2. The estimations for each landmark are shown by wired ellipsoids. The color of the ellipsoids indicates the current state of the corresponding feature: Green ellipsoids were predicted to be inside the current image and matched successfully, red ellipsoids were not successfully matched and gray indicates that the feature was predicted to be outside of the current image. Visualization for points in XYZ encoding is straight forward and their uncertainty can be easily calculated by the variances found on the main diagonal of the covariance matrix. The indicate association between a landmark and its corresponding feature, both are labeled with the same number. To visualize features in inverse depth coding these need to be converted with their corresponding covariance to XYZ representation as discussed in section 4.9. Due to the linearity issues with features in XYZ showing low parallax (see section 4.7) the visualized uncertainty estimation for features in inverse depth might sometimes show odd behaviour with vastly changing uncertainty ellipsoids. However this is just a visualization issue, since inverse depth features that should not be converted to XYZ according to their linearity index, need to be converted in order to visualize them. The estimated camera position and orientation is marked by a blue cone, where the pinhole is located at the center of its flat side (i.e. the top of the cone indicates the opposite of the viewing direction). The estimated path of the camera is drawn in yellow.

The simulation can be influenced via a graphical user interface. The interface allows for manipulation of the virtual camera by determining its linear and angular velocities. This can be done either for a predetermined number of frames or repeatedly until another command is issued. A smooth return to the origin in a given number of steps is implemented employing SLERPS introduced in [43] to determine the proper quaternion rotations. Furthermore several viewing options enable the user to switch on and of the display of uncertainty ellipsoids for XYZ or inverse depth features and the like. The view on the estimated map can be influenced either via sliders in the GUI or by a first-person shooter like navigation. Since VISUAL MONO-SLAM estimates the positions of all features and the camera pose without any prior knowledge usually the estimates differ from the virtual landmarks and camera, but a consistent at a meaningless scale (see section 4.8). To better compare the overall consistency of the estimated map with the given virtual landmarks a scale vector for all features can be calculated easily, since the 3D positions of the virtual landmarks are exactly known. To estimate the scale vector 5 features position estimations are randomly determined and compared with the positions of the corresponding virtual landmarks. The resulting scale vector is then used to scale the visualization of all feature estimations accordingly.

5.1.2 Simulation Results

The simulation yields generally good results for the estimation of both, inverse depth and XYZ coded features. Also the error in the estimated positions compared to the known positions of the estimated landmarks is reasonable for features that have been observed for more than just a few frames. In Table 5.1 the scaled estimates for some selected features are presented. To obtain the values shown in Table 5.1 the virtual camera moved over a total of 210 frames, which is a rather short sequence. The exact velocities and number of frames defining the movement of the virtual camera are shown in Table 5.2. Still for some features enough parallax was detected so that they could safely be converted to XYZ coding. To give an overall view the displayed features were selected according to three criteria: Landmark depth, position in the initial image and how often the feature was observed during the 210 frames sequence. Note that due to the random element in the scale vector not necessarily the best possible estimation compared with the original value might be obtained (for example if a newly initialized feature is selected to contribute



(a) Estimated map **(b)** Virtual camera view

Figure 5.2: Simulation view: **(a)** shows the estimated map. Landmarks are depicted as white spheres, the 95% confidence region for the estimation is illustrated by wired ellipses. The estimated camera position is marked by a blue cone, where the pinhole is in the center of the flat side. In **(b)** the view from the virtual camera is depicted. Note that this view is not exactly the view from the blue cone in (a), since that is the estimation of the camera. However the camera estimation should not differ significantly from the state of the virtual camera. No ellipsoids are displayed in (b), since the estimated confidence regions do not belong to the observed scene.

to the scale vector). To further evaluate the experiment in the simulation the trajectory of the virtual camera and the EKF estimation of the trajectory have been logged. Figure 5.3 shows a comparison of the trajectories projected in the XZ-plane (Figure 5.3a) and XY-plane (Figure 5.3b), respectively. The estimation error in the trajectory peaks at the end of the first camera movement sequence at frame 100 (see Table 5.2). This can be explained by the camera movement: Before frame 100 the camera observes new features and adds them to the current state vector, while some of the initial features begin to drift out of the field of view. Afterwards the changing velocities move the camera in such a way that various features are re-observed. Re-observations of this kind improve pose estimation (similar like real loop closing) and therefore the pose error gradually becomes smaller.

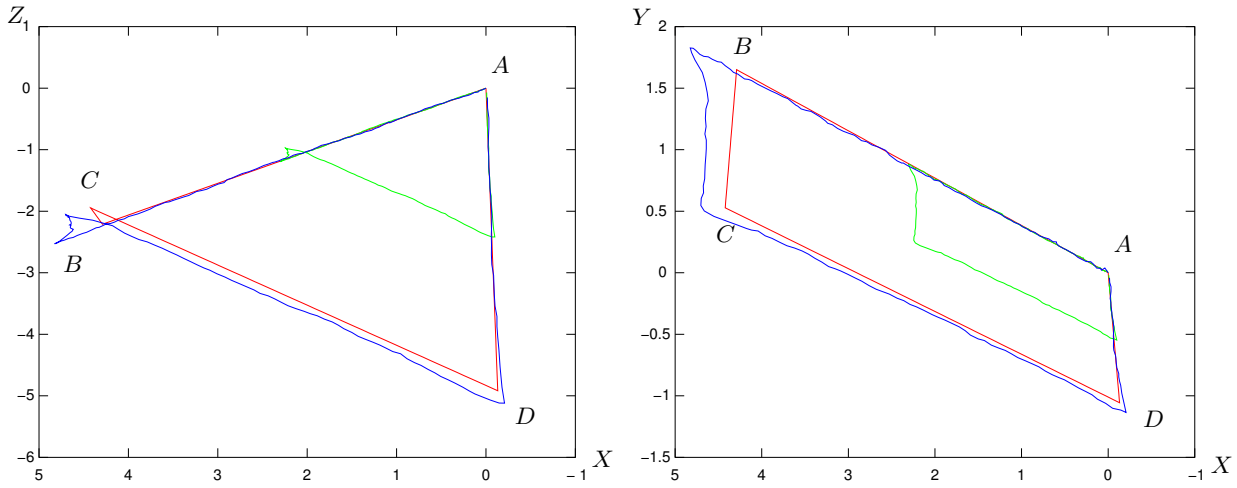
During experimenting with the simulation it became apparent the rotations of the virtual camera, induced by angular velocities play an important part in VISUAL MONO-SLAM. Position estimates obtained if the virtual camera was just subject to linear velocities were less accurate than movements featuring both types of velocities. An interesting effect can be observed in the simulation for small linear velocities in the absence of angular velocities: In this case the predicted movements of the virtual camera differ in the sign of the actual movements, i.e. if the camera is moving along the positive X-axis the EKF estimates a movement in direction of the negative X-axis. Subsequently the features are estimated behind the camera, since these positions would correspond to the estimated camera movements. Investigation of this phenomena has not come up with a satisfactory explanation until now. For a hand-held camera this effect may not prove important, since hand-held devices will always feature small rotations.

Landmark			frame 100		frame 120		frame 180		frame 210	
description	position		mean	3D	mean	3D	mean	3D	mean	3D
ID: 1, low	x	0	-0.03		-0.04		-0.03		-0.03	
depth, centered,	y	0	0.01	✘	0.01	✔	0.01	✔	0.01	✔
always observed	z	5	5.07		5.21		4.92		4.91	
ID: 10, high	x	0	-0.23		-0.26		-0.33		-0.34	
depth, centered,	y	15	14.32	✘	14.73	✘	14.82	✘	14.85	✘
always observed	z	100	92.17		95.28		96.87		97.20	
ID: 20, medium	x	-25	-13.43		-14.24		-24.81		-24.97	
depth, lower right,	y	5	2.71	✘	2.83	✘	4.99	✘	5.01	✘
sometimes observed	z	20	10.67		11.22		19.92		20.03	
ID: 7, medium	x	30	29.41		30.65		29.20		29.42	
depth, upper left,	y	20	19.70	✘	20.29	✘	19.53	✘	19.67	✘
often observed	z	30	29.35		30.35		29.46		29.67	
ID: 37, low	x	15	15.27		15.53		14.51		14.54	
depth, not in initial view	y	0	-0.02	✘	0.00	✘	0.00	✘	0.01	✔
seldom observed	z	5	5.14		5.12		4.85		4.85	
ID: 43, medium	x	-45	–		–		-52.06		-48.99	
depth, not in initial view,	y	20	–	–	–	–	23.13	✘	21.70	✘
seldom observed	z	30	–		–		35.36		32.99	

Table 5.1: Position estimates from simulation. The table shows the estimated positions for some selected features during the simulation. The columns labeled 3D indicate if the feature was already converted to XYZ encoding. The scene with the virtual landmarks is depicted in Figure 5.2a. For a specification of the time steps please refer to Table 5.2. If no estimation is given then the feature was not observed until the corresponding time step.

frames	v_x^W	v_y^W	v_z^W	ω_x^C	ω_y^C	ω_z^C
0 – 99	1.300	0.500	-0.670	-0.050	0.120	-0.060
100 – 119	0.200	-1.700	0.400	0.100	-0.250	0.180
120 – 179	-2.300	-0.800	-1.500	-0.200	-0.130	-0.150
180 – 209	0.133	1.067	4.967	0.533	0.026	0.318

Table 5.2: Virtual camera velocities defining the movement for the estimates presented in Table 5.1. The velocities from frame 180–209 describe a return to the origin.



(a) XZ-plane trajectory

(b) XY-plane trajectory

Figure 5.3: Camera trajectories in simulation. **(a)** shows a trajectory in the XZ-plane, **(b)** in the XY-plane. The velocities for the camera of the trajectories are given in Table 5.2. The real trajectory of the virtual camera is shown as a red line, while the green line shows the estimates of dimensionless VISUAL MONO-SLAM. Multiplying the estimated trajectory with a scale vector results in the blue trajectory. Letters A, B, C and D mark the changes in camera velocities, according to Table 5.2. A marks the position at frame 0 (and frame 210), B the position at frame 100, C at 120 and D at 180.

5.2 Real Data Experiments

Evaluating experiments with real data proved to be much harder than experimenting in the simulation. In subsection 5.2.1 the qualitative results using two different image sequences created by the authors of [13,16,20] have been used as input for VISUAL MONO-SLAM. Afterwards in subsection 5.2.2 results for real-time VISUAL MONO-SLAM using a HERCULES WebCam Classic are discussed. For real-time estimation feature comparison is done by image patch matching, were patches of size 11×11 and 21×21 were tested. As a comparison measure for image patches the normalized cross correlation (see equation (3.16)) was used. The expected amount of predicted features was set to 10 - that means that new features will be initialized if less than 10 features are predicted to be in the current image. The amount of features was determined experimentally - less features led to qualitatively worse pose and position estimates, more features did not significantly improve the estimation but only increase computational load.

For complete image sequences SURF-features were tested as an alternative to image patches. For real-time estimation SURF could not be tested, since the neither the OPENCV implementation of SURF-features nor the original implementation performs fast enough for feature extraction at 15 – 30 Hz.

5.2.1 Given Image Sequences

Two image sequences created by the authors of [13, 16, 20] have been used to test the working of the VISUAL MONO-SLAM implementation. Both feature images of size 320×240 pixels and were obtained by a 30 fps fire-wire camera with known intrinsic camera parameters. One image sequence depicts

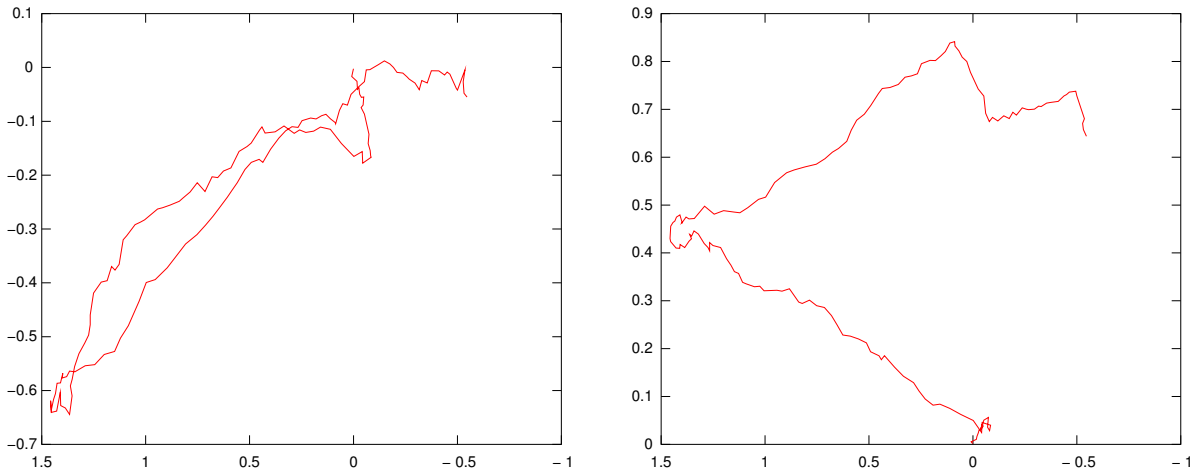
(a) Camera image **(b)** Detail from scene

Figure 5.4: Detected features in outdoor scene. In **(a)** the last frame of the outdoor scene is shown. Detected features in the current frame are marked by a black number. The green squares indicate the area where the image patch was matched. Blue ellipses indicate the 95% confidence region. However the confidence regions are hardly visible in the depicted scene, since they are smaller and hidden by the matched regions (which indicates good predictions). Subfigure **(b)** provides an image detail of (a) and contains all features referred to in Table 5.3. For better readability feature IDs are colored red.

an outdoor scene with varying lighting conditions showing a street with some cars in the foreground and some distant features near the horizon. The sequence is quite short, just providing 180 frames. The second image sequence is longer, comprised of 1000 frames. It shows an indoor lab environment, containing mostly features showing high parallax that can quickly be converted to XYZ from their initial inverse depth encoding. Since for the scenes shown in the image sequences no known scale is provided the evaluation can only be a qualitative, not quantitative. It should be noted that Davison et al. do not publish an quantitative analysis of the estimated 3D positions for real image data in [13, 16, 20] either.

Using image patches for feature matching in both image sequences the visualized estimated state of the camera matches well with the movements done in the image sequence. Also generally the depth estimation seems to be consistent in the scenes. For example in the outdoor sequence features located on a car in the foreground differ in their z coordinate significantly from features on objects in the middle or background as shown in Figure 5.4 and Table 5.3. Furthermore for rotations and translations of the camera seen in the image sequence, similar rotations and translations could be observed for the estimated camera. A plot of the estimated camera trajectory during the sequence is plotted in Figure 5.5.

To illustrate the progress of the location estimate the estimates for selected frames are depicted in Figure 5.6. Note that in the initial frame (Figure 5.6a) the uncertainty ellipsoids are very large and overlapping, due to large uncertainty about the feature depth. However for reliably matched features the uncertainty quickly converges (Figure 5.6b). It should be noted that during the outdoor sequence no feature is converted from inverse depth to XYZ representation. This is due to the fact that in this short image sequence no feature is observed over the whole sequence and the number of successful matches was not sufficient to push the linearity index below the threshold of 0.1 (for the definition of linearity index, its analysis and meaning please refer to sections 4.7 and 4.9).



(a) trajectory in XZ-plane

(b) trajectory in XY-plane

Figure 5.5: Camera trajectory for outdoor sequence. During the sequence the camera is first moved a bit to the right and afterwards to the left and in the direction of the negative Z-axis. During this movement the viewing direction is rotated slightly around the Y-axis. Afterwards the camera is moved roughly back to its initial position. **(a)** shows the estimated plot in the XZ-plane which qualitatively fits the observed camera trajectory well. In **(b)** the trajectory in the XY-plane is plotted. While during the sequence first an upward movement of the camera could be noticed in the end it is moved downward again. This is also reflected in the plotted trajectory. A comparison with the ground truth as in the simulated case (see Figure 5.3) is not possible.

Similar results could be obtained for the indoor sequence. This sequence, being much longer than the outdoor sequence and being in an indoor environment with no faraway features provided much parallax for every stable feature. In fact at the end of the indoor sequence every feature in the estimated map is encoded in XYZ. Two exemplary frames from the indoor sequence are depicted in Figure 5.7. Using image patches of 11×11 or 21×21 pixels did not have any significant influence on camera pose estimation or the estimated positions of feature locations.

Using SURF features instead of image patches showed less promising results. Though SURF features are generally considered stable and have successfully been used for object recognition task single SURF features proved not to be as locally stable as image patches. For object recognition tasks it might not be crucial if a matched SURF feature might be detected one or two pixels off from its actual location, but in an application like VISUAL MONO-SLAM such an error may afflict the current pose estimation and subsequently the predictions and estimations for the next frame. Sometimes features can be observed to slightly move around in the image, which is rather fatal for the state estimation. Examples for faulty feature matching for SURF features are depicted in Figure 5.8.

5.2.2 Real Time Estimation

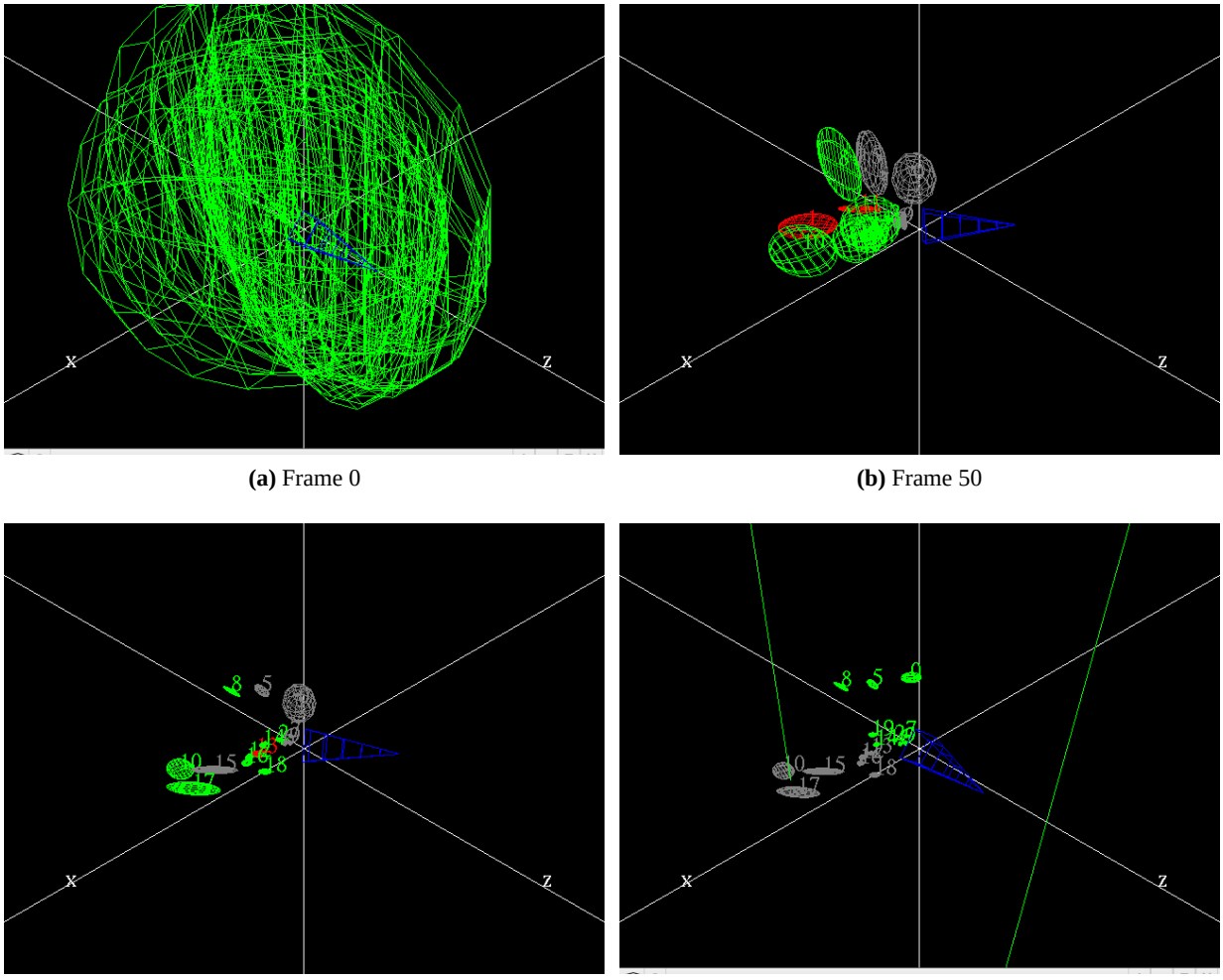
Results obtained using the given image sequences with image patch matching were encouraging with respect to the obtained maps, so that the same method was tried in real-time. For image retrieval a HERCULES WebCam Classic was used, where the native resolution was downscaled to 320×240 pixels

Position in scene		Feature ID	x	y	z
Car	rearview mirror	2	0.5652818	-1.136854	7.700869
	rim	6	-0.8424688	-2.723368	7.394391
	headlight	7	-2.616124	-1.217564	7.140176
Background	right	0	-19.60458	3.135008	22.95719
	center	5	-9.627525	2.788888	27.84539
	left	8	-1.600957	3.779855	31.05553

Table 5.3: Position estimates for selected features of the outdoor sequence. The feature IDs refer to the number depicted in Figure 5.4b. Of special interest is the last column showing the estimate of the z coordinate. From a qualitative standpoint the depth estimates obtains the expected estimations: The z coordinates of the features located on the car in the foreground are similar and differ significantly from the estimates of the features located in the background. Judging from Figure 5.4b feature 8 of the background features #8 is the farthest, while feature #0 is the closest. This is resembled in the estimates of the z coordinates.

and camera calibration parameters were determined by the MATLAB toolbox (see [6]). However real-time results with the HERCULES WebCam Classic do not achieve comparable quality as with the given image sequences. One main difference between the camera used by Davison et al. in [13, 16, 20] and the low-cost HERCULES WebCam Classic is the actual number of frames provided by the cameras. While Davison et al. report a stable frame rate of 30 fps which means a new image is obtained every 33 ms the time to retrieve a new image with the HERCULES WebCam Classic fluctuates. While the manufacturer claims frame rates of "up to 30 fps" the actually obtained frame rate is roughly 15 fps. Unfortunately that does not imply that a new image will be obtained ever 66 ms, as one could guess at first, but retrieval times between 45 ms and 118 ms have been recorded. Keep in mind that the time between two images influences the prediction of the next camera pose (see section 4.3) and will induce larger uncertainty about the features expected position. Furthermore the perceived features proved less stable than the features obtained in the given image sequences. Sometimes "moving features" similarly to the effect observed for SURF features on the image sequences could be observed, with equally bad effect on the estimated camera pose. And once the error in the pose estimation becomes too large, even stable features will not be matched anymore, since they will not be at their expected positions in the image. Increasing the threshold for the response (see 3.1.1) inhibits this effect, but led to less accepted features in the first place. This reduced amount of information gained by the features worsened the pose estimation, so that the ultimate effect is the same as for "moving features". At the moment it is not clear, why these effects could not be observed in the given image sequences, but a first guess would be that the image quality obtained by the HERCULES WebCam Classic is worse than the images in the sequences. Using the native resolution of 640×480 pixels seemed to lead to more image noise, resulting in an even faster lost pose estimation.

Therefore at the moment no meaningful results can be presented for the real-time estimation, but it should be noted that used thresholds and parameters coming into play at various parts of the algorithm influence the outcome greatly. Fine-tuning of these values proves to be very time consuming and sometimes more art than science.



(a) Frame 0

(b) Frame 50

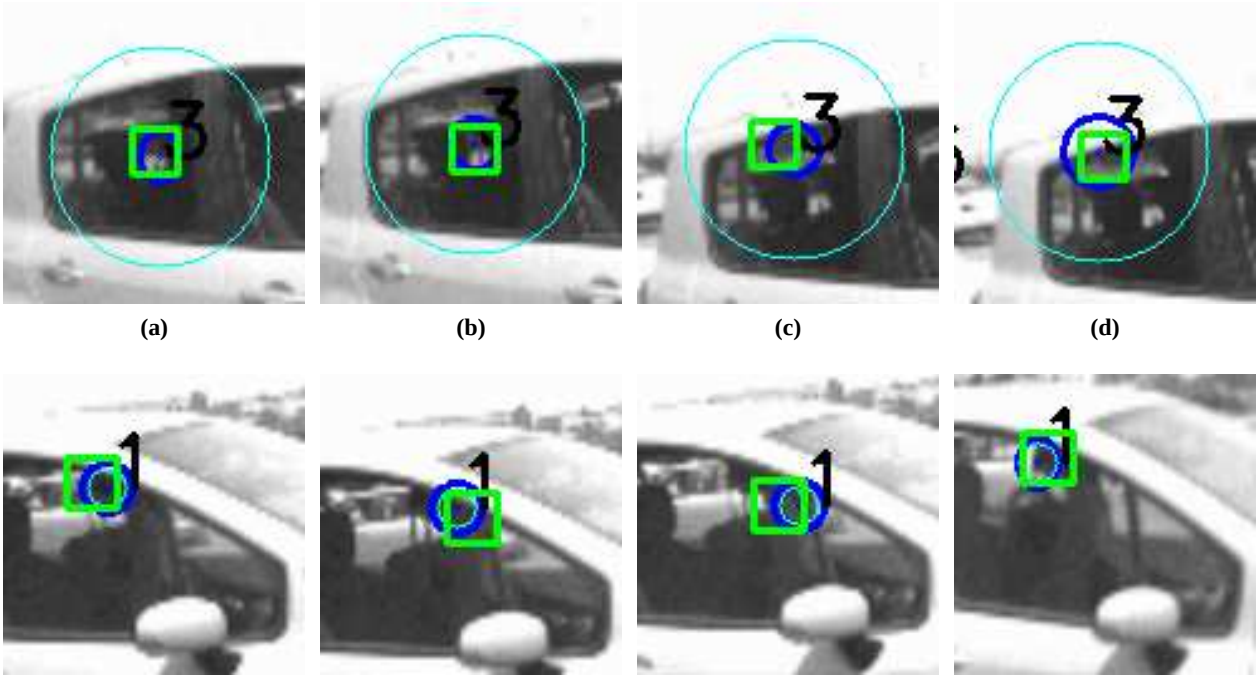
(c) Frame 100

(d) Frame 150

Figure 5.6: Progress of map estimation for outdoor sequence. The ellipsoids illustrate feature uncertainty, estimated camera pose is depicted by a blue cone. Green ellipsoids indicate a successfully matched feature, red means no successful match and gray ellipsoids are not predicted to be on the image sensor. While the estimated map in **(a)** does not convey much information, due to large uncertainty for each feature, estimation quickly improves as seen in **(b)**. Re-observation of features can significantly improve their estimated positions as seen for feature #0 (the feature close to the positive Y-axis) in the different estimation in **(c)** and **(d)**.

(a)

Figure 5.7: Exemplary frames from the indoor sequence. Red indicates unsuccessful matching attempts, while green indicates successful matches.



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

(e) **(f)** **(g)** **(h)**

Figure 5.8: False matches using SURF. The top row (**(a)** – **(d)**) depicts so called SURF features extracted by OPENCV. The OPENCV implementation was inspired by the original surf paper [4] but is not equivalent and performs oftentimes worse. In the second row (**(e)** – **(h)**) results from the original SURF implementation are shown. All matches were considered successful and the feature location is in the center of the green square. In both cases significant movement of the depicted feature position can be observed. Therefore in the context of VISUAL MONO-SLAM these features are not suitable.

Chapter 6

Discussion and Outlook

This thesis provides a detailed analysis of VISUAL MONO-SLAM along with the underlying methods from image processing in chapter 3 and camera models in chapter 2. The basic idea of the extended Kalman filter was briefly introduced in section 4.1 and elaborated by the exemplary application VISUAL MONO-SLAM in the remainder of chapter 4. The achieved results using an OPENCV and OPENGL based implementation were presented in chapter 5.

While qualitatively encouraging results could be obtained for two given image sequences, the current implementation did not yield useful results applied on a stream obtained by a low-cost USB camera. Further parameter tuning and testing will be needed to enable stable pose estimation in the real-time scenario which is crucial to a properly working VISUAL MONO-SLAM implementation.

To improve results for the given image sequences and a working real-time application the image patch matching should be addressed further. In the current implementation obtained image patches are just compared to the patches from other images inside the 95% confidence region of its associated features. While this yields reasonably good results in many cases, it still lacks some desirable properties: Matching will not be rotational invariant nor will image patches be matched if the difference between the current point of view and the point of view of the first observation becomes to large. Two strategies could be employed to address this problem: Firstly one could use a *variable image patch* instead of a constant one. This would mean that for every successful match of an image patch, the patch stored along with the feature will be replaced by the matched patch from the current image. Since orientation and point of view of an image patch will usually change continuously and not abruptly over a sequence of received frames this could provide rotational invariance along with robustness to changes in point of view and scale. However if the camera performs a loop motion then the variable image patch will less likely be matched at the loop closing, than a constant image patch. Secondly image patches could be transformed according to the currently estimated camera pose. This would require for each image patch to have a unique orientation which could possibly be obtained by directional vector $m(\theta_i, \phi_i)$, dependent on azimuth θ_i and elevation ϕ_i at feature initialization. With this orientation for an individual feature, its estimated position and the estimated camera pose one could try to transform the image patch to best resemble expected patch under the current point of view. However some thresholds will be needed for this approach since the transformation will likely result in a blurred patch, so that for small changes in the point of view this technique may actually provide worse results than the simple matching approach. Furthermore if the estimation of the feature position has a large uncertainty as for features just after

initialization, such a transformation will reduce the probability of successful matching. Eventually a combination of both, variable image patches and image patch transformation, may be beneficial. On the other hand other methods apart from image patches should be tested as well. A recently developed feature descriptor like CENSURE [2] might be interesting, since its authors claim real-time capabilities and robust matching.

Also the number of features matched for one frame could be addressed. If more features than the desired number for stable estimation are predicted to be visible in the current frame, one could limit the amount of features that will actually be searched in the image, thus reducing computational effort. The uncertainty of the expected features could be used as a heuristic to select the features: If the uncertainty of a feature is large observation of this feature will usually have more impact on the overall estimation than the observation of a feature whose position is already well known. Thus features with a high uncertainty should be preferred compared to low uncertainty features.

Extending the simulation could also provide some additional insights. Up until now image noise has not been regarded in the simulation - for every observable feature a perfect match can be obtained. Introducing the user with the ability to experiment with different image noises or the noise of the transition function might be interesting. In addition a different method to influence the movement of the virtual camera apart from the values for the 6 velocities is needed. To simulate shaky movements of a hand-held camera a noise function affecting the velocities should be implemented, as well as a simpler method of input to direct the camera instead of tediously inserting the desired velocities would be beneficial.

Different transition functions in simulation as well as in the real application could be used to model scenarios different to the hand-held camera. For example if a camera is mounted on top of a robot, actions influencing the robots pose like steering commands should be incorporated in the state transition to improve a priori pose estimation. In this case one could also think of sensor fusion with other sensors like laser based range finders or time-of-flight cameras. While range finders provide better depth measurements, thus yielding good information about translational movements, rotations are reliably estimated by VISUAL MONO-SLAM. Even if the detected rotation is not accurate it might serve as a first guess used in scan matching approaches, since these generally yield better results the closer the initial pose estimation is to the actual pose.

If fully operational for real-time operation, VISUAL MONO-SLAM could provide a powerful tool as a stand alone application for hand-held cameras and mounted on a mobile robot platform for 6DSLAM with sparse 3D maps. Furthermore in the domain of mobile robotics the estimates obtained by VISUAL MONO-SLAM could be used in a sensor fusion approach together with other sensors to improve overall map quality. This could either be done by combining odometry and or gyro compass measurements to influence the state transition of VISUAL MONO-SLAM or by employing the camera pose estimation of VISUAL MONO-SLAM as an initial pose estimation for scan matching approaches.

As a final remark it should be mentioned that creating a fully operational version of VISUAL MONO-SLAM, yielding results comparable to those of Davison et al., requires a lot of work in two respects. Firstly the software infrastructure needs to be created, before any meaningful experiments may be conducted. In this respect the overall robustness of the EKF may sometimes become actually a disadvantage, since it may cover eventual implementation bugs. Secondly the determination of various parameters and thresholds like image patch size or the required number of features needs time and patience.

Bibliography

[1] Adelson, E.H., Anderson, C.H., Bergen, J.R., Burt, P.J. and Ogden, J.M. 1984, Pyramid methods in image processing. *RCA Engineer*, 29(6):pages 33–41, 1984.

[2] Agrawal, M., Konolige, K. and Blas, M.R. CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching. In D.A. Forsyth, P.H.S. Torr and A. Zisserman, editors, *ECCV (4)*, volume 5305 of *Lecture Notes in Computer Science*, pages 102–115. Springer, October 2008. ISBN 978-3-540-88692-1.

[3] Baker, M. EuclideanSpace - building a 3D world. `http://www.euclideanspace.com/`, August 2009 (access date).

[4] Bay, H., Tuytelaars, T. and Van Gool, L. SURF: Speeded Up Robust Features. In *9th European Conference on Computer Vision*. Graz Austria, May 2006.

[5] Borrmann, D., Elseberg, J., Lingemann, K., Nüchter, A. and Hertzberg, J. The Efficient Extension of Globally Consistent Scan Matching to 6 DoF. In *Proceedings of the 4th International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT '08)*, pages 29–36. Atlanta, GA, USA, June 2008.

[6] Bouguet, J.Y. Camera Calibration Toolbox for Matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`, June 2009 (access date).

[7] Bradski, G. and Kaehler, A. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, October 2008. ISBN 9780596516130.

[8] Brown, D.C. Close-range camera calibration. *Photogrammetric Engineering*, 37(8):pages 855–866, 1971.

[9] Brown, M. and Lowe, D. Invariant Features from Interest Point Groups. In *In British Machine Vision Conference*, pages 656–665. 2002.

[10] Canny, J.F. Finding Edges and Lines in Images. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[11] Civera, J., Davison, A.J. and Montiel, J.M.M. Dimensionless Monocular SLAM. In *IbPRIA '07: Proceedings of the 3rd Iberian conference on Pattern Recognition and Image Analysis, Part II*, pages 412–419. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72848-1.

[12] Civera, J., Davison, A.J. and Montiel, J.M.M. Inverse Depth to Depth Conversion for Monocular SLAM. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation*. April 2007.

[13] Civera, J., Davison, A.J. and Montiel, J.M.M. Inverse Depth Parametrization for Monocular SLAM. *IEEE Transactions on Robotics*, 24(5):pages 932–945, October 2008.

[14] Clarke, T.A. and Fryer, J.G. The Development of Camera Calibration Methods and Models. *The Photogrammetric Record*, 16(91):pages 51–66, 1998.

[15] Crow, F.C. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207–212. ACM Press, New York, NY, USA, 1984. ISBN 0897911385.

[16] Davison, A.J. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 1403. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 0-7695-1950-4.

[17] Davison, A.J., González Cid, Y. and Kita, N. Real-Time 3D SLAM with Wide-Angle Vision. In *Proc. IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon*. July 2004.

[18] Davison, A.J. and Kita, N. 3D Simultaneous Localisation and Map-Building Using Active Vision for a Robot Moving on Undulating Terrain. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition, Kauai*. IEEE Computer Society Press, December 2001.

[19] Davison, A.J. and Murray, D.W. Simultaneous Localization and Map-Building Using Active Vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):pages 865–880, 2002. ISSN 0162-8828.

[20] Davison, A.J., Reid, I.D., Molton, N.D. and Stasse, O. MonoSLAM: Real-Time Single Camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):pages 1052–1067, 2007. ISSN 0162-8828.

[21] Fitzgibbon, A.W. and Zisserman, A. Automatic 3D Model Acquisition and Generation of New Images from Video Sequences. In *Proceedings of European Signal Processing Conference (EUSIPCO '98), Rhodes, Greece*, pages 1261–1269. 1998.

[22] Forsyth, D.A. and Ponce, J. *Computer Vision: A Modern Approach*. Prentice Hall, August 2002. ISBN 0130851981.

[23] Hähnel, D., Fox, D., Burgard, W. and Thrun, S. A Highly Efficient FastSLAM Algorithm for Generating Cyclic Maps of Large-Scale Environments from Raw Laser Range Measurements. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*. 2003.

[24] Harris, C. and Stephens, M. A Combined Corner and Edge Detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151. 1988.

[25] Hecht, E. *Optics*. Addison Wesley, August 2001. ISBN 0805385665.

[26] Heikkila, J. and Silven, O. A Four-step Camera Calibration Procedure with Implicit Image Correction. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 1106. IEEE Computer Society, Washington, DC, USA, 1997. ISBN 0-8186-7822-4.

[27] Horn, B.K. *Robot Vision*. The MIT Press; MIT Press Ed edition, March 1986. ISBN 0262081598.

[28] Lepetit, V. and Fua, P. Monocular model-based 3D tracking of rigid objects. *Foundations and Trends in Computer Graphics and Vision*, 1(1):pages 1–89, 2005. ISSN 1572-2740.

[29] Lindeberg, T. Feature Detection with Automatic Scale Selection. *Int. J. Comput. Vision*, 30(2):pages 79–116, 1998. ISSN 0920-5691.

[30] Lowe, D.G. Object Recognition from Local Scale-Invariant Features. *Computer Vision, IEEE International Conference on*, 2:pages 1150–1157 vol.2, 1999.

[31] Lowe, D.G. Local Feature View Clustering for 3D Object Recognition. *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01)*, I:pages 682–688, 2001.

[32] Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):pages 91–110, 2004. ISSN 0920-5691.

[33] Montiel, J.M.M., Civera, J. and Davison, A.J. Unified Inverse Depth Parametrization for Monocular SLAM. In *Proceedings of Robotics: Science and Systems*. Philadelphia, USA, August 2006.

[34] Moravec, H.P. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Ph.D. thesis, Stanford University, Stanford, CA, USA, September 1980.

[35] Nüchter, A., Lingemann, K. and Hertzberg, J. 6D SLAM with Kurt3D. In *Robotic 3D Environment Cognition, Workshop at the International Conference Spatial Cognition*. Bremen, Germany, 2006.

[36] Nüchter, A., Lingemann, K., Hertzberg, J. and Surmann, H. 6D SLAM with Approximate Data Association. In *Proceedings of the IEEE International Conference on Advanced Robotics (ICAR '05)*, pages 242–249. July 2005.

[37] OpenCV 1.1 C Reference. `http://opencv.willowgarage.com/documentation/index.html`, August 2009 (access date).

[38] Pollefeys, M., Van Gool, L., Vergauwen, M., Verbiest, F., Cornelis, K., Tops, J. and Koch, R. Visual Modeling with a Hand-Held Camera. *International Journal of Computer Vision*, V59(3):pages 207–232, 2004.

[39] Sato, T., Kanbara, M., Yokoya, N. and Takemura, H. Dense 3-D Reconstruction of an Outdoor Scene by Hundreds-Baseline Stereo Using a Hand-Held Video Camera. *International Journal of Computer Vision*, 47(1-3):pages 119–129, 2002. ISSN 0920-5691.

[40] Se, S., Lowe, D.G. and Little, J. Vision-based Mobile Robot Localization And Mapping using Scale-Invariant Features. In *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2051–2058. May 2001.

[41] Se, S., Lowe, D.G. and Little, J. Mobile Robot Localization and Mapping with Uncertainty Using Scale-Invariant Visual Landmarks. *International Journal of Robotics Research*, 21:pages 735–758, 2002.

[42] Shi, J. and Tomasi, C. Good Features to Track. *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[43] Shoemake, K. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254. ACM Press, New York, NY, USA, 1985. ISSN 0097-8930.

[44] Shreiner, D., Woo, M., Neider, J. and Davis, T. *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, August 2005. ISBN 0321335732.

[45] Sprickerhof, J., Nüchter, A., Lingemann, K. and Hertzberg, J. An Explicit Loop Closing Technique for 6D SLAM. In *Conference on Mobile Robotics (ECMR)*, pages 229–234. 2009.

[46] Thrun, S. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring artificial intelligence in the new millennium*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. ISBN 1-55860-811-7.

[47] Thrun, S., Burgard, W. and Fox, D. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005. ISBN 0262201623.

[48] Valgren, C. and Lilienthal, A.J. SIFT, SURF and Seasons: Long-term Outdoor Localization Using Local Features. In *Proceedings of the European Conference on Mobile Robots (ECMR)*, pages 253–258. Freiburg, Germany, September 19–21 2007.

[49] Viola, P. and Jones, M. Robust Real-time Object Detection. *International Journal of Computer Vision*, February 2002.

[50] Zhang, Z. Flexible camera calibration by viewing a plane from unknown orientations. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 1:pages 666–673 vol.1, 1999.

List of Figures

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Osnabrück, October 2009