Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis Nr. 98

Vision and SLAM on a highly dynamic mobile two-wheeled robot

Patrick Suhm

Course of Study:	Technische Kybernetik
Examiner:	Prof. Dr. rer. nat. Marc Toussaint
First Supervisor:	Prof. Dr. rer. nat. Marc Toussaint
Second Supervisor:	M. Sc. Stefan Otte
Commenced:	18.07.13
Completed:	17.12.13
CR-Classification:	I.2.9, I.4.3, I.4.8, B.1.0

Abstract

This thesis examines a sparse feature based visual monocular simultaneous localization and mapping (SLAM) approach with the intension of stabilizing a two-wheeled balancing robot. The first part introduces the basics like camera geometry, image processing and filtering. Further on, the thesis treats the details of a monocular SLAM system and shows some specialties to keep the computational effort low. The last part deals with Andrew Davison's "*SceneLib*" library and how it can be used to obtain the camera state vector.

Zusammenfassung

Die vorliegende Arbeit gibt einen Einblick in das Thema der auf wenigen Bildfeatures basierenden simultanen Lokalisierung und Karten Erstellung (SLAM) mittels monokularer Kamera zum Zwecke der Regelung eines zweirädrigen balancierenden Roboters. Im ersten Teil werden grundlegende Themen wie die Kamerageometrie, Bildverarbeitung und Filtertechniken besprochen. Darauf aufbauend werden im zweiten Abschnitt Details und effizienzsteigernde Maßnahmen erläutert, die ein monokulares Echtzeit-Kamera-SLAM System möglich machen. Im letzten Teil der Arbeit wird beschrieben wie mittels Andrew Davisons „*SceneLib*" Bibliothek die aktuelle Kamera Pose bestimmt werden kann.

Content

Chapter 1

Introduction

1.1 Motivation

The American company "Segway Inc." is known all over the world. They manufacture a two-wheeled self-balancing electrical vehicle called "Segway PT". These devices are quiet successful in niche markets such as transportation for police departments, military bases, warehouses, corporate campuses and industrial sites. The control strategy of a Segway can be compared to the one of the human body. In order to move forward, the center of gravity is shifted in front. After reaching the targeted speed or to stop its motion, the human body, takes action and positions its feet back under the torso. The Segway does pretty much the same thing, except it has wheels instead of legs, a motor instead of muscles, a collection of microprocessors instead of a brain and a set of tilt sensors instead of an inner ear balance system. The Segway's sensors for tilt measurement are gyroscopes and accelerometers. This so called inertial measurement unit (IMU) is the most widely used approach for devices that perform inertial navigation through dead reckoning. They are used in cars for electronic stability control, in aircrafts, satellites and many other technical devices.

The human body, though, shows another way to keep balance. With a bit of training it's possible to maintain balance, only through visual information, even when, for example through a disease, the inner ear balance system doesn't work correctly.

This thesis, examines the visual approach. Unlike the human body which is due to its two eyes, able to get a stereoscopic sight of the world, the robot used in this thesis will only have a monocular camera serving as its sensor information. This limitation makes the pose data acquisition process a bit more challenging. To obtain depth information through one monocular camera, is only possible, when the camera is in motion and it observes the scene from more than one view. And even this is something the human body can do. People which lost one of their eyes are still able to estimate the depth and get a three dimensional impression of the world. In order to do this, the brain builds some sort of map and locates certain points or objects observed from different angels. This map is constantly updated and maintained so that these people are finally able to get an intention of their location and the three dimensional structure of their environment.

In the total awareness that a PC and a piece of code can hardly compare with the human brain, it's quiet surprising that with some tricks and simplification to lower the computational effort, the localization and pose estimation with one monocular camera works pretty well.

For means of just stabilizing a two wheeled mobile robot with a monocular camera, an approach like the one examined in this thesis, could seem a bit long winded. Especially when considering the fact that for an estimation of the angular velocity and tilt, it would be sufficient to analyze the optical flow, in a manner depicted in [Baraldi89]. But that would, at the best, be enough to keep the robot in balance.

When dealing with the topic of localization and pose estimation in an unknown environment, one will inevitably encounter the theory of simultaneous localization and mapping (SLAM). The theory of SLAM is well understood in mobile robotics research and despite the fact that there would probably have been an easier and faster approach, with respect to only controlling the robots tilt, with the help of visual monocular SLAM, the ability of localization and environmental exploration comes on top.

1.2 Why Monocular SLAM?

For a lot of applications laser range finders are the sensor of choice when it comes to localization and dense map building in robotics. However, for determining the pose of a balancing robot, it is neither necessary to have a dense map, nor would it be appropriate to lug a heavy and expansive laser scanner. So, apart from rangefinders (including sonar, laser, infrared and time-of-flight cameras), normal cameras can be used, to construct 3 dimensional spars maps in real-time.

In this regard, basically 2 different types of cameras can be distinguished: Stereo and monocular cameras. Stereo systems consist of two cameras which are arranged in a fixed position to each other and observe the scene. Via triangulation stereo cameras are able to obtain 3D information from the 2 dimensional data they perceive. But there are some disadvantages. First of all stereo cameras are more expansive. Furthermore two cameras are heavier than one and the necessity of a stiff connection between them must also not be neglected. When the frame is not stiff enough, errors occur through vibration and deformation in general. This can be especially important in mobile robotics. Aligning and calibrating (rectifying) two cameras is tricky and the results will depend on the accuracy of that work.

Besides this, a stereo camera is always designed for a specific range, depending on the resolution and the distance of the camera centers. So they're not very adaptable to surroundings with far and near objects.

All these disadvantages bring up another idea in vision based pose estimation. In this approach 3D information is obtained through motion. If the pose of a monocular camera is known at every time step, moving the camera can deliver 3D information. In case of cameras mounted on top of a robot, the current camera pose is normally not known, but can roughly be estimated by other sensor information like odometry. Cameras which are hand held and freely maneuvered don't have this additional information.

Pose estimation is crucial for single camera approaches because depth measurement depends on the camera pose. That's why single camera SLAM is even harder than SLAM with a laser-rangefinder or a stereo camera. A correct localization and pose estimation is not only necessary to build a consistent map, but also to obtain measurements in the first place.

A properly working real time monocular SLAM algorithm consists of many steps which were all tuned and adapted with respect to computational efficiency. This thesis was highly inspired by the work of Andrew Davison, who is a pioneer in the visual monocular SLAM. He presented the first real-time monocular camera SLAM algorithm in 2003 [Davison03]. In this publication, Davison's method of choice for camera pose and landmark estimation is the Extended-Kalman-Filter (EKF). Image patches serve as landmark description. The so called inverse depth parameterization [Davison07] allows initializing features in the EKF without delay. A detailed explanation of Davison's *monoSLAM* system is given in chapter 5.2.

Before dealing with the actual monocular SLAM topic, some basics have to be discussed in the Camera Geometry, Image Processing and Filtering chapters below. The information presented in these basic chapters are, as long as not declared differently, taken from one of the following sources: [MultViewGeo], [CVAlgoAndApplic], [Sola07].

Chapter 2

Camera Geometry

This section shows the relation between 2D pixel-coordinates on the camera sensor and their counterpart in the 3D world scene. The first part deals with the projection of a 3D scene point onto the 2D image plane, which will turn out as the easier of both directions. For this purpose a pinhole camera model is introduced and the basic projection equations are given with the help of homogeneous coordinates. Furthermore a distortion model is explained, to take the processes into account which are occurring in a real lens. The second part of the Camera Geometry chapter will deal with the inverse projection, trying to make up for the loss of one dimension and showing how to get 3D information back from 2D pixel-coordinates.

2.1 Pinhole Camera Model

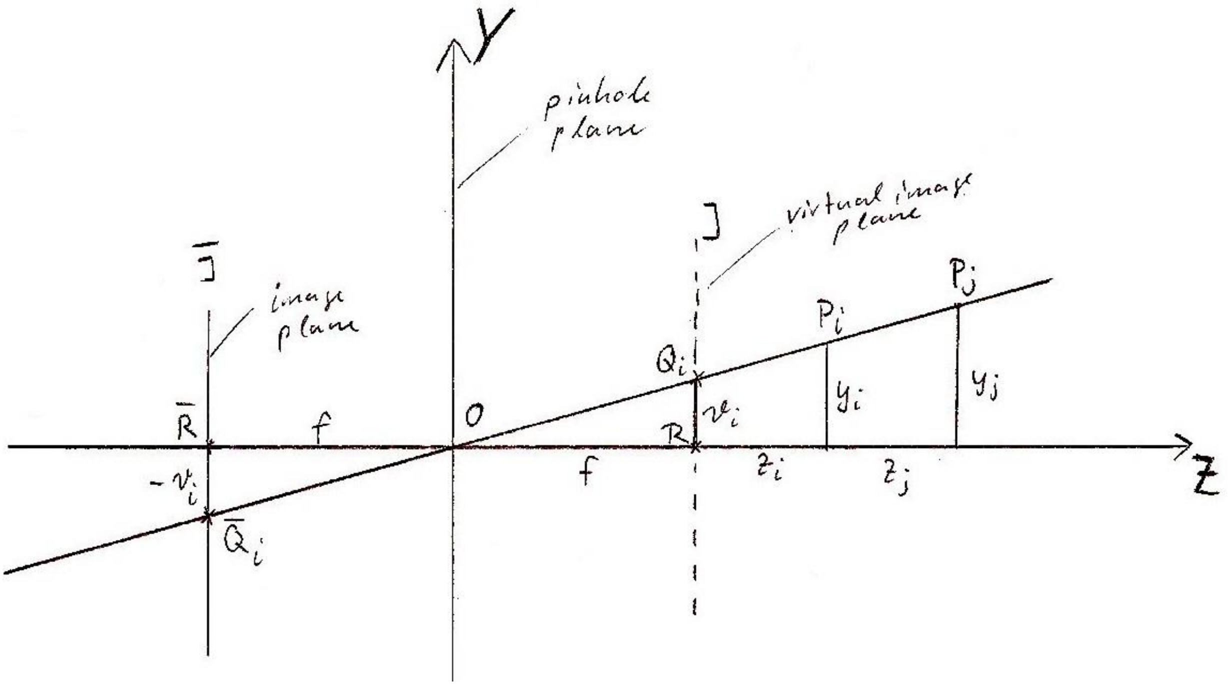


Figure 1: Pinhole Camera Model

The pinhole camera model is the basic camera model in computer vision. It consists of an infinitely small hole at the optical center O. The optical center is located on the pinhole plane, at the intersection point with the optical axis Z. The image plane $\bar{I}$ is situated behind the pinhole plane in a distance f, called the focal distance. The intersection of the Z -Axis with the image plane is called the principle point $\bar{R}$, or the image center. Light rays, were projected through the optical center into the 2D image plane. This image is mirror-inverted and upside down. That's why one considers a virtual image plane I which is located in front of the pinhole plane at the distance f.

For the virtual image plane the following holds:

$$\begin{pmatrix} \frac{u_i}{f} \\ \frac{v_i}{f} \end{pmatrix} = \begin{pmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \end{pmatrix} \xrightarrow{yields} \begin{pmatrix} u_i \\ v_i \end{pmatrix} = f \cdot \begin{pmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \end{pmatrix} \tag{2.1}$$

u_i *and* x_i are the distances in X direction. X completes Z and Y to a right handed orthogonal coordinate frame.

(2.1) is a mapping from $R^3 \rightarrow R^2$ so that point $P_i = (x_i\,; y_i\,; z_i)$ is projected in point $Q_i = (u_i\,; v_i)$. But also every other point P_j on that straight line is mapped in Q_i.

A naïve approach to get the position of the 3D point P_i from the virtual image point Q_i could simply be, to invert formula (2.1).

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

(2.2) shows, that point P_i depends on z_i and f. The focal distance f is a characteristic of the camera and not a problem to determine. To find the depth z_i is the tricky part and that's what later explanations will be about. In the following the easy 3D to 2D camera projection is further treated.

2.2 Pixel Coordinates

Till now all distances were considered in metric units. The camera chip however consists of pixels. These pixels, not necessarily, have the same dimensions in u- and v-direction. To deal with this, the focal distances $f_u = m_u \cdot f$ and $f_v = m_v \cdot f$ are introduced. m_u and m_v are the pixel densities in u and v direction and have the unit $\frac{pixel}{m}$.

Another point that hasn't been considered yet is the position of the image origin C. If it's not identical with the principle point, c_u and c_v have to be added. In this context c_u and c_v can also be used to correct a badly placed image sensor. The unit of c_u and c_v is *pixel* and from now on the coordinates u and v will also have the unit *pixel.* They are built as shown in formula (2.3).

$$\binom{u}{v} = \begin{pmatrix} f_u \cdot \frac{x}{z} + c_u \\ f_v \cdot \frac{y}{z} + c_v \end{pmatrix} \tag{2.3}$$

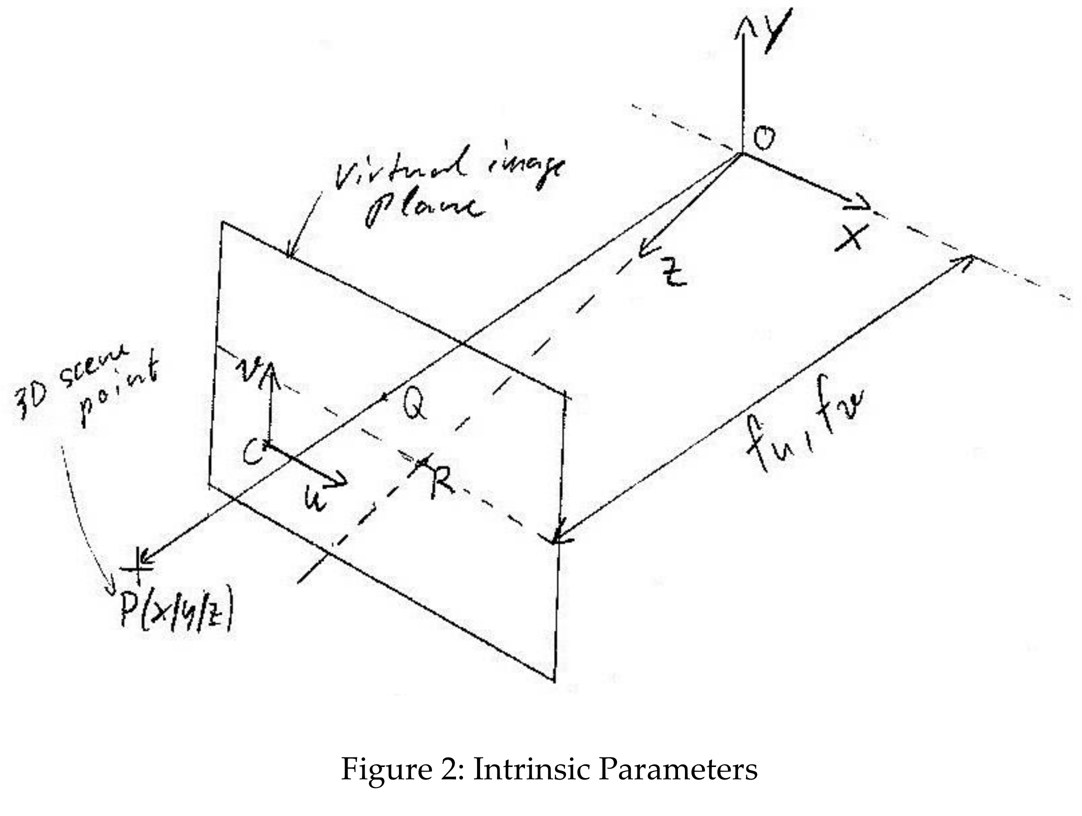


Figure 2: Intrinsic Parameters

Note that the pixel coordinates u and v are integer values and only perceivable of the image sensor if, u and $v \in Z^{+}$ and $0 \leq \mathrm{u} < \text{sensorwidth}$ and $0 \leq \mathrm{v} < \text{sensorheight}$, holds. Of course formula (2.3) doesn't deliver the rounded integer values for u and v. The result of (2.3) would be for example $(u;\ v) = (17{,}793;\ -208{,}496)\ pixel$. The rounding could now be modeled with a function $r(u;\ v)$ which would deliver the result: $(u;\ v) = (18;\ -208)\ pixel$.

For reasons of convenience and due to the fact that in the interesting direction, from 2D -> 3D, modeling of rounding isn't an issue, it will also not be modeled in the direction 3D ->2D. Keep in mind that in the following the perceived values for (u; v) are considered as integer values even if it's not explicitly modeled.

The new parameters f_u, f_v, c_u, c_v used in (2.3) are called intrinsic parameters. Besides these there are other parameters which influence the projection in a pinhole camera. The next section shows an elegant approach to handle perspective camera projections and introduces the six, so called extrinsic camera parameters.

2.3 Perspective Projection with Homogeneous Coordinates

A point in an n-dimensional projective space can be constructed through the points on a line, crossing the origin in the n+1-dimensional space. A 3D point $(x_E; y_E; z_E)$ has the 4D homogeneous coordinates $(x_h,; y_h; z_h; w)$ or $(x_E; y_E; z_E; 1)$, with $w \in R\backslash\{0\}$ .The advantages of this notation are that points at infinity can be represented with finite numbers and the whole process of camera projection can be represented with linear algebra. With the help of homogenous coordinates it's possible to describe a scene point in different coordinate frames in an elegant way.

Now formula (2.3) can be rewritten in homogeneous coordinates:

$$z \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f_u & 0 & c_u & 0 \\ 0 & f_v & c_v & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

(2.4)

$$\mathbf{K} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}; \quad \boldsymbol{P}_{3x4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.5}$$

$\boldsymbol{K}$ is the matrix of intrinsic parameters. It's also called camera calibration matrix. $\boldsymbol{P}$ is the projection matrix and $\boldsymbol{K} \cdot \boldsymbol{P}_{3x4}$ produces the matrix shown in formula (2.4).

Figure 3 shows the virtual image plane, the camera coordinate frame with origin O and optical axis Z, a world coordinate frame with origin W and a 3D scene point P.

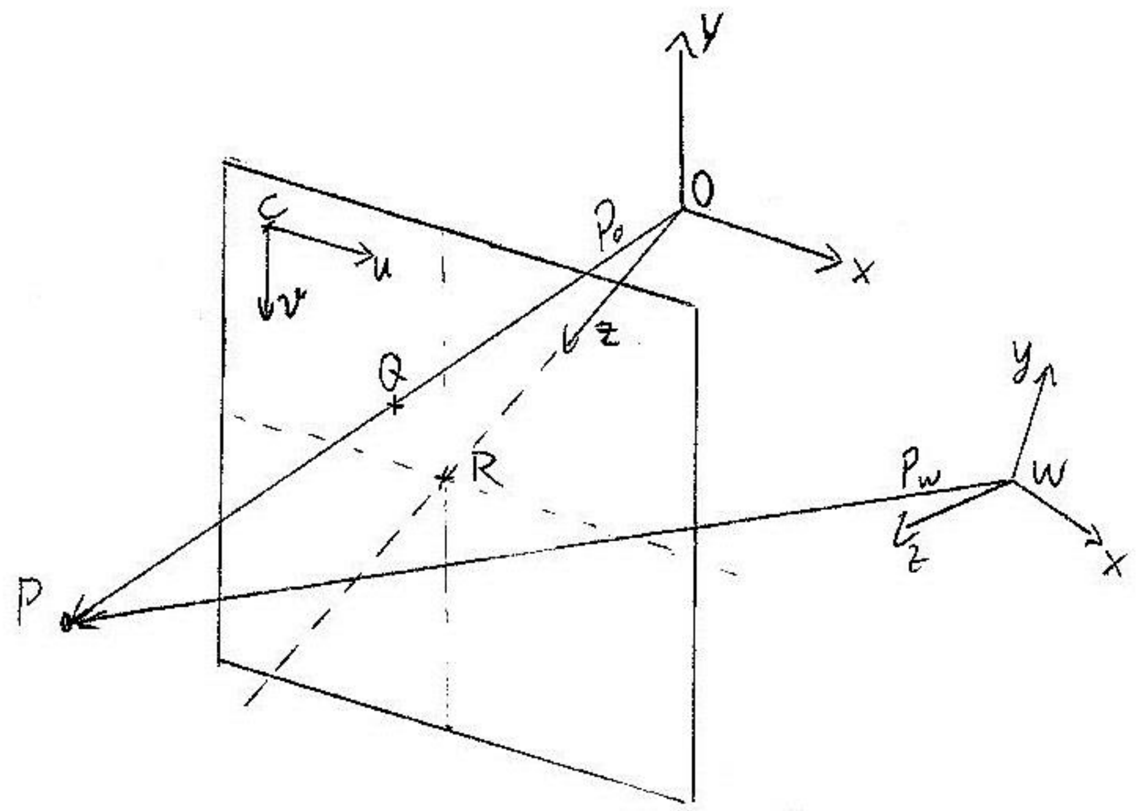


Figure 3: Camera Rotation and Translation

The scene point P with respect to the camera frame O written in homogeneous coordinates is $P^O = (x^O;\ y^O;\ z^O;\ 1)$. P with respect to the world frame is $P^W = (x^W;\ y^W;\ z^W;\ 1)$.

The transformation from $P^W\ to\ P^O$ can be written as: $P^O = \boldsymbol{T}^{OW} \cdot P^W$ with $\boldsymbol{T}^{OW} = \begin{bmatrix} \boldsymbol{R}^{OW} & t^{OW} \\ 0^T & 1 \end{bmatrix}$. Where $\boldsymbol{R}^{OW}$ is a $3x3$ rotation matrix and t^{OW} is a $3x1$ translation vector.

Now it's possible to write down the complete transformation from scene point P in the world coordinate frame to coordinates in the image plane C:

$$z^O \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{R}^{OW} & t^{OW} \\ 0^T & 1 \end{bmatrix} \begin{pmatrix} x^W \\ y^W \\ z^W \\ 1 \end{pmatrix} \tag{2.6}$$

Neglecting distortion and skew there are four intrinsic parameters f_u, f_v, c_u, c_v, three rotation angles defined in $\boldsymbol{R}^{OW}$ and three parameters of translational displacement, defined in t^{OW}. The six parameters defined in $\boldsymbol{R}^{OW}$ and t^{OW} are called extrinsic camera parameters. This makes a total of 10 parameters to describe the perspective projection in a basic pinhole camera.

2.4 Lens Distortion

The infinitely small hole of a pinhole camera makes sure that the light, coming from a point in the external world, will impact within a very tiny region of the image plane. However, the smaller the pinhole, the less light reaches the sensor. This means extremely sensitive, and therefore noisy, sensors are necessary or the exposure time has to be very high. For most applications these limitations are not acceptable. This is why lens cameras are used. The equations which describe the pinhole camera can still be used to describe a perfect lens camera. This chapter introduces a simple distortion model to handle the radial distortion of real lenses.

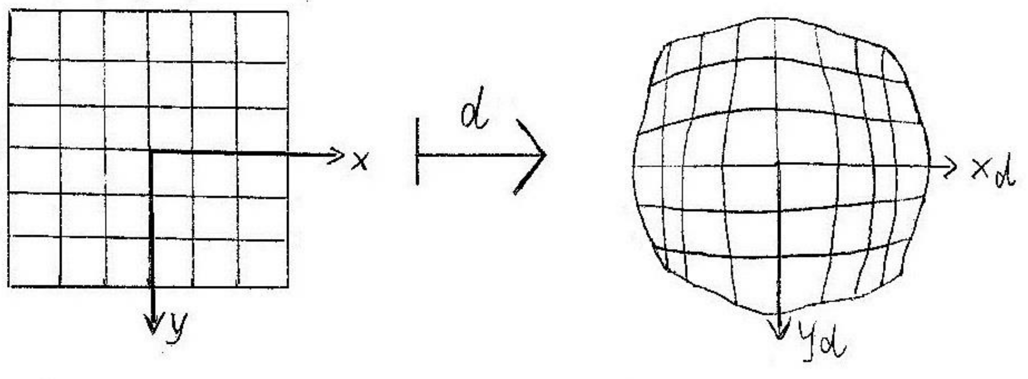


Figure 4: Radial Distortion

Radial distortion is caused by imperfections of the lens. Usually the effects of radial distortion become stronger the larger the distance between a projected point and the image center R is. The following basic radial distortion model can be regarded as a function $d(\cdot)$:

$$R^2 \to R^2; \quad (x_Q^O; y_Q^O) \to d(x_Q^O; y_Q^O) = (x_{Qd}^O; y_{Qd}^O) \tag{2.7}$$

$$\begin{pmatrix} x_{Qd}^O \\ y_{Qd}^O \end{pmatrix} = (1 + d_2 r^2 + d_4 r^4 + \cdots) \cdot \begin{pmatrix} x_Q^O \\ y_Q^O \end{pmatrix}; \quad r = \sqrt{(x_Q^O)^2 + (y_Q^O)^2} \tag{2.8}$$

Where $(x_Q^O; y_Q^O)$ is the tuple of coordinates depicting image point Q with respect to the camera coordinate frame O. $(x_{Qd}^O; y_{Qd}^O)$ is the tuple of distorted coordinates with respect to the camera coordinate frame.

Note that the distortion doesn′t model integer values but real numbers. It is applied in the modeling process before the intrinsic matrix is applied.

The number of coefficients $\{d_2; d_4 \ldots\}$ which are necessary to fulfill the requirements depend on the used lens. Wide angle lenses can require up to three coefficients, while in most cases less than three will be enough.

2.5 Camera Calibration

In the chapters *"Pixel coordinates"* and *"lens distortion"* a total of at least 12 parameters were introduced to describe the internal and external processes of a real camera in contrast to an ideal one. The method to determine these parameters is called camera calibration. It's a crucial step in image processing and there are special tools for this purpose. As one of the most powerful tools for all vision related topics, the OpenCV library also contains methods for camera calibration purposes. In chapter 6 of this thesis, the camera calibration is done by using the *Matlab* toolbox [Calib]. It is worth noting that the computation of the internal camera calibration parameters can occur simultaneously with the extrinsic parameters which describe the pose of the camera relative to the calibration target.

A classical approach like the "Roger Y. Tsai Algorithm" [Tsai87] uses a calibration pattern, which is oftentimes a flat chequered rectangular plate much like a chessboard. This plate needs to be photographed from different angles. The set of distorted images can then be used to determine the camera parameters. The quality of the calibration process will increase with the number of images taken from the calibration pattern. Note that, through the simultaneous estimation of extrinsic and intrinsic parameters, the arrangement of camera and calibration pattern is arbitrary as long as the whole pattern is depicted in each image.

Another approach which needs no special calibration target is called camera auto-calibration. Auto calibration means that the intrinsic camera parameters are determined directly from multiple distorted images showing unstructured scenes. It is also possible to auto-calibrate a sensor from a single image, when for example, parallel lines or concentric circles are identified and information about the parallelism or concentricity of these objects is available.

To describe the calibration process in detail would definitely go beyond the scope of this thesis. For further information, the calibration chapters of [MultViewGeo] and [CVAlgoAndApplic] are highly recommended.

2.6 The Perspective Camera Model

The basic camera model, developed in the previous sections can now be presented as the following.

At the beginning, a scene point P_i with respect to world frame W, is transformed to the camera frame O. Afterwards it's projected in the image plane. The next step is to apply the nonlinear distortion function. And in the last step, the distorted coordinates were transformed into pixel-coordinates through a multiplication with the intrinsic matrix.

Projection:

$$\begin{pmatrix} x_Q^O \\ y_Q^O \\ 1 \end{pmatrix} = \frac{1}{z^O} \cdot \boldsymbol{P}_{3x4} \cdot \boldsymbol{T}^{OW} \cdot P^W \tag{2.9}$$

Distortion:

$$\left(x_{Qd}^O; y_{Qd}^O\right) = d(x_Q^O; y_Q^O) \tag{2.10}$$

Pixel Mapping:

$$P^C = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{K} \cdot \begin{pmatrix} x_{Qd}^O \\ y_{Qd}^O \\ 1 \end{pmatrix} \tag{2.11}$$

2.7 The Inverse Perspective Camera Model

Now that a basic camera model exists and the equations of the projection into the camera plane are announced, it's time to depict the inverse projection process. In the following this process will be called back projection. Back projection will be executed through the following step. Firstly, the pixel mapping will be reversed. The next step will be to revoke the nonlinear process of distortion. The first two steps were invertible, the third, namely the projection process, is not.

Formula (2.2) already gives an intuition of that problem. It shows that, due to the loss of one dimension, the back projection process can only be defined up to an unknown scale factor. An approach to find this factor is described in the Triangulation chapter.

The inversion of the pixel mapping is simply done by an inversion of the intrinsic camera matrix.

$$\begin{pmatrix} x_{Qd}^{O} \\ y_{Qd}^{O} \\ 1 \end{pmatrix} = \boldsymbol{K}^{-1} \cdot P^{C} = \begin{bmatrix} \frac{1}{f_u} & 0 & -\frac{c_u}{f_u} \\ 0 & \frac{1}{f_v} & -\frac{c_v}{f_v} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2.12}$$

The correction of the distortion is, due to the nonlinear function of 4th grade and higher, a bit more challenging. The naïve approach to simply invert formula (2.8) isn't feasible for polynomial higher 4th grade.

$$R^2 \to R^2;\ \left(x_{Qd}^{O}; y_{Qd}^{O}\right) \to \left(x_{Q}^{O}; y_{Q}^{O}\right) = d^{-1}\left(x_{Qd}^{O}; y_{Qd}^{O}\right) \tag{2.13}$$

Fortunately there are other possibilities to un-distort an image. Probably the most commonly used method is to create a lookup table which relates every distorted pixel coordinate to its ideal relative. Note that within the monocular SLAM system described later on, distortion correction is done only for the interesting feature points and not for the whole picture in advance. This is simply because the un-distortion of the whole picture would need too much time. Operations, like feature matching and feature finding, are also possible within the distorted image.

The last step of 2D->3D process is the back projection:

$$P^W = \boldsymbol{T}^{WO} \cdot \boldsymbol{P}_{4x3} \cdot z^O \cdot \begin{pmatrix} x_Q^O \\ y_Q^O \\ 1 \end{pmatrix} \tag{2.14}$$

Entry (4; 3) of the projection matrix $\boldsymbol{P}_{4x3}$ has to be modified with $\frac{1}{z^0}$ in order to prevent a back-projection to infinity. z^O is the feature depth with respect to the camera coordinate frame.

$$\boldsymbol{P}_{4x3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & \frac{1}{z^0} \end{bmatrix} \tag{2.15}$$

$$T^{WO} = \begin{bmatrix} \boldsymbol{R}^{WO} & t^{WO} \\ 0^T & 1 \end{bmatrix} = \begin{bmatrix} (R^{OW})^T & -(R^{OW})^T \cdot t^{OW} \\ 0^T & 1 \end{bmatrix} \tag{2.16}$$

So, according to formula (2.14) the point P with respect to the world coordinate frame is:

$$P^W = \boldsymbol{T}^{WO} \cdot \begin{pmatrix} z^0 \cdot x_Q^O \\ z^0 \cdot y_Q^O \\ z^0 \\ 1 \end{pmatrix} \tag{2.17}$$

The next chapter will show how, with the means of triangulation, it's possible to gather depth information.

2.8 Triangulation

For a static environment it's not important whether the obtained images are sampled by one camera at two different positions (and therefor at two different times) or by two cameras at the same time, known as stereo vision. The triangulation process with one camera is depicted in figure 5.

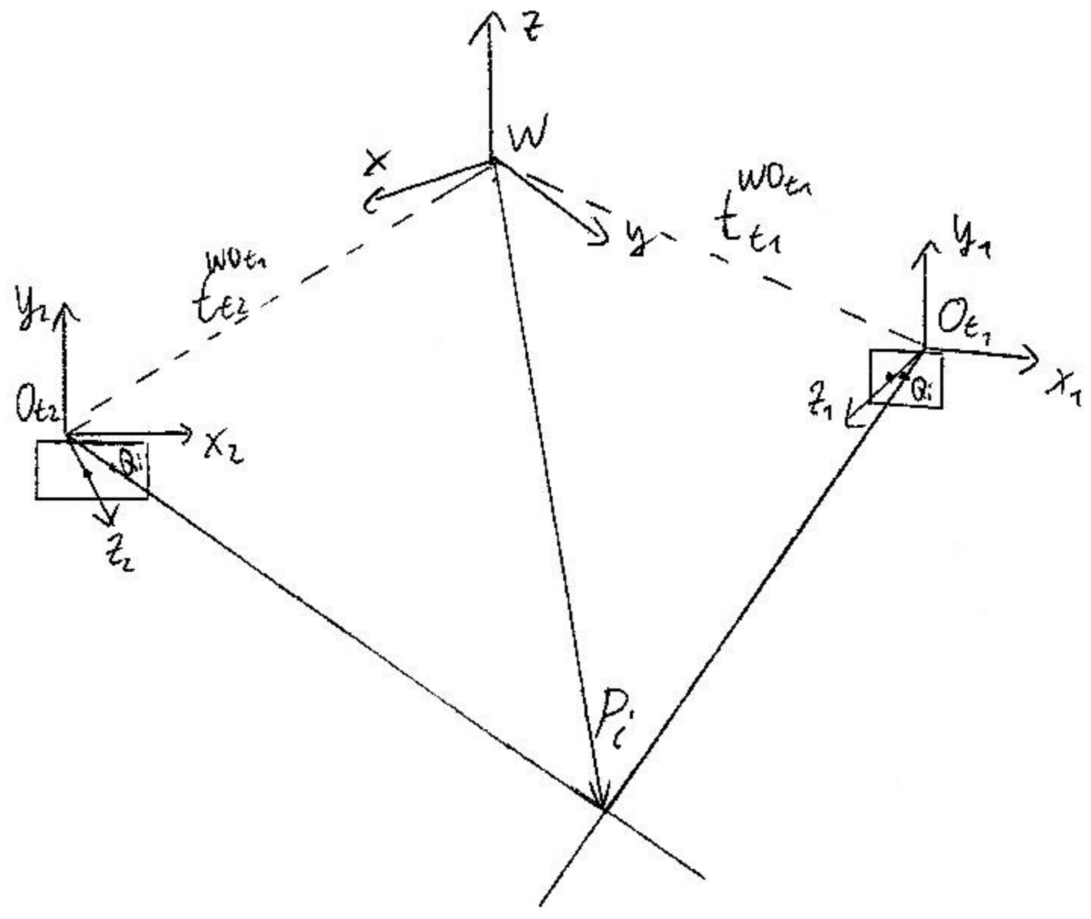


Figure 5: Triangulation

Neglecting the fact that, in the special case of visual monocular SLAM the camera pose $\boldsymbol{T}^{WO}$ isn't available and therefor has to be estimated together with the depth of the scene point, the following equations describe the triangulation process.

Observing feature P_i at the time t_1 yields with respect to the world coordinate frame the following line equation:

$$g_{t1}: \quad P_{i,t1}^{W} = \boldsymbol{R}^{WO_{t1}} \cdot z_{t1} \begin{pmatrix} x_{Q_i}^{O_{t1}} \\ y_{Q_i}^{O_{t1}} \\ 1 \end{pmatrix} + t_{t1}^{WO_{t1}}$$

(2.15)

$\left(x_{Q_i}^{O_{t1}}\ y_{Q_i}^{O_{t1}}\ 1\right)^T$ is the vector of real valued coordinates with respect to O_{t1} after the inverse intrinsic matrix was applied.

Observing P_i at the time t_2 yields with respect to the world coordinate frame:

$$g_{t2}: \quad P_{i,t2}^{W} = \boldsymbol{R}^{WO_{t2}} \cdot z_{t2} \begin{pmatrix} x_{Q_i}^{O_{t2}} \\ y_{Q_i}^{O_{t2}} \\ 1 \end{pmatrix} + t_{t2}^{WO_{t2}}$$

(2.16)

For a static environment we obtain the two unknown parameters z_{t1} and z_{t2} by finding the intersection between g_{t1} and g_{t2}. Note that the method of directly intersecting two lines and therefor defining z_{t1} and z_{t2} remains a theoretical wish. In reality, through several uncertainties, it's only possible to find an estimation for a scene point which minimizes the distance to both lines. Later chapters will depict the triangulation process in a probabilistic manner. It will be shown that a completely static environment isn't necessary under all circumstances and that there's no need to model the triangulation process explicitly. The state of the camera center and the involved landmarks will be handled within the EKF.

Chapter 3

Image Processing

The geometric considerations done in the previous chapter were based on the assumption that features, observed by the camera, are from infinitely small spatial size and unambiguously perceptible and identifiable, from the current camera pose. How realistic these assumptions were and how features can be found within an image will be content of the following section.

3.1 Feature Detection

Gray scale images, and those are the images which are used in visual SLAM, can be seen as a function $R^2 \rightarrow R\,;(u,v) \rightarrow I(u,v)$. In most cases, the intensity function $I(u,v)$ has a codomain of 8 bit. That means every pixel can represent 256 different gray values. After the image had been acquired from the camera and converted to gray scale, it can be thought of as a matrix with $u_{max} - u_{min}$ columns and $v_{max} - v_{min}$ rows.

For sparse feature based visual monocular SLAM systems, collecting information in an image means, determine a set of locally distinguishable and unique points, the so called feature points or interest points, and find them in another image, taken from the same scene but from a slightly different pose. Speaking of the uniqueness of a point, what does that mean? In a gray scale image it's possible to have hundreds of pixels with the same intensity. Describing an interesting pixel only by means of its intensity is by far not enough to create a valuable feature. It is therefore necessary to regard an area around the promising pixel, it's so called neighborhood.

It shows that edges, in the sense of sudden Intensity changes along a line, are quiet good features. However, they still have a disadvantage regarding their spatial indetermination. It is therefore that corners, the intersection of edges, are used as interest points.

The basic idea to detect edges and corners in an image is to choose points where the spatial derivatives are locally maximal. If the derivative is locally maximal in more than one direction, the investigated point is likely to be a corner.

One way to approximate a partial derivative is to convolve the image with the following masks.

$$\partial_u \approx \frac{1}{2h}(-1\ 0\ 1)\ ;\ \ \partial_v \approx \frac{1}{2h}\begin{pmatrix}1\\0\\-1\end{pmatrix} \tag{3.1}$$

Expressed in coordinates of an image I, the approximation of a derivative in direction u and v can be written as:

$$I_u(u,v) = I(u+1,v) - I(u-1,v) \tag{3.2}$$

$$I_v(u,v) = I(u,v+1) - I(u,v-1) \tag{3.3}$$

An improved version of (3.1) would be the perpendicular smoothing *Scobel* mask:

$$\boldsymbol{S_1}:\ \ \boldsymbol{\partial_x} \approx \frac{1}{4}\begin{pmatrix}1\\2\\1\end{pmatrix} * \frac{1}{2h}(-1\ 0\ 1) \ = \ \frac{1}{8h}\begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix} \tag{3.4}$$

$$\boldsymbol{S_2}:\ \ \boldsymbol{\partial_y} \approx \frac{1}{4}(1\ 2\ 1) * \frac{1}{2h}\begin{pmatrix}1\\0\\-1\end{pmatrix} = \frac{1}{8h}\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix} \tag{3.5}$$

In the equations above, h is the grid size or pixel distance and $*$ is the convolution operator.

So far, these masks produce an approximation for a derivative, but no corner detection.

In order to gather information about the structure direction within a neighborhood $N_\sigma(u,v)$ of radius σ, the structure tensor H_σ is introduced:

$$H_\sigma(u,v) = \mathrm{K}_\sigma * \begin{bmatrix} I_u^2(u,v) & I_{uv}(u,v) \\ I_{vu}(u,v) & I_v^2(u,v) \end{bmatrix} \tag{3.6}$$

K_σ is a 2D Gaussian mask with standard deviation σ.

The orthonormal eigenvectors v_1, v_2 of H_σ specify the main structure direction within some integration scale σ. The corresponding eigenvalues λ_1, λ_2 describe the average contrast along these directions.

The eigenvalues of H_σ allow the following assumptions for the local image structure:

$\lambda_1 \approx \lambda_2 \approx 0 \rightarrow$ no structure

$\lambda_1 \gg \lambda_2 \approx 0 \rightarrow$ straight edges

$\lambda_1 \geq \lambda_2 \gg 0 \rightarrow$ corners

Since eigenvalue determination is computationally expansive, [Harris88] and [Shi94] introduced other methods to find corners. One of these methods is depicted in (3.7). Let t be a tunable threshold and m be a local maximum, it's likely to obtain a corner were the following holds:

$$trace\big(H_\sigma(u,v)\big) = \mathrm{K}_\sigma * I_u^2(u,v) - \mathrm{K}_\sigma * I_v^2(u,v) > t \quad and \quad \frac{\det\big(H_\sigma(u,v)\big)}{trace\big(H_\sigma(u,v)\big)} = m \tag{3.7}$$

3.2 Feature Descriptors

The last section has shown how to find corners in an image. This chapter will describe how they were actually used as features, with respect to the ability of finding them again in another image. Once a corner has been found its location can be described as $C_1^{t_1} = (u, v)$. This description, however, is useless in terms of finding $C_1^{t_1}$ in an image taken from another, unknown pose. If the pose was precisely known, it would be easy to find $C_1^{t_2}$ within the second image. It would simply be a transformation from one camera pose to another. Unfortunately the second camera pose is either fraught with uncertainty or not known at all and it's only possible to specify a region for the appearance of the feature. Therefor it's unavoidable to provide some sort of description which allows regaining a previously observed interest point from a new pose.

A feature descriptor is basically a vector which contains abstract information which allows to compare two interest points. Unlike image patches, which will be described later, the descriptor doesn't contain raw image data. It provides higher level information that is, depending on the used algorithms, invariant against scale variation, rotation, illumination changes, perspective and shift.

As there are many possibilities to describe a feature, a whole bunch of descriptors have been developed in computer vision. The "Scale Invariant Feature Transform" (SIFT) [Lowe99] and the "Speed Up Robust Feature" (SURF) [Bay06], are probably the two algorithms for feature description which are most commonly used. For means of visual monocular SLAM, such feature descriptors are rarely used, simply because they are quite computational expensive. Therefore they are not further regarded in this thesis. Instead the image patches are described now.

A patch W is a rectangular region of an image containing N pixels and the interest point as central pixel. The standard deviation of the patches luminosity values is σ and the mean is $\bar{I}$.

It's possible to find features in two different images, with the help of image patches, by executing the following steps. The first step is to determine a set of interest points in an image I by applying a corner detection algorithm with a certain threshold. Afterwards, image patches around the interest points are defined. The final step is then, to choose one patch of image I and compare it to an area in Image J, which is, regarding the actual camera motion, most likely for the appearance of the corresponding interest point. This procedure is described in the chapter 5.2.7.

3.3 Feature Matching

Two patches W_1 *and* W_2 are likely to represent the same feature, when their similarity measure function $S(W_1, W_2)$ is extreme. Similarity can for example be examined by the following measures.

Sum of Squared Differences:

$$S_{ssd} = \sum_{(u,v)\in W} (I(u,v) - J(u,v))^2 \tag{3.8}$$

A small S_{ssd} indicates similarity. It shows however, that the S_{ssd} has some drawbacks with patches under different illumination. A S_{ssd} comparison of two Images which show exactly the same scene but under different illumination $I(u,v) = k \cdot J(u,v)$ (k represents the illumination change) would deliver that the images aren't matching.

That's why another similarity measure is used for patch matching. The *cross correlation* is a widely used similarity measure in signal processing. The normalized and centered version is:

$$S_{zncc} = \frac{1}{N} \sum_{(u,v)\in W} \frac{(I - \bar{I}) \cdot (J - \bar{J})}{\sigma_I \cdot \sigma_J} \tag{3.9}$$

A maximum in the zero-mean normalized cross correlation indicates similarity. The normalization leads to: $S_{zncc}(I,J) \to [-1,1]$.

$S_{zncc} = 1$ means a perfect match whereas $S_{zncc} = -1$ is a perfect mismatch.

The S_{zncc} is a robust measure over long sequences where lighting conditions can change.

A computational more efficient implementation [Sola07] of the S_{zncc} is shown in (3.10).

$$S_{zncc} = \frac{N \cdot S_{IJ} - S_I \cdot S_J}{\sqrt{(N \cdot S_{JJ} - S_J^2)(N \cdot S_{II} - S_I^2)}} \tag{3.10}$$

With:

$$S_I = \sum_{(u,v)\in W} I(u,v), \qquad S_J = \sum_{(u,v)\in W} J(u,v), \qquad S_{II} = \sum_{(u,v)\in W} I^2(u,v),$$

$$S_{JJ} = \sum_{(u,v)\in W} J^2(u,v), \qquad S_{IJ} = \sum_{(u,v)\in W} I(u,v)J(u,v)$$

It is worth noting that S_I and S_{II} have to be computed only once for each patch of image I in the whole comparison process. So these two measures can be saved together with the image patch and serve as some kind of feature descriptor. In comparison to a SIFT feature descriptor, which has in general dimension 128, storing the whole patch plus S_I and S_{II}, needs in the case of reasonable patch sizes only few more memory size, but the generation of this descriptor is much more computational efficient.

A nice overview about available methods is given in the *"Feature detection and matching"* chapter of [CVAlgoAndApplic].

Chapter 4

Filtering

Estimating the state of a dynamical system is an important task not only in monocular SLAM. But why estimating and not measuring directly? Simply because oftentimes the measurements obtained from the system are noisy and the system description is inaccurate. It will be shown that an estimate can be produced by applying a filter which combines measurements and initially available system information.

In case of monocular SLAM, the system is the camera itself with its six degrees of freedom and the location of the landmarks with three degrees of freedom. All measurements obtained by the camera and therefor all calculations of the actual state of the system are larded with uncertainties. Fortunately, the theory of probability permits to take these uncertainties into account and solve the task of refining prior estimates through current, noisy measurements. So, the system state and the measurements can be described through a specific probability density function and not through deterministic values.

This chapter will give an explanation of a system under stochastic and deterministic influence and depict a solution of the filtering problem for the case of Gaussian uncertainties.

4.1 System Description

The behavior of a dynamic system can be described by mathematically expressing the available information. One possible way to formulate this is the discrete state space formalism.

The state x_{t-1} of a system X transitions to the state x_t under the influence of the uncertain control input w_t as depicted in (4.1):

Transition/Evolution Equation:

$$x_t = f_t(x_{t-1}, w_t) \tag{4.1}$$

The measurements y_t obtained at time t which are influenced by the measurements uncertainty v_t can be described with the help of function h_t:

Measurement Equation:

$$y_t = h_t(x_t, v_t) \tag{4.2}$$

As introduced in the section above, the knowledge about the probabilistic character in the modeling process can be described through density functions. The density function $p(w_t)$ expresses the perturbation which enters the system and is called system noise. $p(v_t)$ is called measurement noise and the density function which describes the uncertainties of the initial state is given through $p(x_0)$.

4.2 The Filtering Problem

In the theory of stochastic processes, the filtering problem can be formulated as:

Finding, at each time step t the best, or optimal estimate for the state of a system by regarding the whole history of noisy measurements obtained since the time $t = 0$.

At the beginning of the filtering process there is an initial guess x_0 for the systems state. This initial guess has an uncertainty which is modeled by the density function $p(x_0)$. As it's the beginning of the process there are no previous values for x and so $p(x_0)$ can be called the prior density at time $t = 0$. Now, that the procedure goes on and the first measurement of the system state is received, this prior can be corrected and is now called posterior density $p(x_0|y_0)$ of time $t = 0$. The prior density $p(x_1)$ for $t = 1$ is estimated with the help of the system equations and the knowledge about the perturbation which influence the system. After the next measurement process, the posterior density $p(x_1|y_1)$ at time $t = 1$ is calculated. This procedure is repeated to the time $t = k$ where it's the goal to receive $p(x_k|y_k)$ from the prior $p(x_{k-1})$.

Filtering can be interpreted as a series of prediction and correction steps. The terms posterior and prior are related to the arrival time of a measurement, which is the important event within filtering.

Normally the filtering process is synchronized to the reception of measurement data. When, for example, a camera is capable of delivering 30 frames per second, it makes sense to operate the prediction process at about 30Hz synchronous to the image input. Between two measurements the filter uses the dynamic of the perturbation free system equation.

Note that the posterior density function at time $t = k$ implicitly depends on all the previous measurements. For a real time application, like the visual monocular SLAM, it's crucial that the amount of necessary operations is quiet constant and doesn't increase over time. Luckily an incremental implementation considers all information from the beginning, but can be implemented in a recursive way, so that the state x_k is refined by the state x_{k-1} and the measurement y_k.

The best state estimation $\hat{x}_t$ received from the correction step, should then be the result of the application of an arbitrary quality criterion. The Kalman-Filter which is introduced in the next chapter is a minimum variance estimator. It returns the estimate $\hat{x}_k$ that minimizes the following expression.

$$E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \tag{4.3}$$

$E[\cdot]$ is the expectation operator. It can be shown that the estimate $\hat{x}_k$ which results from a minimization of expression (4.3) is equal to the conditioned mean of $p(x_k|y_k)$. [Sola07]

4.3 The Kalman-Filter

The recursive solution of the filtering problem for a linear time-discrete system under Gaussian uncertainties is delivered by the Kalman-Filter. [Kalman60]

The discrete evolution and measurement equation of a linear Gaussian system are:

$$x_t = \boldsymbol{F}x_{t-1} + u_t + \boldsymbol{G}w_t \tag{4.4}$$

$$y_t = \boldsymbol{H}x_t + v_t \tag{4.5}$$

$$p(x_0) = N(x_0 - \bar{x}_0\,; \boldsymbol{P_0}) \tag{4.6}$$

$$p(w_t) = N(w_t - 0\,; \boldsymbol{Q}) \tag{4.7}$$

$$p(v_t) = N(v_t - 0; \boldsymbol{R}) \tag{4.8}$$

F, **G** and **H** are constant system and measurement matrices of suitable size. **P**, **Q** and **R** are the so called covariance matrices and u_t is the deterministic control input.

$N(x-\bar{x}\,;\boldsymbol{P})$ denotes the density function of a normal distributed n-dimensional variable x with mean $\bar{x}=E[x]$ and covariance matrix $\boldsymbol{P}=E[(x-\bar{x})(x-\bar{x})^T]$ and is determined as shown in (4.9).

$$N(x-\bar{x}\,;\boldsymbol{P})=\frac{1}{\sqrt{(2\pi)^{\mathrm{n}}|\boldsymbol{P}|}}\exp(-0.5(x-\bar{x})^T\boldsymbol{P^{-1}}(x-\bar{x})) \tag{4.9}$$

The Kalman-Filter can be divided in a prediction and correction step:

4.3.1 KF Prediction

At time step $t=k$, according to the dynamic of the perturbation free and through u_k controlled system, the temporal estimate is received as:

$$\hat{x}'_k=\boldsymbol{F}\hat{x}_{k-1}+u_k \mathrel{\widehat{=}} E[x_k|y_{k-1}] \tag{4.10}$$

$$\boldsymbol{P}'_k=\boldsymbol{F}\boldsymbol{P_{k-1}}\boldsymbol{F^T}+\boldsymbol{G}\boldsymbol{Q}\boldsymbol{G^T} \mathrel{\widehat{=}} E[(x_k-\hat{x}'_k)(x_k-\hat{x}'_k)^T] \tag{4.11}$$

So expressed in the terms prior and posterior which were mentioned above, $\hat{x}'_k$ is the prior of time step $t=k$ and is received through applying the perturbation free transition equation to the posterior of time step $t=k-1$.

4.3.2 KF Correction

As the measurement has been received from the system, it can now be compared to the measurement expectation. At the time $t = k$, the difference of the measurement expectation according to the perturbation free measurement equation and the real measurement is called innovation z_k:

$$z_k = y_k - \boldsymbol{H} x'_k \tag{4.12}$$

The covariance matrix of z_k is:

$$\boldsymbol{Z_k} = \boldsymbol{H P'_k H^T} + \boldsymbol{R} \tag{4.13}$$

The prior of time step $t = k$ can now be improved by applying the follow equation which yields the posterior for the time $t = k$:

$$\hat{x}_k = \hat{x}'_k + \boldsymbol{K_k} z_k \tag{4.14}$$

The posterior covariance matrix $\boldsymbol{P_k} = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T]$ of the state estimation error is computed as:

$$\boldsymbol{P_k} = (\boldsymbol{I} - \boldsymbol{K_k H})\boldsymbol{P'_k} \tag{4.15}$$

$\boldsymbol{I}$ is the identity matrix. The matrix $\boldsymbol{K}$ is called the Kalman gain and corrects the prior proportionally to the innovation. It is computed as:

$$\boldsymbol{K_k} = \boldsymbol{P'_k H^T}\left(\boldsymbol{H P'_k H^T} + \boldsymbol{R}\right)^{\boldsymbol{-1}} = \boldsymbol{P'_k H^T Z_k^{-1}} \tag{4.16}$$

With $\hat{x}_k$ and $\boldsymbol{P_k}$ it's now possible to start the next filter run and estimate the prior for the time $t = k - 1$.

It shows that the Kalman-Filter produces the optimal estimation according to the equations above, when the assumptions (4.6) … (4.8) are fulfilled.

When $\hat{x}_0 = E[x_0]$ it's likely that the estimations $\hat{x}_k$ will be as expected from $E[\hat{x}_k] = E[x_k]$.

Note that a lot of computational effort can be saved by computing the matrices $\boldsymbol{P_k}$ and $\mathbf{K_k}$ offline because they don't depend on the measurements y_k. Furthermore the matrices $\boldsymbol{H}, \boldsymbol{F}, \boldsymbol{G}, \boldsymbol{R}$ and $\boldsymbol{Q}$ are often constant and known at the beginning of the filtering process in the majority of cases.

The offline computation of $\boldsymbol{P}_k$ and $\mathbf{K_k}$ can be achieved from the initial $\boldsymbol{P}_0$ through:

$\boldsymbol{P_0} \rightarrow \boldsymbol{P'_1} \rightarrow \boldsymbol{K_1} \rightarrow \boldsymbol{P_1} \rightarrow \boldsymbol{P'_2} \rightarrow \boldsymbol{K_2} \rightarrow \boldsymbol{P_2} \rightarrow \boldsymbol{P'_3} \rightarrow \boldsymbol{K_3} \rightarrow \boldsymbol{P_3} \ldots$

When all these matrices are known at the beginning, the Kalman-Filter can be reduced to one equation which yields the optimal estimate at the time t=k from the estimate at time t=k-1 and the control input u_k as:

$$\hat{x}_k = (\boldsymbol{I} - \boldsymbol{K}_\mathrm{k}\boldsymbol{H})(\boldsymbol{F}\hat{x}_{k-1} + u_k) + \boldsymbol{K}_k y_k \tag{4.17}$$

4.4 The Extended-Kalman-Filter

In reality hardly any system is linear and therefore the Kalman-Filter can't be applied in the majority of cases. An estimation-filter which can cope with nonlinear transition and measurement equations is the Extended-Kalman-Filter. It provides a solution of the filtering problem for nonlinear systems. Note that, while the Kalman-Filter is an optimal estimator, the Extended-Kalman-Filter is due to some linearization error not.

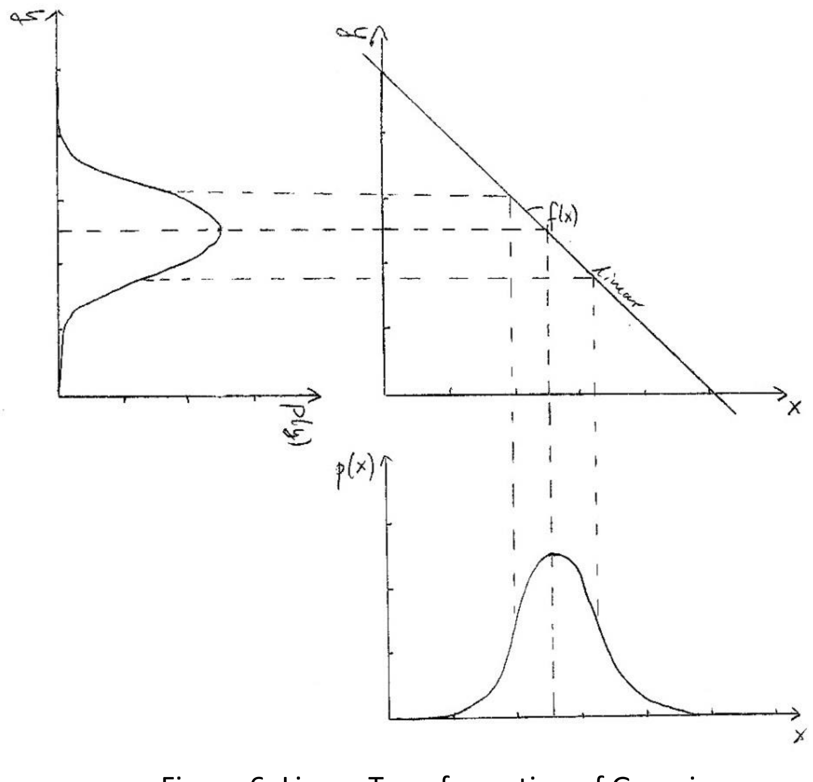


Figure 6: Linear Transformation of Gaussian

The transformation of a Gaussian distribution through a linear function results in a Gaussian distribution. An example for the 1-dimensional case of such a transformation is depicted in figure 6.

In the case of a nonlinear function the previously Gaussian distribution is not transformed into a Gaussian distribution any more. In figure 7, the nonlinear function $f(x)$ is almost linear in the region of confidence of the Gaussian-distributed x variable. However, even a weak nonlinearity in this interval, leads to a non-Gaussian probability density function for y. In this case, the Kalman-Filter would probably not deliver satisfying results.

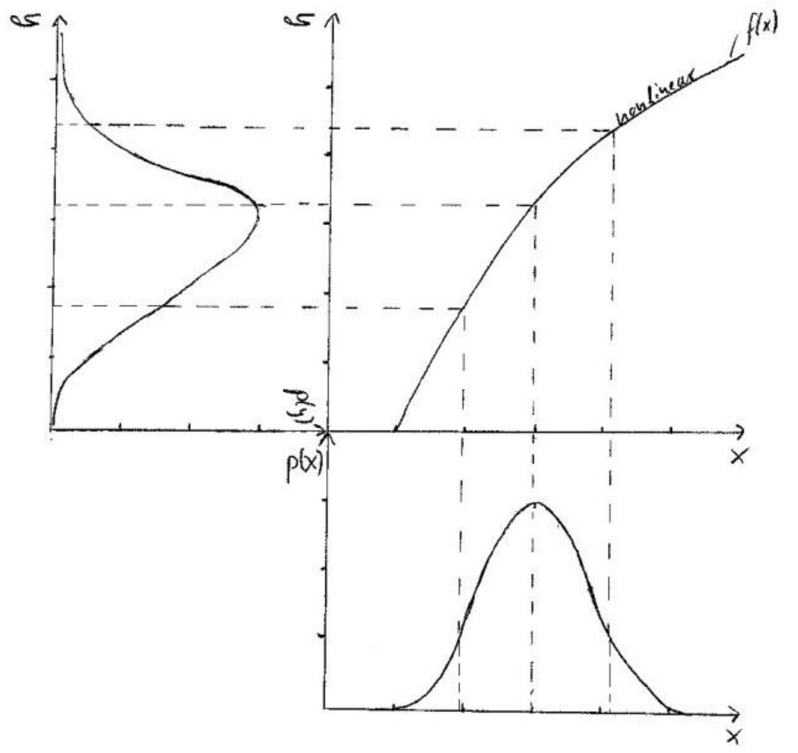


Figure 7: Nonlinear Transformation of Gaussian

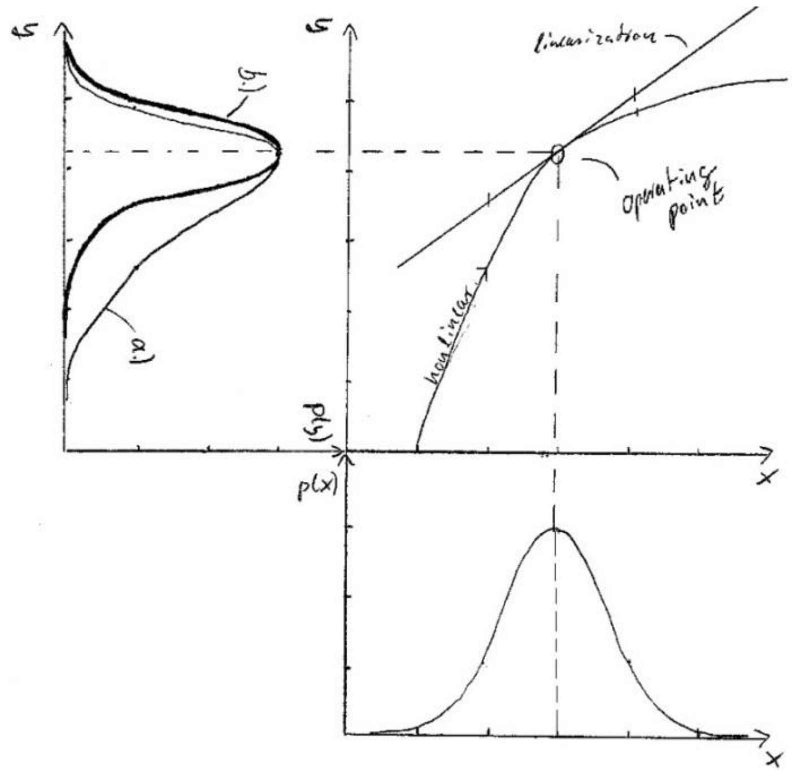


Figure 8: Linearization

In figure 8, the normal distributed density $p(x)$ is transformed through the nonlinear function into the non-Gaussian distribution a.). The idea for a filter algorithm that can be applied to nonlinear systems is to linearize around the operating point. This operating point is chosen for every time step as the state of the system which is most likely. After linearization, the Gaussian distribution is transformed through the linear function into the distribution labeled with b.).

After these considerations, the linear system depicted in equation (4.4) … (4.8) is now extended to a general nonlinear system which can be described with the following equations:

$$x_t = f(x_{t-1}, w_t) \tag{4.18}$$

$$y_t = h(x_t) + v_t \tag{4.19}$$

$$p(x_0) = N(x_0 - \bar{x}_0 ; \boldsymbol{P_0}) \tag{4.20}$$

$$p(w_t) = N(w_t - \overline{w}_t ; \boldsymbol{Q}) \tag{4.21}$$

$$p(v_t) = N(v_t - 0; \boldsymbol{R}) \tag{4.22}$$

The general case for the measurement equation would be: $y_t = h_t(x_t, v_t)$. But normally it's a quiet realistic assumption that the measurement noise v_t is additive [Sola07]. This leads to equation (4.19). Furthermore it is noticeable that the system perturbation w_t has mean 0. Therefore $\overline{w}_t$ shown in (4.21) turns out to be the deterministic control input which was called u_t in the section about the Kalman-Filter.

In order to linearize the system, its Jacobian matrices are built:

$$\boldsymbol{F_x} = \frac{\partial f}{\partial x} \; |_{\hat{x}_t, \overline{w}_t} \tag{4.23}$$

$$\boldsymbol{F_w} = \frac{\partial f}{\partial w} \; |_{\hat{x}_t, \overline{w}_t} \tag{4.24}$$

$$\boldsymbol{H} = \frac{\partial h}{\partial x} \, |_{\hat{x}_t}$$

$\hat{x}_t$ *and* $\overline{w}_t$ are the last best estimate of the system state and the current control input.

Now it's possible to process the prediction correction pattern as known from the Kalman-Filter.

4.4.1 EKF Prediction

A temporal system state estimation is achieved through:

$$\hat{x}'_k = f(\hat{x}_{k-1}, \overline{w}) \tag{4.25}$$

$$\boldsymbol{P}'_k = \boldsymbol{F}_x \boldsymbol{P}_{k-1} \boldsymbol{F}_x^T + \boldsymbol{F}_w \boldsymbol{Q} \boldsymbol{F}_w{}^T \tag{4.26}$$

4.4.2 EKF Correction

$$z_k = y_k - h(x'_k) \tag{4.27}$$

$$\boldsymbol{Z_k} = \boldsymbol{H P'_k H^T} + \boldsymbol{R} \tag{4.28}$$

$$\boldsymbol{K_k} = \boldsymbol{P'_k H^T}\left(\boldsymbol{H P'_k H^T} + \boldsymbol{R}\right)^{\boldsymbol{-1}} = \boldsymbol{P'_k H^T Z_k^{-1}} \tag{4.29}$$

$$\hat{x}_k = \hat{x}'_k + \boldsymbol{K_k} z_k \tag{4.30}$$

$$\boldsymbol{P_k} = (\boldsymbol{I} - \boldsymbol{K_k H})\boldsymbol{P'_k} \tag{4.31}$$

Note that similar to the Kalman-Filter, all Extended-Kalman-Filter matrices that are independent of the current measurement y_k can be computed offline at the beginning of the filtering process.

After linearization, the Extended-Kalman-Filter equations basically correspond to the ones known from the Kalman-Filter chapter. As mentioned above, the difference is that due to linearization errors, the EKF, unlike the KF, isn't an optimal state estimator any more. For most applications it will none the less deliver good enough performance and its easy and fast structure makes the EKF a widely used approach to estimate nonlinear system states.

4.5 The SLAM Problem

One of the basic problems in mobile robotics is called simultaneous localization and mapping. Before addressing more advanced tasks like reasoning or path planning, a mobile robot needs to know how its environment looks like and where it is located in this environment. If a map of the environment is available, the robot is, with the help of its sensors, able to localize itself in this map and afterwards use it to navigate in the unknown environment. If the robot knows its position relative to a fixed coordinate system, it can build a map by sensing features and relating them to the fixed coordinate system.

So, without the methods of simultaneous localization and mapping, a robot either needs to have a map at the beginning of its exploration or needs access to his absolute position. An estimate of the absolute position can be obtained through GPS for example. However, there are situations where neither an absolute position nor a map is available. Another point which should be regarded is that a map which isn't up to date, is quiet useless. So if a map of the environment is available, but not valid anymore because some obstacles for example changed their position, this can be dangerous. With the SLAM approach, the map is built incrementally and therefor contains the newest available data of the environment.

SLAM can be thought of as a chicken or egg problem: An unbiased map is needed for localization, while an accurate pose estimate is needed to build that map.

At the beginning of the SLAM process no map is available and the position of the robot is unknown. This scenario is mentioned in literature as the "kidnaped robot" scenario. The position at the beginning defines the origin of the navigation coordinate system, respectively the map. The next step would be to take a measurement. This measurement provides, depending on the used sensors, a more or less dense description of the environment. This description can then be related to the coordinate system defined at the beginning. When the robot moves it observes the same scene, but from a slightly different pose. The overlap of the observed field permits to estimate the robots movement. Additionally other sensors like odometry or an inertial measurement unit can deliver information about the robots movement. With the new position estimate, the robot is able to update the map and implement the new measurement data. So a map can be built incrementally and the position of the robot is estimated simultaneously.

The SLAM process verbally described above can be implemented in a lot of different ways. All approaches can basically be structured into the three parts: *perception*, *estimation* and *decision* [Sola07].

In visual SLAM, the *perception* part is about finding interest points in different images. This is called correspondence problem and was described in chapter 3 of this thesis. After this problem is solved, the *estimation* process depicted in chapter 4 can start. How many and which landmarks are stored in the map and the landmark initialization, conversion and deletion process is part of the *decision* step of a SLAM algorithm.

Chapter 5

Visual Monocular SLAM

The previous chapters have shown some basics which are necessary to understand the visual monocular SLAM. This section will now show the missing parts, put all components together and give a more detailed description of the necessary steps to yield a working visual monocular SLAM system.

5.1 Literature

SLAM is still an active field in mobile robotics. The estimation and filtering side is nowadays well understood and no knew methods are expected in this area. Therefore recent researches concentrate on the decision and perception area. The launch of Microsofts RGB-D (red green blue and depth) *Kinect* camera in 2010 enormously stimulated the research on the perception side. With the *Kinect* camera it's possible to build dense 3D maps in real time and render them with the image data. Besides the RGB-D SLAM, there are rangefinder (sonar, laser or infrared), visual monocular- and visual stereo- based SLAM approaches which have been developed over the last 15 years. As this work is about monocular camera SLAM, in the following mainly monocular systems are introduced.

An area which is closely related to camera based SLAM is the one called Structure form Motion (SfM). As SfM has so much in common with camera based SLAM, it is also briefly introduced in the section below.

5.1.1 Structure from Motion

The intention of visual SLAM algorithms is most often the localization in an unknown environment. Therefor only a sparse map is created and besides the localization task, this map is seldom used to solve high level problems like path planning or manipulation. So the creation of a map is a necessity to support the localization but not the goal itself.

There is another vision based technique, which shares a lot of the methods of visual SLAM, but with a slightly different goal. Structure from Motion systems aim at a, preferably accurate, reconstruction of the observed scene. In contrast to a visual SLAM system, SfM systems don't need to work in real time. The model of the environment produced by SfM systems usually is generated from an image sequence which shows a static scene from different positions. Not needing to work in real time also provides the possibility to optimize the structure estimation globally. This means an image sequence not necessarily has to be investigated sequentially, frame to frame, as it's the case in visual SLAM, but separated parts of the whole sequence, can be used together for the optimization. This optimization process is known as "bundle adjustment". With SfM it's possible to build 3D models of a scene, which can then be further examined with object recognition algorithms to yield a comprehension of the scene. Another application could be to model the inside of buildings and render it with the RGB information to receive an intuitive map for human orientation and navigation. This could be an expansion of *google maps* for the inside of buildings. There are quiet powerful applications conceivable and so the visual SLAM benefits from the development in the SfM area. A good overview about the SfM topic is given in [CVAlgoAndApplic].

5.1.2 Visual Monocular SLAM

The earliest approach in the field of monocular SLAM was probably the DROID system in 1987 by [Harris87]. This system was far ahead of its time and the authors continuously improved it, so that they could present a real-time implementation view years later. Although the DROID system was able to perform visual monocular SLAM with long image sequences and build 3D maps with many features, it had one big disadvantage. It neglected correlation between features and handled the estimation of every feature position in a separate EKF. That meant, loop closures didn't improve the estimation of all features and therefore the DROID system suffers from drift. Later approaches by [Ayache91] or [Beardsley95] also presented working visual monocular SLAM systems but they also neglected correlation. [Smith87] and [Moutarlier89] were the first who came up with the idea of one single state vector together with one covariance matrix. In the early 1990s, the believe that estimates obtained by regarding correlations between features through one covariance matrix wouldn't converge, limited the efforts to implement a so called "joint state" approach.

The breakthrough for visual SLAM systems with joint state came with the evidence of the convergence of this procedure in 1995 [Durrant06]. Not least through the increase in computational power, most recent approaches in the visual monocular SLAM area all use joint state approaches.

Among the latest systems the so called *monoSLAM* system [Davison07] is quiet popular. This system enables to localize a monocular freely moved camera in real-time and is based on the EKF. It is a further development of Davison's active stereo camera system [Davison98] and his publication from 2003 [Davison03]. [Chekhlov06] presented a solution of the monocular SLAM problem which is quiet similar to Davison's approach but uses Histograms of Oriented Gradients (HOG) as feature descriptor. Another recent system is the vSLAM [Karlsson05] which bases on a particle filter and the SIFT feature descriptor.

A comparison of some interesting visual SLAM approaches is given in the following table:

Approach	**Year**	**Camera-type**	**Feature Descriptor**	**Joint State**	**Filter**	**DoF**	**Specific features**
DROID [Harris87]	1993	monocular and stereo	edges	uncorrelated	EKF	6	-
Active Vision [Davison89]	1998	stereo	patches	correlated	EKF	3	actuated cameras
MonoSLAM [Davison03]	2003	monocular	patches	correlated	EKF	6	real-time capable
Mono-Camera-Tracking [Pupilli05]	2005	monocular	patches	partly correlated	particle filter	6	-
vSLAM [Karlson05]	2005	monocular	SIFT	correlated	Fast-SLAM	3	odometry
Monocular SLAM [Chekhlov06]	2006	monocular	HOG	correlated	Un-scented KF	6	odometry
MonoSLAM [Davison07]	2007	monocular	patches	correlated	EKF	6	inverse-depth

5.1.3 Camera Pose Tracking with Dense Methods

All systems presented above are based on features and sparse maps. Monocular systems that use dense maps but still rely on feature based tracking of the camera pose are for example [Newcombe10] or [Stuehmer10]. Besides these, there are other methods, completely relinquishing feature extraction by using dense methods that consider all pixels in an image and exploiting all available information through global optimization.

One of these systems is the so called "Parallel Tracking and Mapping" (PTAM) [Klein07]. The authors present a system which splits tracking and mapping into two separate tasks, which can then be processed in parallel threads on a multicore processor. This procedure allows them to use batch optimization techniques (Bundle Adjustment) which are, due to their computational effort, usually not found in real time systems. The results are detailed, dense maps with thousands of landmarks and real time camera tracking with high accuracy.

Another approach in the field of dense tracking is the "Dense Tracking and Mapping" (DTAM) system presented by [Newcombe11] in 2011. Their system, unlike any other real-time monocular SLAM system, creates a dense 3D surface model and immediately uses it for dense camera tracking via whole image registration.

5.2 The *MonoSLAM* System from Davison

There are various possibilities how to realize a visual monocular SLAM system in detail. Besides the obvious big topics like filter algorithm or feature descriptors, a whole bunch of other problems have to be solved. Andrew Davison's *monoSLAM* system offers some interesting detail solutions which help to make it computational efficient and therefore real-time capable. The basics introduced in the chapters above are now supplemented to get to a working system.

5.2.1 State Representation

As depicted in chapter 4.1, the mathematical expression of the behavior of a dynamical system can be obtained by using a discrete state space formalism. Davison suggest using the state vector $\boldsymbol{x}_c$, depicted in (5.1), to represent the state of the camera and the vector $\boldsymbol{x}_{l_i}$, depicted in (5.2), to represent the state of the landmark i. This vectors need to encode all necessary information to describe the system completely.

$$\boldsymbol{x}_c = \begin{pmatrix} \boldsymbol{r}^W \\ \boldsymbol{q}^W \\ \boldsymbol{v}^W \\ \boldsymbol{\omega}^W \end{pmatrix} \tag{5.1}$$

The $\boldsymbol{r}^W$ component of (5.1) denotes the position of the camera optical center in the world coordinate system W. $\boldsymbol{q}^W$ is the unit quaternion which depicts the camera orientation relative to the world frame. Note that the redundancy in the quaternion part of the state vector means that a normalization needs to be performed at each update step of the EKF [Davison03]. This normalization also influences the covariance matrix through a corresponding Jacobian calculation and multiplication according to formula (4.23) and (4.26) in the EKF chapter.

$\boldsymbol{v}^W$ encodes the linear velocities of the camera along the coordinate axes of W and $\boldsymbol{\omega}^W$ is the vector of angular velocities between world and camera coordinate frame. This means $\boldsymbol{x}_c$ is a 13 dimensional vector.

$$\boldsymbol{x}_{l_i} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \tag{5.2}$$

The landmark state vector $\boldsymbol{x}_{l_i}$ consists of the x_i, y_i *and* z_i component of the landmark i relative to the world coordinate system.

So the total state vector of the whole system is:

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x_c} \\ \boldsymbol{x_{l_1}} \\ \vdots \\ \boldsymbol{x_{l_i}} \end{pmatrix} \tag{5.3}$$

Note that the size of $\boldsymbol{x}$ is time-variant and depends on the number of landmarks which are stored in the map.

5.2.2 Joint State

An important discovery in the field of SLAM was made in the early 1980th. Till then, researchers in the SLAM area mainly avoided algorithms which use correlations between state variables. The fact, that there are strong correlations between different landmarks and the robot position and that this information is important and shouldn't be neglected, had been noticed before. However, due to the computational effort and the insecurity of the convergence of a correlation based approach, it remained mainly unused until the early 1990th [Durrant06]. An intuition of these correlations can be obtained by regarding that an uncertainty in the estimate of the current robot position influences not only the next robot position, but also all the observed landmarks and all landmarks that will be observed later on. So it's profitable to combine the robot position and the landmark's position in one state-vector and express the correlations in one matrix.

$$\boldsymbol{P} = \begin{bmatrix} P_{x_c x_c} & P_{x_c x_{l_1}} & P_{x_c x_{l_2}} & \cdots \\ P_{x_{l_1} x_c} & P_{x_{l_1} x_{l_1}} & P_{x_{l_1} x_{l_2}} & \cdots \\ P_{x_{l_2} x_c} & P_{x_{l_2} x_{l_1}} & P_{x_{l_2} x_{l_2}} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{5.4}$$

5.2.3 System Model and State Transition

Modeling the motion of a hand held monocular camera is different from modeling a camera mounted on top of a wheeled robot, moving on a plane. In the case of a wheeled robot, the control inputs which drive the motion are available whereas in the case of a freely maneuvered camera, this information is missing. Furthermore, through the lack of any geometric constraints which would simplify the motion model, or in the case of a robot moving on a plane, would remove one dimension, no simplification is possible here.

Theoretically, the behavior of a dynamical system can be modeled with infinite precision. With such a perfect system model, it would be possible to calculate even future states of a system with no uncertainty. In practice, however, it's only possible to model a system with limited precision. In order to take these modeling uncertainties, the so called process noise, into account, a probabilistic system consideration is beneficial. The EKF approach described above, like every other probabilistic approach, needs a transition function a measurement function and some information about the kind of uncertainties, provided through the covariance matrices. The way how one state transitions to another state is depicted below. There is an infinite amount of possibilities to model the camera motion.

[Davison03] proposes the following approach for the nonlinear system model. In each time step Δt, an unknown, zero-mean, Gaussian-distributed accelerations $\boldsymbol{a}^W$ and angular acceleration $\dot{\boldsymbol{\omega}}^W$ affect the camera. These accelerations cause an impulse of velocity which can be depicted with the following noise vector:

$$\boldsymbol{n} = \begin{pmatrix} \boldsymbol{V}^W \\ \boldsymbol{\Omega}^W \end{pmatrix} = \begin{pmatrix} \boldsymbol{a}^W \\ \dot{\boldsymbol{\omega}}^W \end{pmatrix} \cdot \Delta t \tag{5.4}$$

A state transition from t to $t+1$ under a disturbance $\boldsymbol{n}$ can be modeled through:

$$\boldsymbol{f} = \begin{pmatrix} \boldsymbol{r}_{t+1}^{W} \\ \boldsymbol{q}_{t+1}^{W} \\ \boldsymbol{v}_{t+1}^{W} \\ \boldsymbol{\omega}_{t+1}^{W} \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}_{t}^{W} + (\boldsymbol{v}_{t}^{W} + \mathbf{V}_{t}^{W})\Delta t \\ \boldsymbol{q}_{t}^{W} \otimes \boldsymbol{q}((\boldsymbol{\omega}_{t}^{W} + \boldsymbol{\Omega}_{t}^{W})\Delta t) \\ \boldsymbol{v}_{t}^{W} + \boldsymbol{V}_{t}^{W} \\ \boldsymbol{\omega}_{t}^{W} + \boldsymbol{\Omega}_{t}^{W} \end{pmatrix} \tag{5.5}$$

$\boldsymbol{q}\left((\boldsymbol{\omega}_{t}^{W} + \boldsymbol{\Omega}_{t}^{W})\Delta t\right)$ is the quaternion which depicts the rotation of the camera frame C around the rotation vector $(\boldsymbol{\omega}_{t}^{W} + \boldsymbol{\Omega}_{t}^{W})\Delta t$.

5.2.4 Inverse Depth Feature Parameterization

The easy and intuitive landmark description depicted in (5.2) shows some drawbacks when it comes to initializing a new landmark and adding it to the EKF. This is why the inverse depth parameterization is introduced now.

It would be possible to use the XYZ representation of a landmark when the initial depth estimate would be quiet accurate and the measurement equation would be tolerably linear. In fact, the more linear the measurement equation is and the better the initial guess was, the better is the performance of the Extended-Kalman-Filter. Or in other words: If the initial estimate of the state is terribly wrong and the measurement equation is highly nonlinear, the linearization error ensures that the filter will quickly diverge.

The first few frames that show a new landmark, especially in low parallax situations, deliver little information about the landmarks depth. The depth can only be modeled through a distribution that rises sharply at a well-defined minimum depth to a peak, but then declines very slowly [Montiel06]. That means it is quiet hard to tell whether a landmark has depth 10 units or depth 1000 units. The only possible statement is that the landmark is likely to be farther than 10 units. This is way to less when considering that ideally a narrow Gaussian distribution is aimed. A commonly used approach to get a quite accurate initial (with respect to the time when a new feature is inserted in the EKF) depth is the delayed initialization. This means, after the first observation of the landmark, it is not inserted into the EKF immediately, but further information is collected till the depth uncertainty is low enough and the depth can be modeled as a Gaussian distribution.

So, newly observed landmarks are treated differently from the others and special algorithms, independent from the main EKF estimation loop, are applied to reduce depth uncertainty. In his earlier approaches, Davison used a particle filter to explicitly refine the depth estimate of new landmarks over several frames, before inserting them in the EKF loop. The particle filter is more robust when it comes to initializing new landmarks with a depth which is far from the real depth.

No matter which algorithm is used to refine the depth at the beginning, the disadvantage of all delayed initialization approaches is that, while held outside the main loop, the landmarks cannot contribute to the estimation of the camera state for some serious amount of time. Furthermore, some of the landmarks, that show little or no parallax during motion, such as far ones or ones that are close to the motion epipole, never reach the point where their depth uncertainty is low enough and therefor never contribute to the state estimation with the EKF.

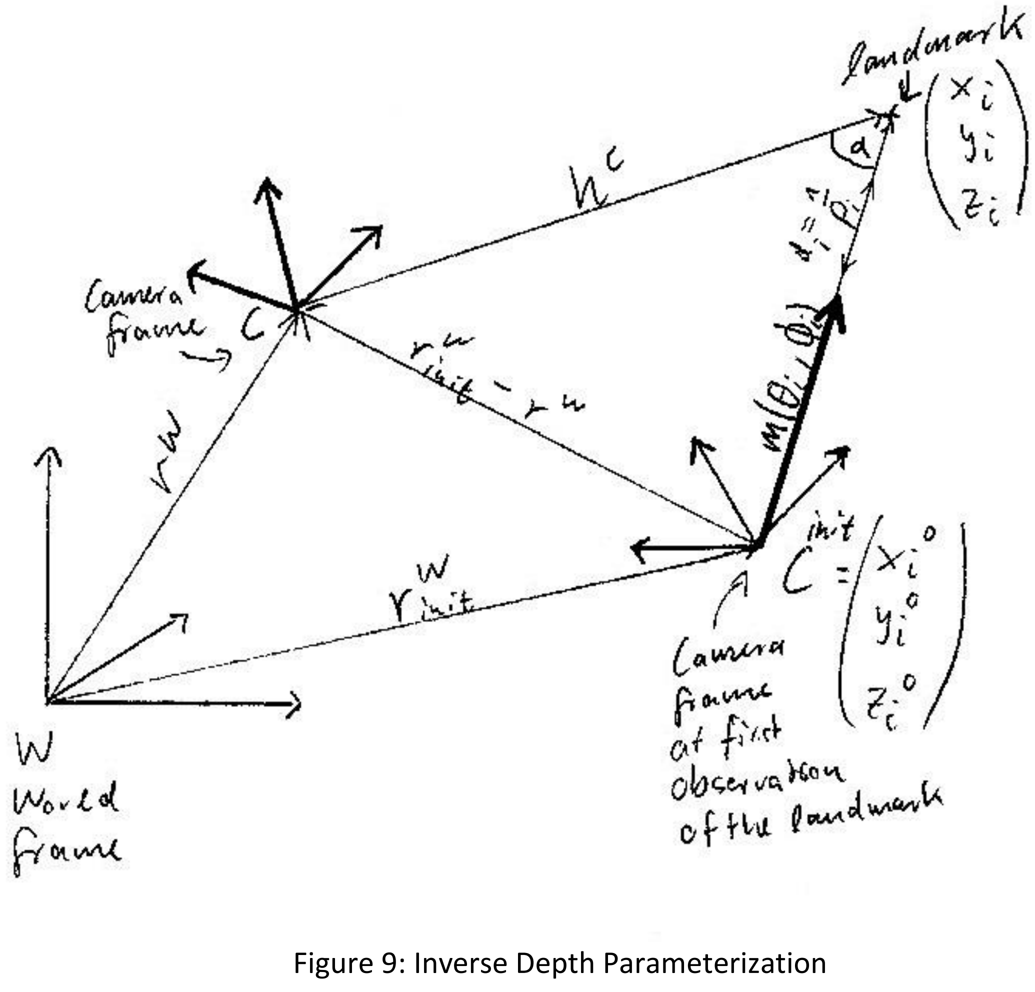


Figure 9: Inverse Depth Parameterization

This is why [Montiel06] suggested to use the inverse depth feature parameterization which has two main benefits. It shows that, in inverse depth coding, the measurement equation linearization error at low parallax is lower than in XYZ coding. The second advantage is that inverse depth coding allows handling landmarks at all depth, from very close to infinity, within the standard EKF framework and with no special treatment. Besides these great advantages, the only drawback of this kind of feature representation is the increased state vector size and therefor, the increase in computational effort. But the authors of [Montiel06] also proposed a solution for this problem. They introduce a linearity threshold that allows switching from inverse depth to XYZ representation.

A landmark in inverse depth parameterization has the following appearance:

$$\boldsymbol{x}_{l_i}^{\rho} = \begin{pmatrix} x_i^0 \\ y_i^0 \\ z_i^0 \\ \Theta_i \\ \Phi_i \\ \rho_i \end{pmatrix} \tag{5.4}$$

$\boldsymbol{r}_{init}^{W} = \left(x_i^0\ y_i^0\ z_i^0\right)^T$ is the camera position from which the feature was observed at the first time. Θ_i, Φ_i are the azimuth and elevation of the vector $\boldsymbol{m}(\Theta_i, \Phi_i)$ obtained in the initial camera frame C^{init}. The depth of the landmark from its first observation to the current position along the vector $\boldsymbol{m}(\Theta_i, \Phi_i)$ is encoded with $d_i = 1/\rho_i$. So the landmark description in XYZ coding can be obtained from the inverse depth representation with the following formula.

$$\boldsymbol{x}_{l_i}^{W} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} x_i^0 \\ y_i^0 \\ z_i^0 \end{pmatrix} + \frac{1}{\rho}\boldsymbol{m}(\Theta_i, \Phi_i)^W \tag{5.5}$$

$$\boldsymbol{h}^{C_{init}} = \boldsymbol{m}(\Theta_i, \Phi_i)^{C_{init}} = \begin{pmatrix} sin\Theta_i cos\Phi_\mathrm{i} \\ -sin\Phi_i \\ cos\Theta_i cos\Phi_i \end{pmatrix}$$

(5.6)

5.2.5 Measurement Equation

If $\boldsymbol{h}^C$ is the measurement vector depicted in the camera coordinate system and $\boldsymbol{R}^{CW}$ is the rotation matrix transforming a vector from the world system to the Camera system, the following equation holds:

XYZ-Coding:

$$\boldsymbol{h}^C_{XYZ} = \boldsymbol{R}^{CW}\begin{pmatrix} x_i \\ y_i - \boldsymbol{r}^W \\ z_i \end{pmatrix}$$

(5.7)

Inverse Depth Coding:

$$\boldsymbol{h}^C_\rho = \boldsymbol{R}^{CW}\left(\begin{pmatrix} x_i^0 \\ y_i^0 \\ z_i^0 \end{pmatrix} + \frac{1}{\mathrm{\rho_i}}\boldsymbol{m}(\Theta_i, \Phi_i)^W - \boldsymbol{r}^W\right)$$

(5.8)

Note that the camera doesn't observe $\boldsymbol{h}^C$ directly, but its projection in the image plane, according to formula (2.2) from chapter 2.

The difference between (5.7) and (5.8) is that in (5.8) a point at infinity can be modeled through $\mathrm{\rho_i} = 0$.

A modification of equation (5.9) would be to substitute $\frac{1}{\mathrm{\rho_i}}$ with d_i and therefore get the ability to code a point at zero depth.

The parallax angle α is defined between $\boldsymbol{m}(\Theta_i, \Phi_i)^{\boldsymbol{W}}$ and $\boldsymbol{h}^C$. If the parallax angel is small, both rays are almost parallel. This is the case for distant landmarks or when the camera moves slowly or not at all. Rewriting equation (5.8) yields:

$$\boldsymbol{h}^C = \boldsymbol{R}^{CW}\left(\rho_i\left(\begin{pmatrix} x_i^0 \\ y_i^0 \\ z_i^0 \end{pmatrix} - \boldsymbol{r}^W\right) + \boldsymbol{m}(\Theta_i, \Phi_i)^W\right) \tag{5.9}$$

For a low parallax angle α, the term $\boldsymbol{r}_{init}^W - \boldsymbol{r}^W$ is close to zero and $\boldsymbol{h}^C$ can be approximated as:

$$\boldsymbol{h}^C \approx \boldsymbol{R}^{CW}(\boldsymbol{m}(\Theta_i, \Phi_i)^W) \tag{5.10}$$

This means, no information about the camera translation can be obtained from this landmark i. The measurement equation only provides information about the camera orientation and the direction of i. The low parallax of landmark i is coded in a shifting of ρ_i towards zero which decreases depth uncertainty and leads to the conclusion of, either a far landmark, or one that lies next to the motion epipole. If i is distant, the inverse depth ρ_i confirms this by decreasing further and further. So, even when the depth cannot be estimated because of a distant landmark, some information about its depth has been coded. If the landmark is nearby the motion epipole, some parallax will eventually be detectable which then leads to a narrowing Gaussian distribution.

5.2.6 Feature Conversion and Linearity

Despite all the advantages when it comes to initialization, the inverse depth parameterization leads to a doubled landmark state vector size. This is a big drawback because it means that that the computational costs for landmark handling are four times higher than in XYZ parameterization. So while the XYZ coding shows drawbacks in the initialization without prior knowledge, inverse depth coding isn't suitable to discribe the landmark in the long term.

The solution, suggested in [Civera07], is to switch between different parameterizations. Therefore a threshold which allows to distinguish the moment where it is save to convert from one to another depth coding needs to be found.

As depicted in the EKF chapter, it shows, that when a function, for example the measurement function, is linear within an interval around the mean of a Gaussian distributed random variable, the projection of this random variable is also Gaussian. So, in order to switch to the XYZ parameterization and still obtain Gaussian distributed measurements, the moment has to be determined, when the measurement function is linear enough.

The linearity of a function f can be determined by examining its first derivative. If a function is linear in an interval, its first derivative is constant in that interval.

Let x be a Gaussian random variable in the notation of (4.9). The interesting interval in which about 95% of all values lie is then: $[\bar{x} - 2\sigma; \bar{x} + 2\sigma]$

The Taylor expansion for the first derivative is:

$$\frac{\partial f}{\partial x}(\bar{x} + \Delta x) \approx \frac{\partial f}{\partial x}|_{\bar{x}} + \frac{\partial^2 f}{\partial x^2}|_{\bar{x}} \cdot \Delta x \tag{5.11}$$

An idea of the linearity of a function within a certain interval is given through the following dimensionless quotient:

$$L = \frac{\frac{\partial^2 f}{\partial x^2}|_{\bar{x}} \cdot 2\sigma}{\frac{\partial f}{\partial x}|_{\bar{x}}} \tag{5.12}$$

This quotient compares the derivative at the interval center $\bar{x}$, with the derivative at the interval extremes $\bar{x} \pm 2\sigma$.

When $L \approx 0$ the function can be considered linear and the Gaussianity is preserved.

This means, in order to compare the inverse depth with the depth parameterization, this linearity index has to be calculated for the measurement function of each representation.

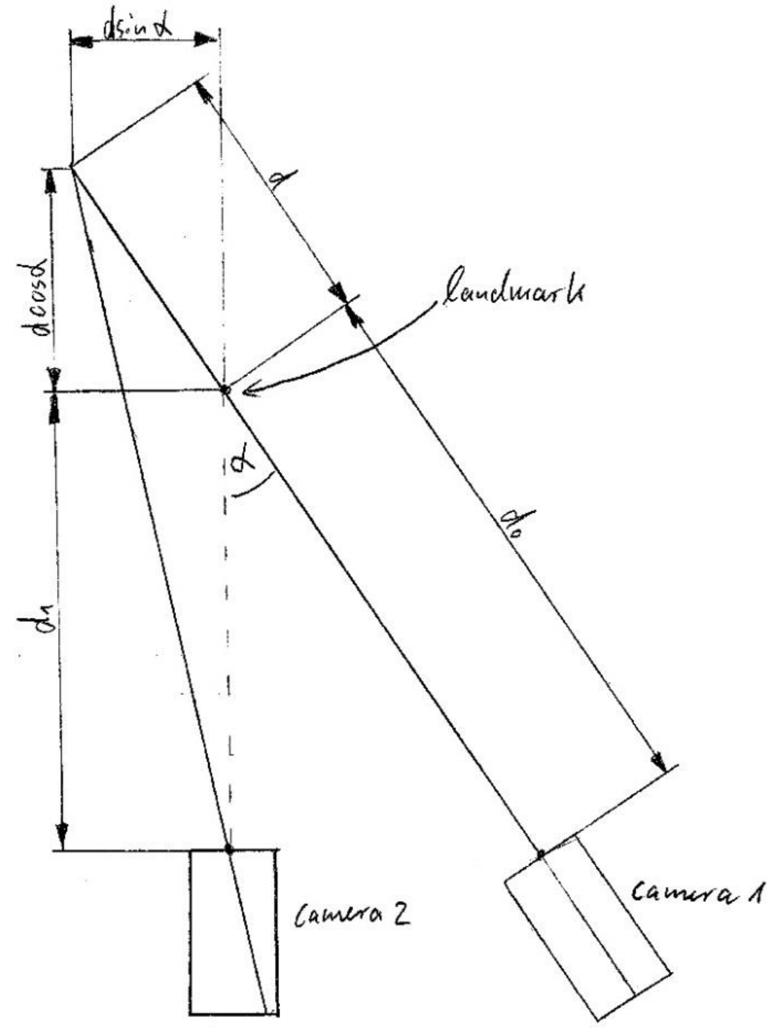


Figure 10: Simplified Camera Model for Depth Coding

Therefore, the authors of [Civera07] simplified the measurement model as depicted in figure 10. The landmark, located in distance d_0 is observed by camera 1 in the distance $D = d_0 + d$ with the Gaussian distributed location error $d = N(0; \sigma_d^2)$.

The simplified measurement equation for a camera with normalized focal length shows how the error d is propagated to the image taken with camera 2:

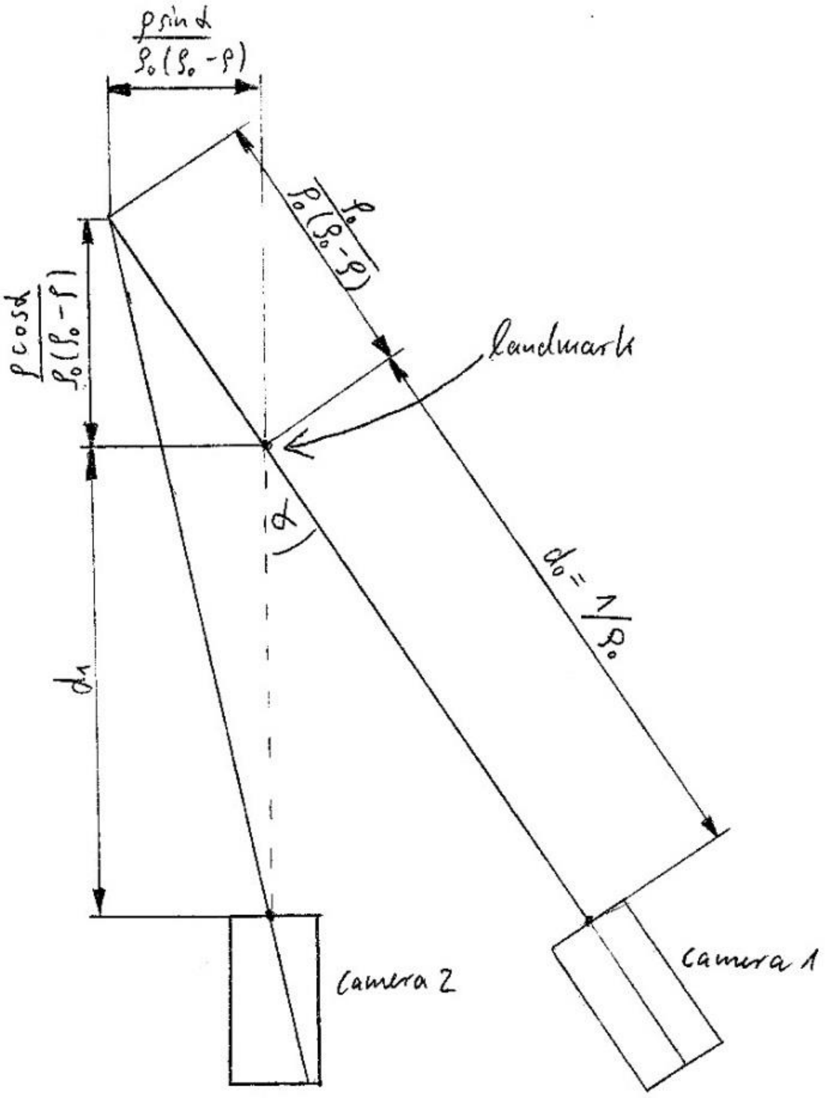
$$u = \frac{d \cdot sin\alpha}{d_1 + d \cdot cos\alpha} \tag{5.13}$$

When the linearity quotient (5.12) of u is calculated, the following results:

$$L_d = \frac{4\sigma_d}{d_1}|cos\alpha| \tag{5.14}$$

This means, L_d is only close to zero when the parallax angle α is very high (which for initialization is impossible) or when the initial uncertainty is very low in contrast to the distance d_1.

For the inverse depth coding, the simplified camera model is depicted in figure 11.



$\frac{\rho \sin \alpha}{\rho_0(\rho_0-\rho)}$
$\frac{\rho_0}{\rho_0(\rho_0-\rho)}$
$\frac{\rho \cos \alpha}{\rho_0(\rho_0-\rho)}$
landmark
α
$d_0 = 1/\rho_0$
d_1
Camera 2
Camera 1

Figure 11: Simplified Camera Model for Inverse Depth coding

The landmark in distance $\frac{1}{\rho_o}$ is observed by camera 1 in the distance $D = 1/(\rho_0 - \rho)$ with the Gaussian distributed location error $\rho = N(0; \sigma_\rho^2)$. With $d_0 = \frac{1}{\rho_0}$ follows: $d = D - d_0 = \frac{\rho}{\rho_0(\rho_0 - \rho)}$

The measurement equation for the inverse depth coding for camera 2 is then:

$$u = \frac{\rho \cdot sin\alpha}{\rho_0 \cdot d_1(\rho_o - \rho) + \rho \cdot cos\alpha} \tag{5.15}$$

This leads to the linearity quotient for the inverse depth:

$$L_\rho = \frac{4\sigma_\rho}{\rho_0}\left|1 - \frac{d_0}{d_1}cos\alpha\right| \tag{5.16}$$

So L_ρ will be automatically close to zero for low parallax, because then, $\frac{d_0}{d_1}cos\alpha$ will be close to 1 due to a small α and due to $d_0 \approx d_1$. This means, inverse depth is perfectly suitable for initialization, when low parallax is obtained. Therefor it's not important whether the initial uncertainty σ_ρ is high or how far or close a landmark is.

(5.14) and (5.16) deliver information about the linearity of the particular parameterization. The next step is to define a threshold which can be compared with the linearity quotient (5.14) and allows determining a point in time, from where the XYZ parameterization can be used.

In order to define this threshold, some information about the dependency of the real distribution from the linearity quotient has to be collected. [Civera07] applied a Kolmogorov-Smirnov test for a number of sets $(\alpha;\ d_1;\ \sigma_d)$ to verify the Gaussianity of u and at the same time they computed the linearity quotient for these sets. The result of their simulation was that a linearity quotient smaller 0.1 indicates an acceptable Gaussianity for u and therefore its save to switch from inverse depth to depth parameterization when:

$$L_d = \frac{4\sigma_d}{d_1}|cos\alpha| < 0.1 \tag{5.17}$$

5.2.7 Active Search

The idea of active search is to use all available information about the landmark and the current camera pose, to speed up the feature matching process. An intuition of the active feature search can be obtained by considering a projection of the probabilistic state vector into the image plane. When applied intelligently, the idea of active feature search leads to an enormous decrease in computation time. The dramatically decreased computation time originates not only from a smaller search region, but mainly from the fact that very simple appearance-based feature descriptors, like patches, together with correlation-based matching, are applicable together with active feature search. Active feature search or guided matching, how it's sometimes called, allows relinquishing processor burdening feature descriptors like SIFT or SURF. But, depending on the sort of implementation, it also considers scale and or rotation changes through "affine patch warping" or other advanced methods [Davison07] [Molton04]. In order to shrink the search region, the region of confidence of the landmark position is projected into the image plane. This means, instead of searching the whole image for a particular landmark, with active search, only an elliptical region, representing for example the 2σ confidence interval, has to be searched. Furthermore, because of the Gaussianity of this interval, by searching the inside of the elliptical region first, the matching time is further decreased. Another consideration which shouldn't be neglected is that by considering only a tiny search region through active search, the number of false feature matches sinks dramatically.

5.2.8 Feature Deletion

Computational power is a rare resource in real time applications, so it's crucial to have a mechanism to delete features and keep the map dimension reasonable. Davison's *monoSlam* system works with map sizes somewhere around 100 landmarks. It has to be distinguished between reliable features, which can be matched in many frames and always appear in places where they are predicted, and between features which haven't been matched repeatedly. One possibility to detect unreliable features is to compare the number of successful matches with the number of match attempts. Of course, this measure only makes sense, when a particular number of match attempts was made.

Besides the decreased map size, this deletion mechanism also provides some robustness in terms of a non-static environment. When for example, a landmark was initialized at a moving object, the corresponding feature will soon be deleted because it will never show up in the predicted search region. The result is that non-static objects in the environment remain automatically disregarded.

5.2.9 *MonoSLAM* Summary

The following sequence plan orientates on Davison's *monoSLAM* system. This is only one possibility to concretize a monocular SLAM system. Although the points below are quite coarse and fuzzy, they give an overview about the most important steps in monocular camera SLAM.

1. get new image from camera
2. for every feature stored in the map, search the region in which its appearance in the new image is most likely
3. match features through image patches, regarding scale and transformation
4. if necessary, search new features in the current image
5. for every new feature, define a semi-infinite ray from the camera optical center to the landmarks according to the pinhole camera model and initialize it with its inverse depth
6. decide for all features in the map to switch from inverse depth to XYZ coding
7. delete features which haven't been appeared in their predicted area for several measurements
8. refine camera position through an update of the state vector considering the changes in covariance matrices and state vector size in the EKF framework
9. Start from 1.

Note that these steps suite a running system and that a different start-up procedure is needed.

Chapter 6

Practical Part

The theoretical considerations of the previous chapters have shown how a visual monocular SLAM system works in detail. This part introduces Andrew Davison's "*SceneLib*" and describes how it can be used to perform the visual monocular SLAM procedure.

6.1 The Setup

The monocular SLAM system examined in this work is intended for controlling a two-wheeled balancing robot. The hardware is quiet simple. Besides the obvious components depicted in figure 12, the robot possesses a powerful ODROIDX2 mini PC with a Linux operating system which makes real-time image processing accessible. A standard low-cost USB digital camera provides the images. Although it would be possible to use the data from the IMU and the odometry sensors to support pose estimation, this will not be part of this thesis. For further information about the topic of visual- and IMU data fusion see for example [Bleser08]. An interesting approach of a lightweight SLAM alternative using optical flow and an IMU is introduced in [Bleser07].

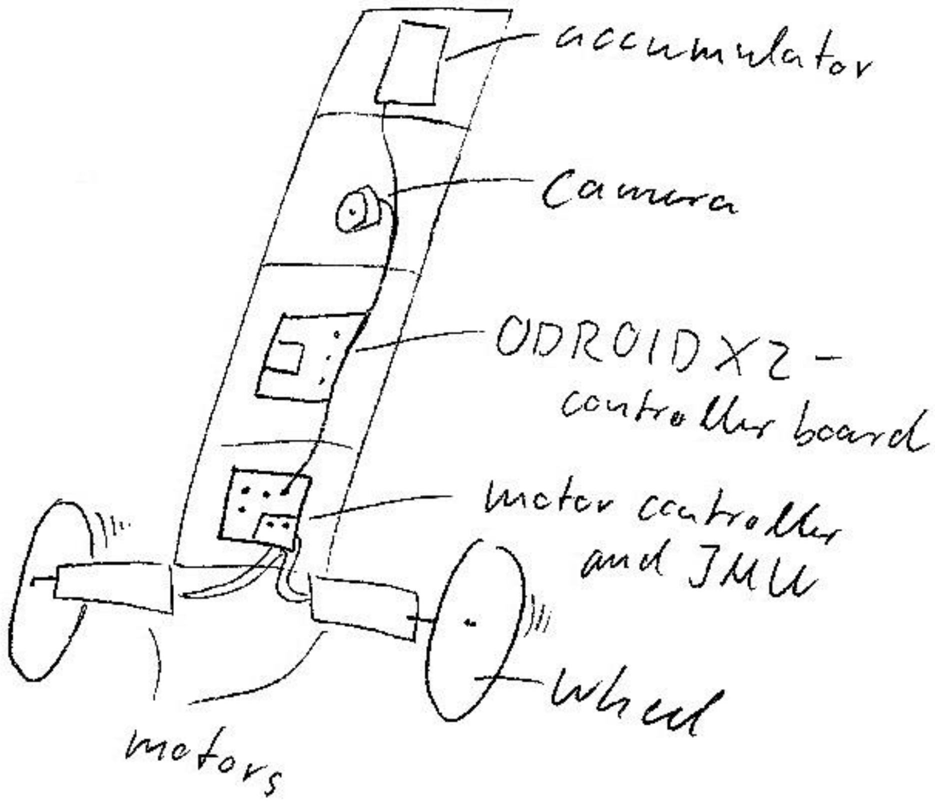


Figure 12: Robot setup

Since the *monoSLAM* system was designed to work with six degrees of freedom and doesn't depend on a model of the robot or any other motion constraints, a changed dynamic of the robot or the position of the mounted camera, shouldn't bother the pose estimation. This would provide the possibility to create a redundant system. Failure of one of the two available sensors (IMU or camera) will not endanger the stability of the robot. It also ensures that the robot stays highly adaptable to different tasks and designs. Note that visual monocular SLAM is not able to deliver absolute distances. That means the data provided by the algorithm will depend on an unknown scale factor. To determine this factor is important for a proper control. [Davison03] proposes to fix this issue by using an initialization template of known geometry. Another interesting approach, which uses the IMU data to obtain the missing scale factor, is presented in [Nützi07].

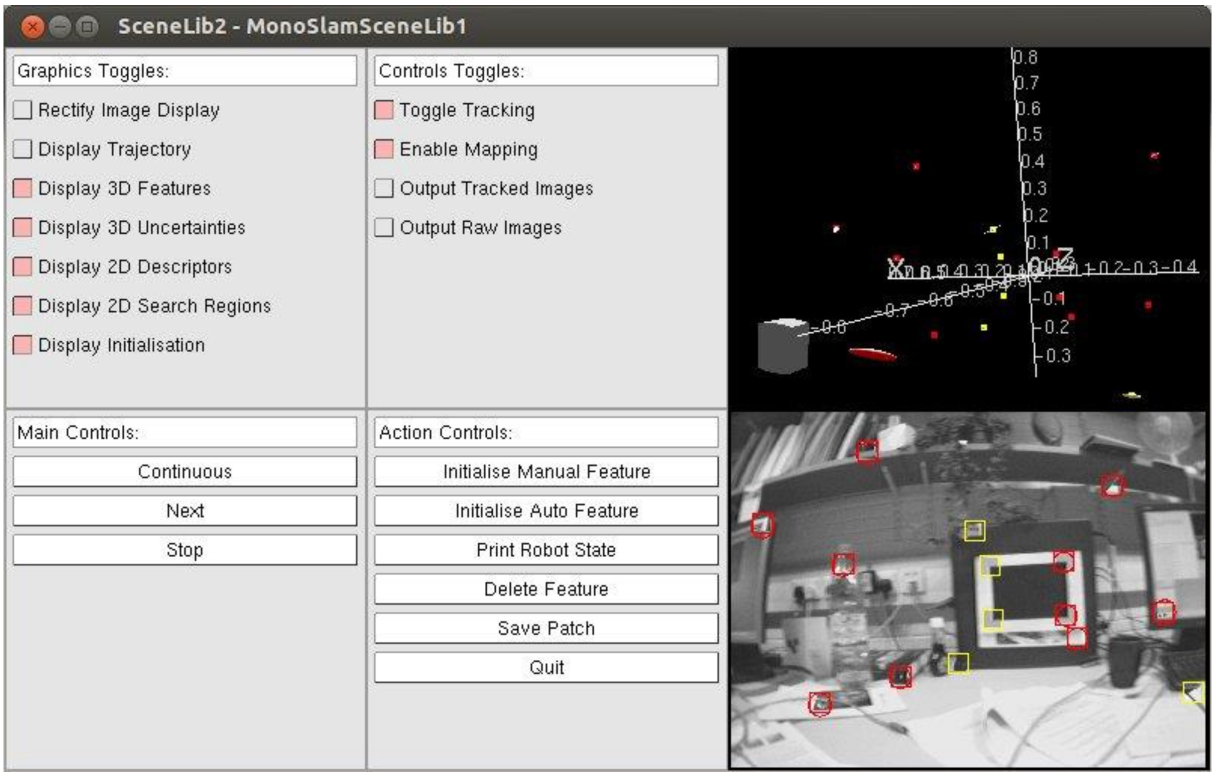
6.2 The *SceneLib* SLAM library

The first version of *SceneLib* was released in May 2006 and was called *SceneLib1.0* [SceneLib]. It was an open-source C++ library for SLAM in general and mainly designed and implemented by Andrew Davison with the intention to separate the core algorithms of probability propagation from the details of a particular robot application. Andrew Davison is a Professor of Robot Vision at the Imperial College in London. He also leads the Imperial College Robot Vision Research Group at the Department of Computing.

Together with the *SceneLib* library, Davison presented the so called *monoSLAMGlow* application. This program had an interactive GUI which used the GLOW toolkit and OpenGL and was able to take images in real-time from an IEEE1394 camera and perform monocular camera SLAM. Some more advanced features like "affine patch warping" [Davison07], [Molton04] or "relocalization" [Williams07], were not implemented in this version, nor are they provided in later open-source implementations of the *monoSLAM* system. So when the application demands a system, which can cope with significant rotation or strong camera shakes, the user needs to extend the open source version of *SceneLib*.

SceneLib2 [Kim13] is a reimplementation of Davison's system and the one used in this thesis. Hanme Kim, the author of *SceneLib2,* is currently a PhD student at Imperial College London. He created the new version to make the m*onoSLAM* system available for USB cameras instead of IEEE1394 cameras and to replace some older libraries (*VW34, GLOW, VNL*) with new ones (*Pangolin, Eigen3, Boost*).

The installation process of *SceneLib2* is very easy. After installing the various development related packages, and the four important libraries *OpenCV, Eugen3, Boost and Pangolin*, the *SceneLib2* can be built with the help of *CMake*. After the building process, much like in *SceneLib1,* a graphical user interface (GUI) is available which visualizes the results.


SceneLib2 - MonoSlamSceneLib1

Graphics Toggles:
- ☐ Rectify Image Display
- ☐ Display Trajectory
- 🟥 Display 3D Features
- 🟥 Display 3D Uncertainties
- 🟥 Display 2D Descriptors
- 🟥 Display 2D Search Regions
- 🟥 Display Initialisation

Controls Toggles:
- 🟥 Toggle Tracking
- 🟥 Enable Mapping
- ☐ Output Tracked Images
- ☐ Output Raw Images

Figure 13: *SceneLib2* GUI

Within the program, there are four main control areas. The "*Graphics Toggles*" allow displaying uncertainty regions or the camera trajectory. The *"Main Controls"* area is used to switch between "*continuous*" and frame to frame depiction of the camera images by clicking the "*Next"* button. By clicking the *"Stop"* button, the image sequence stops and new features can be initialized manually by clicking "*Initialize Manual Feature*" in the "*Action Controls"* area. The "*Action Controls*" section provides some other additional features to print the current robot state or save a particular image patch. These features are optional and can in some cases improve the localization or deliver additional information about the system. In order to start the mapping and localization process, it's only necessary to click the "*Continuous"* button in the *"Main Controls"* and then start tracking by clicking the "*Enable Mapping*" and "*Toggle Tracking*" buttons in the "*Controls Toggles*" section while pointing the camera at the initialization pattern. This initialization template, mentioned in the chapter above, is important to determine the scale factor in a monocular SLAM system and to be able to display absolute measures for the camera position. The template also provides reliable features within the first seconds of the process. It is beneficial for the tracking performance when the initialization frame displayed in the camera image on the right bottom side, fits the initialization template quite accurately. So the patches depicted in the image should ideally overlay the template corners.

Before using the program, the configuration parameters have to be adapted. This can be done by editing the file *SceneLib2.cfg* which is in the *data* folder of the *SceneLib2* directory.

The *SceneLib2.cfg* file provides all information for the initialization process. So besides the initial uncertainty regions, the initial values for the camera state vector and data concerning the geometry of the initialization template, the configuration file mainly contains information about the used camera.

6.3 Calibrating the Camera

The *SceneLib2* can either be used with an example image sequence or with a USB camera. In this thesis, a Logitech HD C270 webcam was used.

The first attempts with the *SceneLib2* were made with a non-calibrated camera. Apart from this, it showed that the used camera, as almost any low cost CMOS USB webcam, is not very suitable for monocular SLAM. First of all, the operations, which are applied by the hardware and software of the camera like jpeg-compression, interpolation or noise reduction, can be very contra productive. A smoothing operation, for example, can destroy some important image features. The biggest drawback of such a camera is, however, the distortion caused by the rolling shutter effect which means that not all parts of the image are recorded at exactly the same time.

So after the first attempts with the *SceneLib2*, it manifests that a non-calibrated camera in addition to a cheap camera consequently led to a system state vector that diverged within the first view seconds.

The next step was to find the intrinsic camera parameters as they were introduced in chapter 2. In order to do this the *Matlab* toolbox [Calib] was used. This toolbox provided by Jean-Yves Bouguet from California Institute of Technology is open source and identical to the C-code version which is part of the *OpenCV* library.

Before the calibration process can start, sample images have to be taken. The Logitech HD C270 is capable of delivering 640x480 pixels at 30 frames per second and images with up to three Megapixels. As the *SceneLib2* uses images with 320x240 pixels, the calibration process was also performed with images of the size 320x240 pixel. So, capturing images of the calibration target from different camera poses, in this case 10 images, and providing them to the toolbox were the next steps.

Afterwards, the outer grid corners had to be chosen manually and the program needed to know how many squares are displayed along the grid boundaries. Another parameter that had to be provided was the size of the squares. This geometric information was then used to display the assumed location of the corners in the calibration target. When these points weren't close to the real corners, an initial guess for the first distortion coefficient had to be entered. In most of the cases, the displayed corners were quite accurate and the corner extraction algorithm was able to find the corner location in an area around that assumed points. This process was then repeated for every image. Finally, the main calibration step which consists of an initialization and a nonlinear optimization was executed and the calibration results were obtained.

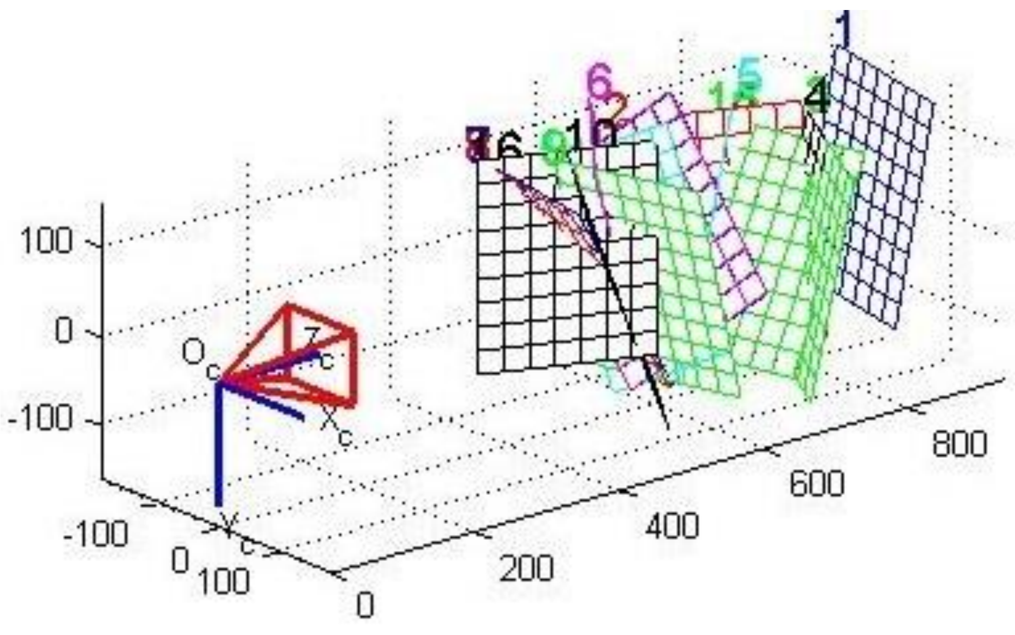


Figure 14: Extrinsic Parameters

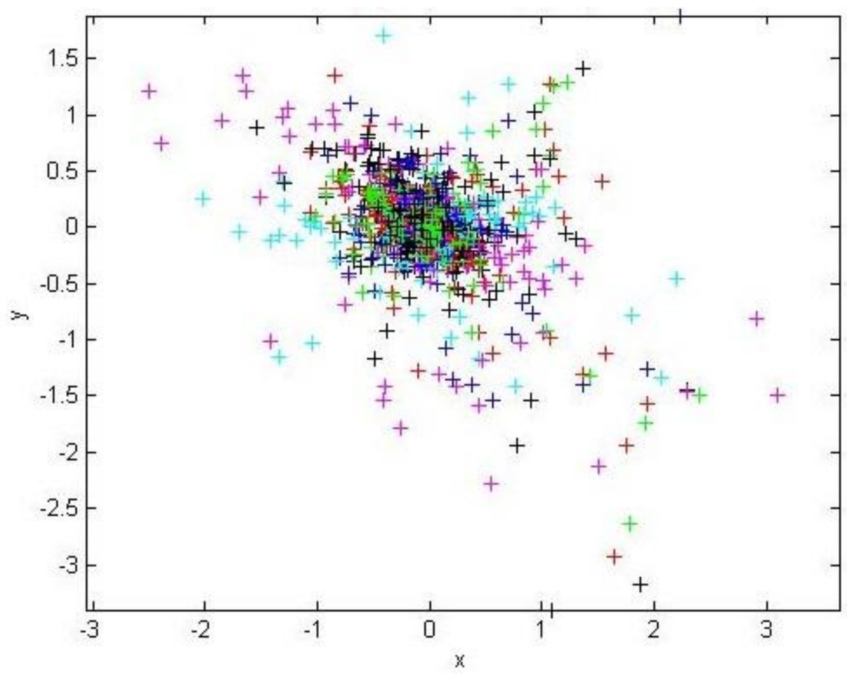


Figure 15: Reprojection Error

The reprojection error corresponds to the image distance between a measured point and a point which is projected according to the camera model.

After the first optimization process the following results were obtained.

Focal Length:	[407.52491 404.93005] ± [6.35612 6.94583]
Principal Point:	[171.77256 101.44741] ± [8.95702 10.01769]
Skew:	[0.00000] ± [0.00000]
Distortion:	[0.07079 -0.54587 -0.01676 0.00832 0.00000] ± [0.12484 0.92908 0.00946 0.00847 0.00000]
Pixel Error:	[0.41433 0.52834]

A further improvement of these parameters could then have been obtained, by using the projected corner points as initial location for corner extraction and passing the calibration process another time.

6.4 Results and Conclusion

With a calibrated camera, the *SceneLib2* was then tested again. The fact that in the open source version of the library, "affine patch warping" is not implemented makes it very vulnerable for large rotations. Additionally, the rolling shutter of the camera leads to distortion in dynamic situations. These two points were then probably the reason why even with a calibrated camera, the performance of the system was not significantly improved. By preventing camera shakes and moving very slowly with few rotations, it was finally possible to get a trajectory over more than 20 seconds that seemed to be quiet accurate.

However, a stabilization of a mobile two wheeled robot is definitely far beyond the possibilities of the unimproved, current system. On a mobile robot, rotation and shaky movements are daily business and therefore, the attempts performed with the hand held camera led finally to the conclusion, that it wouldn't make sense to use this system together with the balancing robot without supplementing the code with approaches presented in [Davison07] or [Williams07] and using a global shutter camera.

Bibliography

[Ayache91] N. Ayache, "Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception", MIT Press, 1991.

[Baraldi89] P. Baraldi, E. De Micheli and S. Uras, "Motion and Depth from Optical Flow", Alvey Vision Conference, 1989.

[Bay06] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features", 9th European Conference on Computer Vision, 2006.

[Beardsley95] P. A. Beardsley, I. D. Reid, A. Zisserman, and D. W. Murray, "Active Visual Navigation Using Non-Metric Structure", Proc. Fifth Int'l Conf. Computer Vision, page 58-65, 1995.

[Bleser07] G. Bleser and G. Hendeby, "Using Optical Flow as Lightweight SLAM Alternative", German Research Center for Artificial Intelligence, 2007.

[Bleser08] G. Bleser and D. Stricker, "Advanced tracking through efficient image processing and visual-inertial sensor fusion", IEEE Virtual Reality, page 137-144, 2008.

[Calib] J. Y. Bouguet, "Camera calibration Toolbox for Matlab", available at http://www.vision.caltech.edu/bouguetj/calib_doc/index.html, Dec. 2013.

[Civera07] J. Civera, A. Davison and J. Montiel, "Inverse Depth to Depth Conversion for Monocular SLAM", Proceedings of IEEE International Conference on Robotics and Automation, 2007.

[Civera08] J. Civera, A. Davison and J. Montiel, "Inverse Depth Parametrization for Monocular SLAM", IEEE Transactions on Robotics, page 932–945, 2008.

[Chekhlov06] D. Chekhlov, M. Pupilli, W. W. Mayol-Cuevas, and A. Calway, "Real-time and robust monocular slam using predictive multi-resolution descriptors", International Symposium on Visual Computing Number 2, page 276–285, 2006.

[CVAlgoAndApplic] R. Szeliski, "Computer Vision: Algorithms and Applications", Springer, 2010.

[Davison98] A. J. Davison, "Mobile Robot Navigation Using Active Vision", PhD dissertation, Univ. of Oxford, 1998.

[Davison03] A. J. Davison, "Real-Time Simultaneous Localisation and Mapping with a Single Camera", Proc. Ninth Int'l Conf. Computer Vision, 2003.

[Davison07] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, “MonoSLAM: Real-Time Single CameraSLAM”, IEEE Transactions on pattern analysis and machine intelligence, page 1052–1067, 2007.

[SceneLib] A. J. Davison, “SceneLib”, http://www.doc.ic.ac.uk/~ajd/Scene/index.html, Dec. 2013.

[Durrant06] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: Part 1”, Robotics & Automation Magazine, page 99–110, 2006.

[Kim13] H. Kim, „SceneLib2“, http://hanmekim.blogspot.co.uk/2012/10/scenelib2-monoslam-open-source-library.html, Dec. 2013.

[Harris87] C.G. Harris and J.M. Pike, “3D Positional Integration from Image Sequences”, Proc. Third Alvey Vision Conf., page 233-236, 1987.

[Harris88] C. Harris and M. Stephens, “A Combined Corner and Edge Detection”, Proceedings of The Fourth Alvey Vision Conference, pages 147–151, 1988.

[Kalman60] R. Kalman, “A new approach to linear filtering and prediction problems”, Transactions of the ASME-Journal of Basic Engineering, page 35–45, 1960.

[Karlsson05] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich, “The vSLAM algorithm for robust localization and mapping,” IEEE International Conf. on Robotics and Automation, 2005.

[Klein07] G. Klein and D. Murray, „Parallel Tracking and Mapping for Small AR Workspaces“, International Symposium on Mixed and Augmented Reality, 2007.

[Lowe99] D. G. Lowe, "Object recognition from local scale-invariant features", International Conference on Computer Vision, page 1150-1157, 1999.

[Molton04] N. Molton, A. Davison and I. Reid, “Locally Planar Patch Features for Real-Time Structure from Motion” Proceedings of the British Machine Vision Conference, 2004

[Montiel06] J. Montiel, J. Civera, and A. Davison, “Unified inverse depth parameterization for monocular SLAM”, Proceedings of Robotics: Science and Systems, Philadelphia, 2006.

[Moutarlier89] P. Moutarlier and R. Chatila, “Stochastic Multisensory Data Fusion for Mobile Robot Location and Environement Modelling”, Proc. Int’l Symp. Robotics Research, 1989.

[MultViewGeo] R. Hartley and A. Zisserman, “Multiple View Geometry in Computer Vision”, Cambridge University Press, 2000.

[Newcombe10] R. Newcombe and A. Davison, “Live dense reconstruction with a single moving camera”, Proc. Of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.

[Newcombe11] R. Newcombe, S. Lovegrove and A. Davison, "DTAM: Dense Tracking and Mapping in Real-Time" IEEE International Conference on Computer Vision, 2011.

[Nützi07] G. Nützi and S. Weiss, "Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM", ETH Autonomous Systems Laboratory Zürich, 2007.

[Pupilli05] M. Pupilli and A. Calway, "Real-time camera tracking using a particle filter", Proc. British Machine Vision Conference , page 519–528, 2005.

[Shi94] J. Shi and C. Tomasi, "Good Features to Track", IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94), page 593 – 600, 1994.

[Smith87] R. Smith, M. Self, and P. Cheeseman, "A Stochastic Map for Uncertain Spatial Relationships", Proc. Fourth Int'l Symp. Robotics Research, 1987.

[Sola07] J. Solà, "Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot: a Geometric and Probabilistic Approach", PhD thesis, LAAS-CNRS, 2007.

[Stuehmer10] J. Stuehmer, S. Gumhold and D. Cremers, "Real-time dense geometry from a handheld camera", Proceedings of the DAGM Symposium on Pattern Recognition, 2010.

[Tsai87] R. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses'', IEEE Journal of Robotics and Automation, Vol. RA–3, No. 4, page 323–344, 1987.

[Williams07] B. Williams, G. Klein and I. Reid, "Real-time SLAM Relocalisation", Proc. International Conference on Computer Vision, 2007.

List of Figures

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.

Stuttgart, December 2013

(Patrick Suhm)