Masterarbeit am Institut für Informatik der Freien Universität Berlin,

Dahlem Center for Machine Learning and Robotics

Fisheye Camera System Calibration for Automotive Applications

Christian Kühling
Matrikelnummer: 4481432
kuehling@zedat.fu-berlin.de

Erstgutachter: Prof. Dr. Daniel Göhring
Zweitgutachter: Prof. Dr. Raúl Rojas
Betreuer: Fritz Ulbrich

Berlin, 05.05.2017

Abstract

In this thesis, the imagery of the fisheye camera system mounted to the autonomous car *MadeInGermany* of the Freie Universität Berlin is processed and calibrated. Hereby, distortions caused by fisheye lenses are automatically corrected and a surround view of the vehicle is created.
Over the next decade, autonomous cars are expected to radically change mobility as we know it. While intelligent software systems made astonishing improvements over the past years, the eventual quality of autonomous cars depends on their sensors capturing the environment.
One of the most important sensors are cameras for visual input. Hence, it does not surprise that current autonomous prototypes often have multiple cameras, for example to prevent having blind spots. For this specific reason, fisheye lenses with a large field of views are often used.
To utilize recordings of these camera systems by computer vision algorithms, a camera calibration is required. It consists of the intrinsic calibration, rectifying possible distortions and extrinsic calibration, determining position and pose of the cameras.

Statement of Academic Integrity

Hereby, I declare that I have composed the presented paper independently on my own and without any other resources than the ones indicated. All thoughts taken directly or indirectly from external sources are properly denoted as such. This paper has neither been previously submitted to another authority nor has it been published yet.

05.05.2017

Christian Kühling

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

There has been a remarkable growth in the use of cameras on autonomous and human-driven vehicles. In 2014 analysis of the market reveals possible growth at a rate of over 50% Compound Annual Growth Rate until 2018 for Advanced Driver Assistance Systems [Gro]. Cameras offer a rich source of visual information, which can be processed in real-time thanks to recent advances in computing hardware. From the automotive perspective, multiple cameras are recommended for using them for driver assistance applications such as lane detection, traffic light detection, the recognition of other traffic participants or simply the termination of blind spots, where there is no or little view.
Image processing applications utilizing multiple cameras for a vehicle require an accurate calibration. Camera calibration is divided into two parts: intrinsic and extrinsic calibration.
An accurate intrinsic camera calibration consists of an optimal set of parameters for a camera projection model that relates 2D image points to 3D scene points. This is especially challenging for fisheye lenses, whose advantage is a large field of view at the cost of strong distortions. Latter should be rectified by using the parameters resulting from the intrinsic calibration.
An accurate extrinsic calibration corresponds to accurate camera positions and their rotations with respect to a reference frame on the vehicle. This is needed whenever the size of an object has to be measured, or the location or position of an object has to be determined.
Overall camera calibration is a crucial part of computer vision being the requirement for most available computer vision algorithms.

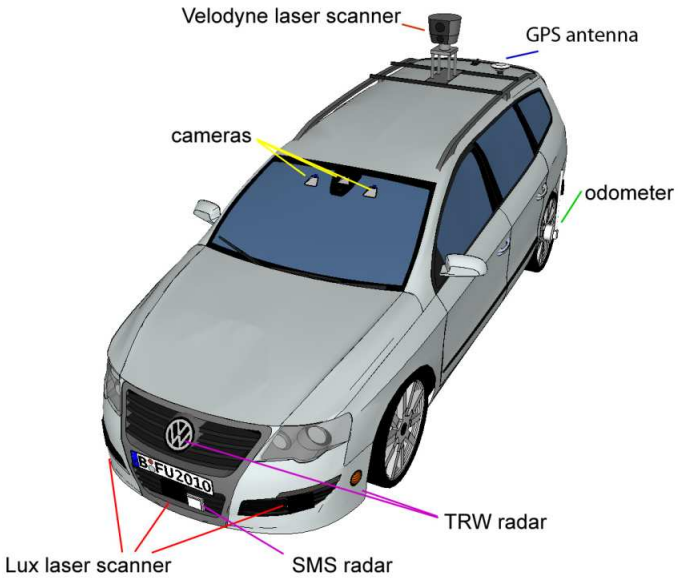


Figure 1.1: Autonomous Car *MIG* of Freie Universität Berlin [Aut]
Not displayed: the fisheye cameras used in this thesis

1.2 AutoNOMOS project

In the year 2006 Prof. Dr. Raúl Rojas and his students of the working group Artificial Intelligence at the Freie Universität Berlin launched the ancestor of the AutoNOMOS project [Neu16]. A Dodge Grand Caravan was converted to an autonomous car by installing several sensors and computer hardware. This car was called *Spirit of Berlin*. It participated in the DARPA(Defence Advanced Research Project Agency) Grand Urban Challenge 2007, a competition for autonomous vehicle.
Since 2007 tests were ran, while driving autonomously at the area of the former airport Berlin-Tempelhof. This led to public funds granted by the federal ministry of education and research, which resulted in the AutoNOMOS project [Aut]. Two new test vehicles were designed. The first one is called *e-Instein*, an electrically powered vehicle whose basis is a Mitsubishi i-MiEV. The other car is a Volkswagen Passat Variant 3C equipped with Drive-by-Wire, Steer-by-Wire technology, overall more sensors and computer hardware than *e-Instein*. It is called *MadeInGermany* which will be shortened to *MIG* going forward. Due to exceptional permissions, testing functionalities in real traffic situations in Berlin, Germany was possible. This resulted in various publications of miscellaneous topics like vehicle detection through stereo vision [Neu16], Swarm Behaviour for Path Planning [Rot14] or Radar/Lidar Sensor Fusion for Car-Following on Highways[GWSG11].

In figure [1.1] the following sensors are displayed:

- Hella Aglia INKA cameras: Two front-orientated stereo-cameras, placed next to the rear mirror
- Lux Laser Scanner: For detecting obstacles, placed in the front and rear bumper bar
- Smart Microwave Sensors GmH Radar(SMS): At the front bumper to detect the speed of preceding vehicles
- Odometer(Applanix POS/LV System): For calculating the travelled distance and the wheel rotations. It is placed at the left rear wheel.
- TRW/ Hella radar system: This radar system is placed in the front and rear area. It is installed for measuring the distance between the *MIG* and surrounding objects.
- Velodyne HDL-64E: To detect obstacles all around the *MIG* this LIDAR-system is placed at the roof of the vehicle.

Not all installed sensors are displayed in figure [1.1], because from time to time the setup changes. New sensors get installed, tested, deactivated or even deconstructed. *MIG* is also equipped with four fisheye cameras, which were not used for researching until now. In this thesis, this camera system is the main component wherefore this very thesis only refers to the *MIG*, not discussing the *e-Instein*. A detailed description of the fisheye camera system can be found at section [2.1].

1.3 Goal

The goal for this master's thesis is to calibrate a fisheye camera system integrated into the autonomous car *MIG* of Freie Universität Berlin for further usage.
For this purpose, a camera driver is required in addition to an easy to use calibration script for gaining the intrinsic and extrinsic camera parameters. The verification of the resulting intrinsic parameters will be done by the rectifying the current fisheye distortion. Furthermore, a combination of the intrinsic and extrinsic calibrations represents the basis of a surround view. The aim of this work is also to provide the captured images and calibrations within the ROS(Robot Operation System) [2.5.1] framework, for further utilization.

1.4 Thesis structure

This thesis is structured as follows: Subsequent to the current introduction chapter, essentials in terms of the available setup, camera models, coordinate systems, camera calibration and the used software will be illustrated. Afterwards chapter three will cover the implementation made for this thesis in order to calibrate the existing fisheye camera system and evaluate the resulting intrinsic and extrinsic parameters. Eventually, in the last chapter the achieved results and possible optimizations will be discussed.

1.5 Related Work

In the master thesis of Söderroos [Söd15] a surround view for heavy vehicles is presented. Their system consists of of two fisheye cameras. As part of the thesis, two different fisheye camera calibration toolboxes are compared. In the end, the images of the two fisheye cameras were stitched together and as the camera's field of view is overlapping a blending function is needed. In terms of the comparison of the intrinsic calibration toolboxes, this work is a good orientation to start with.
For the extrinsic calibration the master thesis of Middelplaats [Mid14] gives good insights. In this work an industrial flexible robot arm is equipped with a camera. To prevent collisions of the robot arm with obstacles when moving the arm, the camera is used to create a collision map. Hence an extrinsic calibration is needed, while several calibration options are investigated.
Both, extrinsic and intrinsic calibration are part of the master thesis of Hofmann [Hof13]. Mainly this thesis is about sensor fusion of radar and camera for object tracking. Here side cameras are used to enhance and refine RADAR tracking results. The theoretical fundamentals of several camera models and camera calibration are compared and evaluated. Therefore this master thesis is very inspiring.

Chapter 2

Fundamentals

This chapter provides the basic knowledge for the further progression of this thesis. Starting with the hardware setup [2.1] covering detailed information about the camera system installed in the *MIG*. Afterwards, the theoretical fundamentals [2.2] like coordinate systems, the basics of camera models and finally the essentials of intrinsic [2.3] and extrinsic camera calibration [2.4] are presented. An overview of the used libraries and software [2.5] completes this chapter.

2.1 Hardware setup

There is a total of four cameras installed in the *MIG*. The first camera is located in the front central placed between the numberplate and the car icon. Mirrored at the other end but just a little bit higher is the rear camera. The left and right cameras are positioned under the side mirrors of the vehicle. All cameras are displayed in detail in Figure [2.1] respectively their positions in Figure [2.2].
The BroadR-Reach SatCAM [Tec] has the following basic features and is displayed in [2.4a]:

- Automotive Grade Surround View Camera used in series cars as front rear or mirror camera.
- Up to 1280x800 pixels at 30 frames per second
- Freescale PowerPC Technology
- Automotive BroadR-Reach Ethernet data transfer
- Integrated Webserver for easy configuration.
- Delivers compressed JPEG-pictures

(a) Front camera

(b) Rear camera

(c) Left camera

(d) Right camera

Figure 2.1: The installed camera system

(a) Front view of the *MIG*

(b) Back view of the *MIG*

Figure 2.2: Camera positions

Figure [2.4b] exhibits the car trunk of the *MIG*. The blue box right front is the technica engineering media gateway [Med], the switch of the BroadR-Reach cameras. In table [2.3] the BroadR-Reach SatCam's technical data is shown.
Due to restricted information its not definitely clear, but considering all available information the camera lenses are classified to belong to the OVT OV10645 [Omn] camera lenses family. These offer a field of view of 190°.
Based on the technical data of the camera sensors a rough model is shown in figure [2.5] to display the camera positions and their theoretical sensor coverage in one overview. Cameras positions are marked with an orange dot. The blue area represents the area only visible to the front and rear camera, the yellow area showing the area only visible to the left and right camera and at last the overlapping Fields of View are displayed in green.

Feature	
Power requirement	8 to 14 Volt DC (nominal 12 Volt DC)
Size	25 x 28 x 55 mm
Weight	0,1 kg
Operation Temperature	-40 to +80 degree Celsius

Figure 2.3: Technical data of BroadR-Reach SatCAM

(a) BroadR-Reach Camera [Tec]

(b) Car trunk of the *MIG*

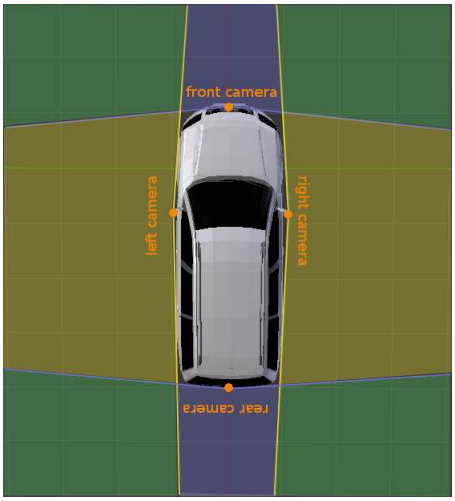


Figure 2.5: Overview of the camera positions and their theoretical FOV Orange Dots: camera positions. Blue area: front/rear only FOV, yellow area: left/right only FOV, green area: overlapping FOV.

2.2 Theoretical fundamentals

In subsection [2.2.2] the coordinate systems and mathematical notations [2.2.1], that are used in this thesis, are presented. In this sections end [2.2.3] an introduction to camera systems is shown. Parts of this section were structured based on the work presented in [BK13] and [Hof13].

2.2.1 Mathematical notations

Matrices, points and vectors are frequently used in this thesis, alongside angles, which are particular Greek letters annotated kind of scalar. In table [2.6] the respective notations can be found.

Type	Notation	Explanation
Angle	α	Greek letter, normal font
Coordinate transformation	$\mathbf{T}_{A\to B}$	Bold, big letters transforms from System A to B
Matrix	$\mathbf{M}$	Bold, big letter
Point	P	Big letter, normal font
Scalar	s	Small letter, normal font
Vector	$\vec{v}$	Small letter with an arrow above

Figure 2.6: Mathematical Notations

Coordinate representations of n-dimensional space, representing lines in n + 1-dimensional space by adding one as a constant to the vector can be defined as homogeneous coordinates.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \to \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.1}$$

This compact combination of rotations and translations are annotated as $\mathbf{T}_{A\to B}$. For arbitrary transformations from n-dimensional coordinate system A to d-dimensional coordinate system, these annotations can be used. Commonly they define a transformation matrix [2.2] consisting of a rotation matrix $\mathbf{R}$ and a translation vector $\vec{t}$ that is used to transform homogeneous coordinates. The rotation matrix and translation vector combined together are the extrinsic parameters. While the translation vector describes the position of a camera, the rotation matrix represents the rotations of the camera regarding the yaw, pitch and roll angles. A more detailed description is expressed later on in this thesis.

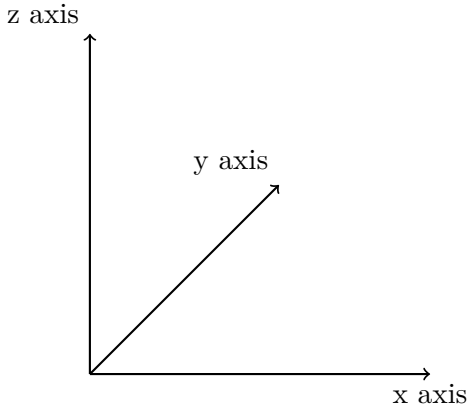
$$\mathbf{T}_{A\to B} = \begin{pmatrix} & \mathbf{R} & & \vec{t} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

2.2.2 Coordinate systems

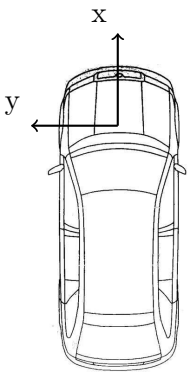
At some point when working with multiple cameras involving multiple coordinate systems, they have to be transformed to each other. To do so, mathematical notations need to be introduced. There are several coordinate systems used in this thesis. Starting with a world coordinate systems, which is used as a general reference. A local coordinate system which is carried along with the motion of the vehicle is defined as the ego vehicle coordinate system. There are two coordinate systems used by the pinhole camera model [2.2.3]: the 3-dimensional camera coordinate frame and the image frame referencing to pixel coordinates. Following these coordinate systems are explained in detail.

World coordinate system

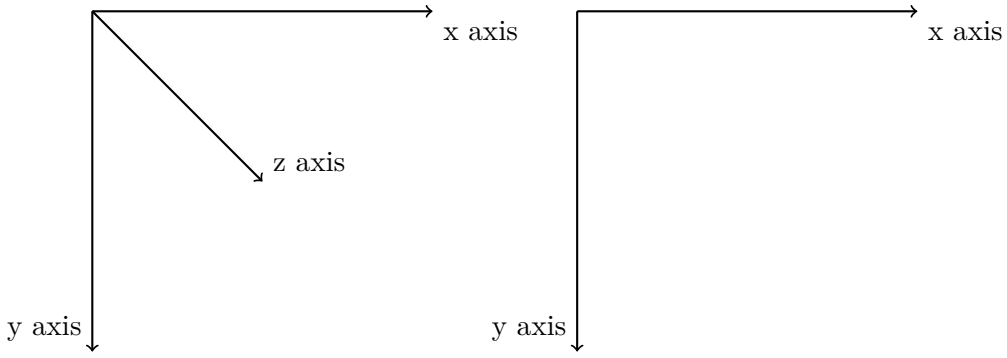
The world coordinate system as shown as in figure [2.7a] defines the x-y-plane as the ground plane, while the z-axis is pointing upwards. So the coordinate system is right-handed.



(a) World coordinate system



(b) Ego Vehicle Coordinate System, model taken from [Ego]



(a) Camera coordinate system (b) Image coordinate system

Ego vehicle coordinate system

Alongside with the motion of the car, the so-called ego coordinate system reference frame is carried along. Like the world coordinate system, it is right handed. Its origin is located in the middle of the front axle. The x-axis points to the driving direction, the y-axis runs parallel to the front axle and the z-axis is pointing upwards. It is displayed in figure [2.7b].

Camera coordinate system

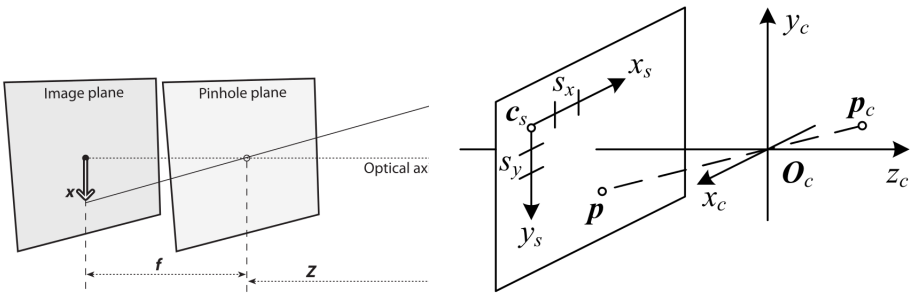
As shown in [2.8a], the camera coordinate systems viewpoint with the z-axis pointing away from the camera in the direction of the view is the origin of the camera system.

Image coordinate system

Images captured from camera sensors making use of the image coordinate frame. Its center is at the top left of the position of the image and references to pixel coordinates.

2.2.3 Pinhole camera model

A simple model to describe the illustration properties of cameras is the pinhole camera model, which is explained in more detail in the following works: [MK04], [BK13] and [Sze10]. In this model, light is envisioned as entering from the scene or a distant object, but only a single ray enters from a particular point, which is "projected" onto an imaging surface. The image of this *image plane* is as a result always in focus. A single parameter of the camera, the so-called *focus length* gives the size of the image relative to the distant object. The distance from the pinhole aperture to the screen in this idealized pinhole camera is precisely the focus length. Also, this figure [2.9a]



(a) Pinhole camera Model [BK13] (b) Pinhole camera Model [Sze10]

shows the distance from the camera to the object Z, the length of the object X, the cameras focus length f and the object's image on the imaging plane x. This is summarized in equation [2.3]:

$$-x = f \cdot \frac{X}{Z} \tag{2.3}$$

Another visualization by Szeliski [Sze10] is displayed in figure [2.9b] and illustrates the basic structure of the pinhole camera model. It shows a 3D-Point P_c, which is assumed to be in the camera coordinate system that has its origin at O_c. This is the optical center with the axes x_c, y_c and z_c. The 3D-Point P_c is transformed onto the image sensor plane, usually by a projective transformation like in equations [2.4] and [2.5]. The 3D origin C_s and the scaling factors s_x and s_y determines the projected point P on the sensor plane. The last variable z_c defines the so-called optical axis.

$$x' = \frac{X}{Z} \tag{2.4}$$

$$y' = \frac{Y}{Z} \tag{2.5}$$

In order to transform the 3D-Points from its own world coordinate system [2.7a] to the camera coordinate system [2.8a] equation [2.2] is used. Examining the full relationship between a 3D-Point P and its image projection x under the usage of homogeneous coordinates this results in the following formula [2.6] [Sze10]. The Matrix **K** is called the *intrinsic matrix*, containing the *intrinsic camera parameters*. They are needed to acquire the pixel coordinates of the projection point (u,v). s is a scalar scaling factor. Known as the 3×4 camera matrix is the combination of $\boldsymbol{K}[\boldsymbol{R}|\vec{t}]$.
The *intrinsic camera Matrix* **K** combines the *focal lengths* f_x and f_y together with the skew γ between the sensor axes and the principal point (C_x, C_y),

which determines the intersection of the camera Z-axis with the viewing plane. The skew γ is usually negligible in real-world cameras, which is why it will be assumed to be zero. Further the *aspect ratio* between the x-axis and y-axis can be explicit by adapting the definition of the second focal length $f_y = \alpha\ f_x$. It is worth mentioning, that this ratio is not directly related to the aspect ratio between an image produced by the camera ($\frac{width_{px}}{height_{px}}$), but rather defines the pixel aspect ratio. Usually it is possible to simplify $f_x = f_y = f$ because of the assumption of square pixels, which means that in most cases a single focal length is suitable. Another presumption is, that the principal point usually lies near the image center. This way it is possible to get the principle point P by $P_x = \frac{width_{px}}{2}$ and $P_y = \frac{height_{px}}{2}$.
The intrinsic camera matrix does only depend on internal camera properties, not on scene properties. So as long as parameters like focal length, which only changes when the lens is modified, and the output image size stays the same, the intrinsic camera matrix can stay the same.

$$sx = \boldsymbol{K}[\boldsymbol{R}|\vec{t}]P = s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & \gamma & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \vec{t}_1 \\ r_{21} & r_{22} & r_{23} & \vec{t}_2 \\ r_{31} & r_{32} & r_{33} & \vec{t}_3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.6}$$

Besides the *intrinsic camera Matrix* $\mathbf{K}$ the formula [2.6] contains the already in [2.2] mentioned rotation matrix $\mathbf{R}$ and the translation vector $\vec{t}$. Both together are the *extrinsic parameters* and they determine the orientation respectively the position of the camera in the world coordinate system [2.7a]. The *extrinsic parameters* consist of six degrees of freedom, three coordinates namely X,Y,Z and three angles α, β, γ. From the aviation the names roll, pitch and yaw were established for the angles [2.10].
Normally rotations are done in the following order:

- Rotation around the x-axis: roll

$$\boldsymbol{R_x} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{pmatrix} \tag{2.7}$$

- Rotation around the y-axis: pitch

$$\boldsymbol{R_y} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(beta) \end{pmatrix} \tag{2.8}$$

- Rotation around the z-axis: yaw

$$\boldsymbol{R_z} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

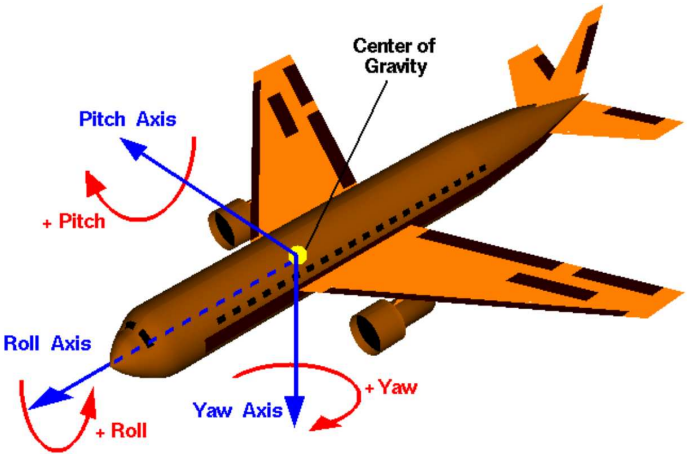


Figure 2.10: Roll, pitch and yaw [Rol]

- which results in:

$$R = R_x R_y R_z \tag{2.10}$$

2.3 Intrinsic camera calibration

The advantage of using a wide-angle or fisheye lenses is the larger field of view. Though there are the disadvantages of distortion, which makes straight lines appear as curved lines instead, and the impact on the resolution of the image. In figure [2.11] two common types of radial distortion are displayed: the barrel distortion and the pincushion distortion. These types are the most occurring distortions besides slight tangential distortion.
Wide-angle cameras have a field of view(FOV) of 100° - 130° while a fisheye camera has a larger FOV of about 180°. Since the pinhole camera model [2.9a] can not handle a zero value on the z-coordinate, the axis parallel to the optical axis, a lens using a classic wide angle model can not cover a 180° FOV. The pinhole camera model will project 3D-points covered from a 180° FOV at infinity and it can therefore not be used as a projection model for fisheye lenses.

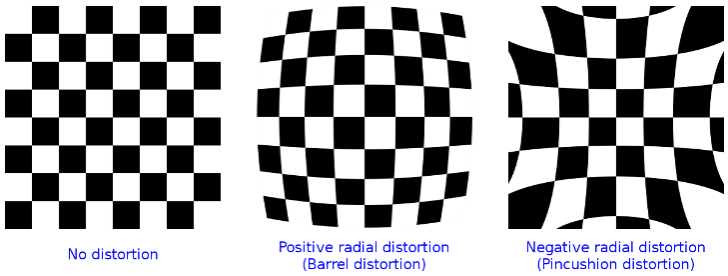


Figure 2.11: Two types of distortion [Dis]

Hence new camera models that are able to handle this distortion are required. These models use parameters to describe distortions. The process to estimate the set of intrisic parameters in a camera model is called intrinsic camera calibration. Following, this section discusses three different options to gain the intrinsic camera parameters and is inspired by these works: [Söd15], [Hof13] and [Sch].

2.3.1 Mei's calibration toolbox

The first calibration toolbox was developed by Mei [Meia] for providing the camera extrinsic, intrinsic and distortion parameters in MATLAB [2.5.6]. It can be used to calibrate and evaluate the intrinsic camera parameters for multiple camera types such as parabolic, catadioptric and dioptric. This toolbox is based on a specific projection model [Meib], an extension of [BA01] also made by Mei. Here the world points are projected onto the unit sphere before they are projected onto the normalized image plane. This allows the center of projection to be shifted onto the image plane. The center of the images are used as an estimate of the principal points with a diotric camera

model and a planar calibration grid is used in the calibration procedure. Figure [2.12] illustrates the projection model for projection 3D-points and is described by the following steps:

1. Project a world point Y in the camera coordinate frame onto the unit sphere
$$(x_n, y_n, z_n) = \frac{Y}{\|Y\|} \tag{2.11}$$

2. Change the points to a new reference frame centred in $C_p = (0, 0, \xi)$
$$(x_S, y_S, z_S) = (x_n, y_n, z_n + \xi) \tag{2.12}$$
$$\xi = \frac{df}{\sqrt{df^2 + 4p^2}} \tag{2.13}$$
where df is the distance between focal points and $4p$ is the lactus rectum, the chord through a focus parallel to the conic section directrix.

3. Project the points onto the normalised image plane. The shift of the center of projection in the previous step ensures, that Z_S is never zero and this perspective projection runs smoothly.
$$\boldsymbol{M_u} = (\frac{x_S}{z_S}, \frac{y_S}{z_S}, 1) \tag{2.14}$$

4. Add the radial distortion to the point, where k_i are the distortion parameters, and D is the distortion function [2.17].
$$\boldsymbol{M_d} = \boldsymbol{M_u} + D(\boldsymbol{M_u}, k_i) \tag{2.15}$$

5. Project the points onto the image using the camera matrix $\mathbf{K}$
$$\boldsymbol{P} = \begin{pmatrix} f & s & x_c \\ 0 & f_\alpha & y_c \\ 0 & 0 & 1 \end{pmatrix} \boldsymbol{m_d} \tag{2.16}$$
Equation [2.17] shows the distortion function D with distortion parameters $k_i = 1, .., 5$ and $p = \sqrt{x^2 + y^2}$
$$D(p) = 1 + k_1 p^2 + k_2 p^4 + k_3 p^6 + k_4 p^8 + k_5 p^{10} \tag{2.17}$$

It is required to collect images of the chessboard calibration pattern shown in different positions and angles to the camera before using the toolbox. The user has to select at least three non-radial points, meaning not placed on radii from the image center, which belong to a line in an image, and the focal length is estimated from them. Also, the user has to select four grid corner points in each image which can be used for initialization of the

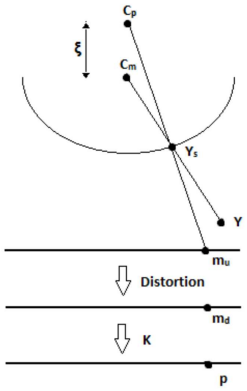


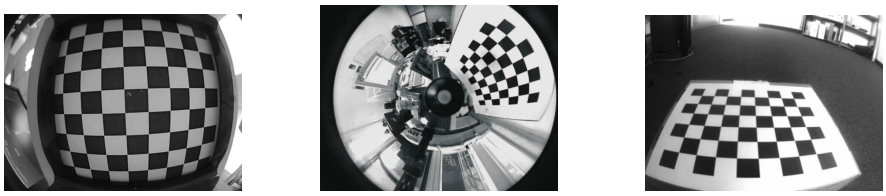
Figure 2.12: 2D illustration of Mei projection model [Söd15]

extrinsic grid parameters. Afterwards, the grid pattern is reprojected and a global minimization can be done when calibrating the intrinsic camera parameters.
Unfortunately, this calibration toolbox is not working automatically because the user has to select and click on the four outer grid corners in every image. Also, it is not possible to relocate the position of a displaced corner manually. Therefore images, which work poorly with the corner detection should be removed from the calibration dataset. In addition MATLAB [2.5.6] is required to use this tool and for that, a charged license is needed. Because of these drawbacks, Mei's calibration toolbox cannot be part of an easy-to-use calibration script.

2.3.2 Scaramuzza's OcamCalibToolbox

Another calibration toolbox for MATLAB has been provided by Scaramuzza [OCa] based on his work [SMS06]. It can be used for any central omnidirectional cameras including fisheye lenses. There are similarities to Mei's toolbox, because both require MATLAB, make use of the chessboard calibration pattern and can be used for calibrating the intrinsic and the distortion parameters. The user needs to collect some images of the calibration pattern shown in different angles and positions. An advantage over Mei's toolbox is the possibility to detect the corners automatically instead of manually proceeding. In a case of a faulty automatically detection the corners positions can also be changed manually. A disadvantage of this toolbox is, that coordinate system is defined differently regarding not using the standard right-handed coordinate system. When using the calibration results in terms of rotation and translation this has to be corrected.
Mathematically the simple pinhole camera model [2.2.3], even incorporating distortions, is not able to model field of views with angels larger than 180°. Already with much smaller angles problems can occur in practice. That is

why Scaramuzza presented a new camera model [SMS06] which allows higher viewing angles. Systems exhibiting strong radial distortion like mirror lens optics(catadioptric) [2.13b], wide angle [2.13c] and fisheye lenses [2.13a] can be used with the proposed approach. Those distortions can be recognized in fisheye lenses when the middle of the image has a higher zoom factor than the edges. Because of these distortions, relative sizes change and straight lines become bent.



(a) fisheye (b) Catadioptric system (c) Wide angle

Figure 2.13: Images taken with different optics [Hof13]

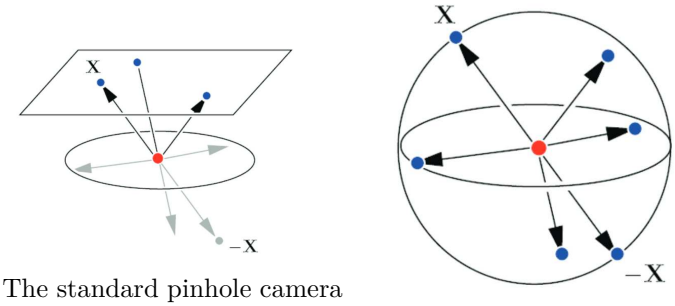
Scaramuzza's model does not simply reposition the pixels on the image plane with a distortion function. Instead it computes a vector $\begin{pmatrix} x & y & z \end{pmatrix}^T$ radiating from the single viewpoint to an picture sphere pointing in the direction of the incoming light ray for each pixel position $\begin{pmatrix} u & v \end{pmatrix}^T$. This reference frame originates in the center of the image. In figure [2.14] the difference of the central projection used in the standard pinhole camera model [2.2.3] and the model by Scaramuzza is shown.

Function $g(p)$ [2.18] is defined as a Taylor polynomial and models the radial distortion. It consists of coefficients a_i which define the intrinsic parameters and the Euclidean distance of the pixel position $\begin{pmatrix} u & v \end{pmatrix}^T$ from the image center through function [2.19]. Function [2.18] is needed to make use of the spherical projections properties, so that a point in camera coordinates can always be represented as a point on a specific ray. This is shown in equation [2.20] with s being an arbitrary scaling factor.

$$g(p) = a_0 + a_1 p + a_2 p^2 + a_3 p^3 + .. + a_n p^n \tag{2.18}$$

$$p = \sqrt{u^2 + v^2} \tag{2.19}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = s \begin{pmatrix} u \\ v \\ g(p) \end{pmatrix} \tag{2.20}$$



(a) The standard pinhole camera model not distinguishing scene points lying on opposite half-lines

(b) The spherical model distinguishing two scene points lying on opposite half-lines

Figure 2.14: Scaramuzza perspective projection limited to view angles covered by the projection plane and spherical projection which covers the whole camera reference frame space [Sca07].

To project a point from camera coordinates onto an image, equation's [2.18] polynomial has to be solved, in order to get u and v. That is computationally inefficient. So Scaramuzza estimated an inverse Taylor polynomial [2.21]. The radial positioning on the image plane based on the angle α of a light ray to the z axis is described by it. Just like Mei's toolbox [2.3.1], Scaramuzzas requires MATLAB. Since there is no full-automatically possibility to calibrate the cameras and license costs of MATLAB, this toolbox is insufficient for an easy-to-use calibration script.

$$f(\alpha) = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3 + .. + b_n\alpha^n \tag{2.21}$$

$$r = \sqrt{x^2 + y^2} \tag{2.22}$$

$$\alpha = \arctan(z/r) \tag{2.23}$$

$$u = \frac{x}{r} f(\alpha) \tag{2.24}$$

$$v = \frac{y}{r} f(\alpha) \tag{2.25}$$

2.3.3 OpenCV camera calibration

The widely used open source computer vision library OpenCV [2.5.2] provides a intrinsic camera calibration tool. It is based on Brown's [Sze10] distortion model. The coordinates x' [2.4] and y' [2.5] are the ideal pinhole camera coordinates. x'' and y'' are their respective repositioning based on the distortion parameters. Those are k_1, k_2, k_3 and the tangential distortion coefficients p_1 and p_2. Those calculated coordinates are sufficient for most

and weaker distortions, but in case of stronger ones the first term of [2.27] and [2.28] are divided by $1 + k_4r^2 + k_5r^4 + k_6r^6$ where k_4, k_5 and k_6 are additional distortion coefficients.

$$r^2 = x'^2 + y'^2 \tag{2.26}$$

$$x'' = x'(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1x'y' + p_2(r^2 + 2x'^2) \tag{2.27}$$

$$y'' = y'(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y'^2) + 2p_2x'y' \tag{2.28}$$

$$u = f_x \cdot x'' + c_x \tag{2.29}$$

$$v = f_y \cdot y'' + c_y \tag{2.30}$$

Especially for fisheye lenses this model has been adjusted [Opeb] in OpenCV version 3. Subsequent the OpenCV 3 fisheye camera model is explained. In equation [2.31] the coordinate vector $\vec{x}_c$ of a point P in the reference frame is shown. This point P is a point in 3D coordinates X in the world coordinate system, which is stored the matrix $\boldsymbol{X}$. The matrix $\boldsymbol{R}$ is the rotation matrix corresponding to the rotation vector $o\vec{m}$: $\boldsymbol{R} = rodrigues(om)$ [Gup89]. Since the point P is a point in 3D coordinates it consists of 3 coordinates $x = \vec{x}_{c_1}$, $y = \vec{x}_{c_2}$, $z = \vec{x}_{c_3}$.

$$\vec{x}_c = \boldsymbol{RX} + \boldsymbol{T} \tag{2.31}$$

The pinhole projection coordinates of P is $\begin{pmatrix} a & b \end{pmatrix}^T$ where $a = x/z$, $b = y/z$, $r^2 = a^2 + b^2$ and $\theta = \arctan r$. Equation [2.32] describes the fisheye distortion:

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8 \tag{2.32}$$

This results in the distorted point coordinates $\begin{pmatrix} x' & y' \end{pmatrix}^T$ where

$$x' = (\theta_d/r)x \tag{2.33}$$

$$y' = (\theta_d/r)y \tag{2.34}$$

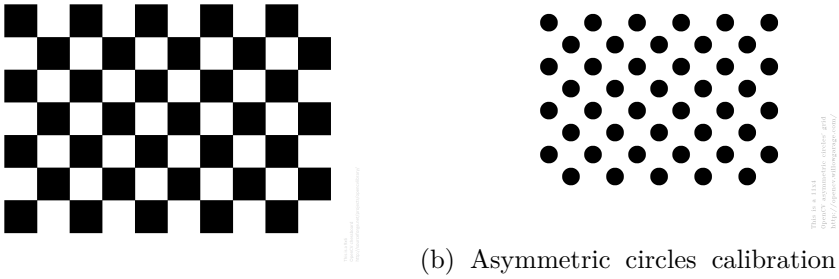
Finally the conversion into pixel coordinates with the resulting vector $\begin{pmatrix} u & v \end{pmatrix}^T$ where

$$u = f_x(x' + \alpha y') + C_x \tag{2.35}$$

and

$$v = f_y yy + C_y \tag{2.36}$$

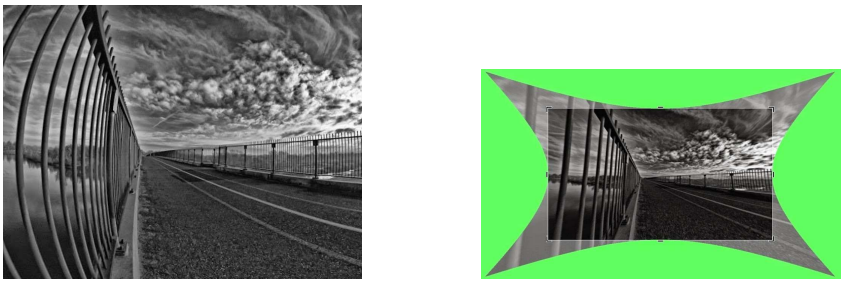
The actual calibration method is based on the work of Zhang [Zha00] and Bouguet [Bou15]. A known pattern has to be printed like in figure [2.15a] on a piece of paper and attached to a rigid surface to make it reasonably planar. Information like the used calibration pattern, the size of its circles or corners and the pure amount of circles or corners must be known.



(a) Chessboard calibration pattern

(b) Asymmetric circles calibration pattern [pat]

Figure 2.15: Calibration patterns



(a) Original fisheye image

(b) Complete rectified fisheye image

Figure 2.16: Fisheye image rectification [Opec]

To calibrate one camera it takes several images of the calibration pattern in various orientations. In the process either the calibration pattern or like in this case the camera must be fixated while moving the other one around. In each image the feature points of the pattern have to be detected by a pattern specific feature detection algorithm like the algorithm for chessboards by Bennett [BL14].
Since the intrinsic and the distortion parameters depend on the camera and its lens, it is not essential to calibrate all wanted cameras at once.
Accompanied by the rectification of curved images is the loss of outer parts of the original fisheye image. Figure [2.16] exhibits an original fisheye image and its completely rectified version. Only the inner tagged rectangle of the image is further used, the rest of the image is excised.
OpenCV's camera calibration is robust, works with multiple calibration patterns and runs automatically. Hence it is a good choice for a camera calibration script.

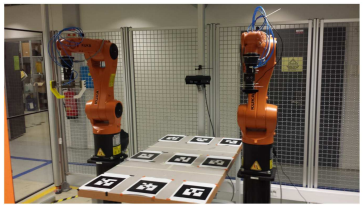
2.4 Extrinsic camera calibration

Analogous to intrinsic camera calibration, the process to estimate the set of extrinsic parameters in a camera model is called extrinsic camera calibration. The extrinsic parameters of a camera are the camera's position, described through the translation vector $\vec{t}$ and its orientation, displayed via the rotation matrix $\mathbf{R}$ as already introduced in [2.2.1].
When multiple cameras are used, like in this thesis, the camera's extrinsic parameters have to be recalibrated, if their position relative to each other is changed. Fortunately, the used broadreach-cameras are firmly installed so that only an extraction and renewed installation or strong impacts would make this necessary.
It is important to estimate the extrinsic parameters for each of the cameras in relation to the same reference coordinate system. This way the transformation between all cameras can be estimated.
Because the relation between the cameras is crucial here, it is an easy possibility to place the reference system in one of the cameras. Thus, the positions of the other cameras can be estimated in relation to the first one.
There are many ways to calculate the extrinsic parameters of a camera system. Subsequent works providing options to gain the extrinsic parameters are presented.

2.4.1 Approaches

There are several options available for calculation the extrinsic camera parameters. First of all the most obvious: measuring of the translation and rotations manually. This is only possible, if the cameras are exposed, like the front [2.1a], left [2.1c] and right [2.1d] camera of the *MIG*, but unlike the rear camera [2.1d]. The body of the last-mentioned is not visible at all, only the camera's lense can be seen. Hence no reasonable assertions can be made about this cameras rotation.
An often used approach to determine the extrinsic camera parameters is the use of markers. Just like the calibration patterns used for intrinsic calibration [2.15], the algorithms recognize the predefined forms of the marker.
The work of Vatolin, Strelnikov and Obukhov [OSV08] presents an approach to fully automatic pantilt-zoom (PTZ) camera calibration, using a visual marker detection system. It focuses on extrinsic parameters only. A set of measurements, each represented by the correspondence between the Cartesian work coordinates and the camera's internal coordinates for a given point, is used. The camera position and rotation can be calculated independently because this approach uses a "direct measurement". The above mentioned visual marker are detected via a corresponding detection system to obtain the world coordinates of specific points, even though internal coordinates are accessible in most cameras. Unfortunately, this approach

(a) ARTag visual marker acting as a calibration object [OSV08]



(b) AR markers used to calibrate robots [Rak16]

requires fixed camera positions and is meant to be used for indoor cameras as can be seen in figure [2.17a], thus adjusting this work for this thesis needs would take have been elaborate.
In the master thesis [Rak16] multiple approaches to get extrinsic parameters were compared to each other. ROS [2.5.1] was used, just like in this thesis working group. That is why, it gives an insight, especially because besides AR(Augmented Reality) marker were used besides point clouds; more concretely the ROS library *ar_track_alvar*. The results of this thesis were promising, but unfortunately when testing this library with the existing hardware got problems with false positive marker recognition.
A completely different technique is presented in the work of Takahashi, Bobuhara and Matsuyama [TNM12]. This paper assumes that there is no direct visibility to a 3D reference object. So this reference object is captured via a mirror under three different unknown poses. Afterwards the extrinsic parameters are calibrated from 2D appearances of the appropriate reflections in the mirror. Since the cameras of the *MIG* can have direct visibility to reference objects, this is not worth taking a deeper look.
A very fast approach has been made in a paper called "extrinsic calibration of a set of range cameras in five seconds without pattern" [FMGJRA14]. It relies on finding and matching planes. The presented method serves to calibrate two or more range cameras, requiring only to observe one plane from different viewpoints simultaneously. According to its authors, this approach is very fast. It is generic, because it does not require a 3D calibration pattern or relies on the robustness of simultaneous localization and mapping (SLAM) or Visual Odometry(VO), which highly depends on the environment. Though this works takes advantage of the fact, that structured environments like floors, walls and ceiling contain large planes, so is made for indoor camera calibration, similar to [OSV08] and therefore it is no option for this thesis.
In the work [TR13] by Raúl Rojas and Ernesto Tapia the pattern of appar-

ent motions of objects or surfaces in a visual scene caused by the relative motion between an observer and a scene is the base for extrinsic calibration. This is called optical flow. Here it is assumed, that the camera is mounted on a moveable object like a robot or a vehicle while pointing forward. Next this object is moving forward and sideways on a flat area to acquire the images needed for the calibration. This way a subset of the segments defined by the optical flow are the projection of parallel line segments of the world coordinate frame. At a common point, the so called vanishing point, this set of parallel lines intersects in the image plane. If the vehicles moves sideways while still capturing the video frames, the set of vanishing points define the horizon. According to this promising work, estimating the direction and position of the camera is possible, if the slope of horizon line and the coordinates of vanishing points of lines parallel to the forward movement of the vehicle is known.
Since this approach requires forward pointing cameras it could have been used for the front camera only or most probably the rear camera as well. Therefore it is not possible to get extrinsic parameters for rest of the camera system. So this calibration method was not chosen.

The work [CAD11] written by Davison, Angeli and Carrera is based on the widely used Simultaneous Localization and Mapping (SLAM) algorithm. It is used on automatically extrinsic calibrate a mulit-camera rig. The same algorithm is part of the chosen extrinsic calibration pipeline CamOdoCal [2.4.2]. No calibration pattern, overlapping fields of view or other infrastructure is required. Known individual intrinsic parameters and synchronised capture across the multiple cameras are assumed, although they can have varying types and optics and uses a multi-camera rig mounted on a mobile robot. The approach has the following steps:

- Attachment of the cameras to a mobile robot in arbitrary rigid positions and orientations.
- Pre-programmed movement of the robot through an unprepared environment. Meanwhile the synchronized video streams from the cameras are captured.
- Every cameras video stream is analysed by a modified version of the MonoSLAM algorithm [DRMS07]. This is visualized in figure [2.18]. MonoSLAM builds a 3D map of visual feature locations, while every feature is characterized by a SURF(Speeded Up Robust Features) descriptor. Also the camera motions are estimated.
- By using bundle adjustment, each camera's map and motion estimates are individually refined.

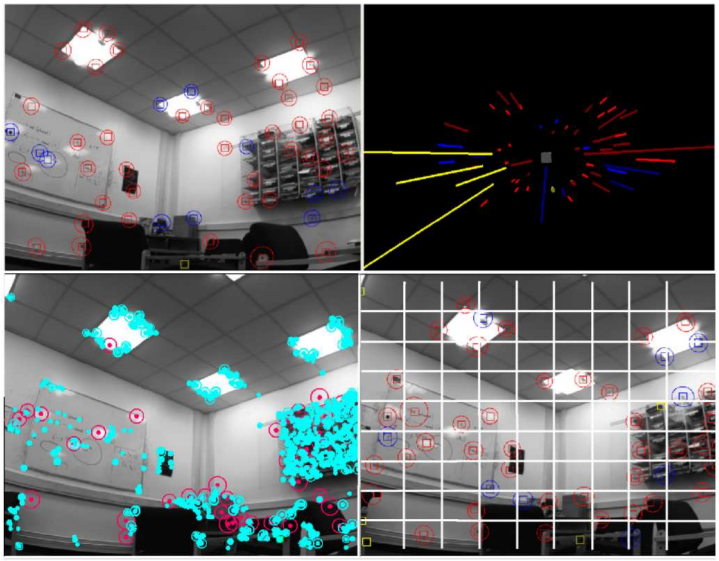


Figure 2.18: MonoSLAM. Top: in operation. Bottom: SURF features added. [CAD11]

- Via thresholded matching between their SURF descriptors, candidate feature correspondences between each pair of individual maps are obtained.
- To generate hypotheses, sets of three correspondences between maps from the SURF [BTVG06] matches are used. These sets in combination with RANSAC [FB81] and 3D similarity alignment is used, to achieve an initial alignment of the maps' relative 3D pose and scale. Afterwards a reliable set of correspondences between each pair of maps is derived, which satisfies descriptor matching and geometrical correspondence. The monocular maps are merged into a single joint map.
- In the end, full bundle adjustment optimization with this initial joint map as the starting point is run to estimate the relative poses of the cameras, 3D positions of scene points and motion of the robot.

Overall results of this approach are satisfying. That is why in this thesis the further developed pipeline CamOdoCal [2.4.2] is part in the subsequent implementation.

2.4.2 CamOdoCal

One of the key aspects of this thesis's goal is camera calibration and hence also the calculation of extrinsic parameters. Therefore within this thesis, an easy-to-use calibration script has to be implemented, which has some requirements. The user interaction should be minimized and optimally there should not be any expenses for the requirement tools. Hence open source applications are preferred. Therefore this thesis uses *CamOdoCal*(**Cam**era **Odo**metry **Cal**ibration) for calculating the extrinsic parameters. It is an open source pipeline written in C++, available at GitHub [Cam] and is based on the work of Heng, Li and Pollefeys [HLP13] as part of the autonomous car project V-charge of the ETH-Zürich. CamOdoCal is the base for further works like [HFP15]. The pipeline is able to calculate both intrinsic and extrinsic parameters. Though the intrinsic calibration is very similar to the one provided by Mei [2.3.1], only the extrinsic calibration will be discussed. The name of the framework directly tells which data is used for calibration: camera images and wheel odometry. These data has to be recorded while driving around with the vehicle for a short time at a rich textured environment. The extrinsic calibration finds all camera-odometry transforms, is unsupervised and uses natural features. Another advantage of CamOdoCal over the other approaches is its flexibility, because it is able to compute and handle parameters of the pinhole camera model [2.2.3], the unified projection model [2.3.1] and the equidistant fisheye model [KB06].
In short, CamOdoCal does the following steps when calibrating the extrinsic parameters:

1. Visual odometry for each camera
2. Triangulate 3D points with feature correspondences from mono visual odometry and run bundle adjustment
3. Run robust pose graph simultaneous localization and mapping (SLAM) and find inlier 2D-3D correspondences from loop closures
4. Find local inter-camera 3D-3D correspondences
5. Run bundle adjustment

Ensuing, a more detailed summary of the single steps strongly based on the CamOdoCal paper [HLP13] is presented.

Monocular visual odometry

First for each camera separately monocular visual odometry with sliding window bundle adjustment is run. Visual odometry is the process of determining the orientation and position of an moving object, that has a camera affixed like a robot or a vehicle, by analyzing its camera images. In the work

of Triggs, McLauchlan, Hartley and Fitzgibbon [TMHF99] bundle adjustment is explained in the following way: "Bundle adjustment is the problem of refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameter (camera pose and/or calibration) estimates. Optimal means that the parameter estimates are found by minimizing some cost function that quantifies the model fitting error, and jointly that the solution is simultaneously optimal with respect to both structure and camera variations. The name refers to the 'bundles' of light rays leaving each 3D feature and converging on each camera centre, which are 'adjusted' optimally with respect to both feature and camera positions. Equivalently - unlike independent model methods, which merge partial reconstructions without updating their internal structure - all of the structure and camera parameters are adjusted together 'in one bundle'.".
CamOdoCal uses the five-point algorithm [KRC$^+$11] for performance reasons and linear triangulation to triangulate the feature points in the last views. While running the linear triangulation it is examined, that the reprojection error of the resulting 3D scene point in the first view does not exceed a threshold. When the corresponding odometry pose is at least 0.25 meters away from the odometry at which the last set of keyframes was taken, a synchronized set of keyframes from all cameras is extracted. The OpenCV implementation of SURF(Speeded Up Robust Features [BTVG06]) is used to extract feature points and their descriptors, match feature points via the distance ratio metric and lastly find inlier feature point correspondences through geometric verification. If a feature point correspondences does not fit to a previously initialized 3D scene point, this correspondence is triangulated. Otherwise the already initialized 3D scene point is associated to the correspondence. The current camera pose is found by the iterative form of PnP RANSAC [FB81] from 3D-2D point correspondences. At each iteration of monocular visual odometry a sliding window bundle adjustment is used. The inlier feature point tracks are shown in figure [2.19].

Initial estimate of the camera-odometry transform

Secondly for creating a initial estimate of the camera - odometry transform, the method [GMR12] is used and also expanded to act more robust in feature-poor environments. The following variables will be used:

- q_{yx} = pitch-roll quaternion
- q_z = yaw rotation quaternion
- $\vec{t}_{O\to C}$ = translation transforming the camera frame to odometry frame, with $\vec{t}_{C\to O} = \begin{pmatrix} t_x & t_y \end{pmatrix}^T$
- $\vec{t}_{C_i\to C_{i+1}}$ = translation transforming camera frame i to camera frame $i+1$

- $\vec{t}_{O_i \to O_{i+1}}$ = translation transforming odometry frame i to odometry frame $i+1$
- s_j = scale for each of the m sparse maps
- $q_{O_i \to O_{i+1}}$ = unit quaternion rotating odometry frame i to odometry frame $i+1$
- $q_{C_i \to C_{i+1}}$ = unit quaternion rotating camera frame i to camera frame $i+1$
- $q_{C \to O}$ = unit quaternion rotating camera frame to odometry frame, with $q_{C \to O} = q_z q_{yx}$

A quaternion can be thought of a composite of a scalar and a vector, a vector with four components, or as a complex number with three different imaginary parts [Hor87]. It is possible to convert between a rotation matrix and a quaternion [K$^+$99]. Since quaternions are more compact than rotation matrices, they are used as a representation of rotations in this subsection.
The hand-eye calibration problem consists of calculating the rotation and translation between a sensor like a camera on a robot actuator and the actuator itself [HD95]. It is described using quaternion representation is described via equations [2.37] and [2.38].

$$q_{O_i \to O_{i+1}} q_{C \to O} = q_{C \to O} q_{C_i \to C_{i+1}} \tag{2.37}$$

$$(\boldsymbol{R}(q_{O_i \to O_{i+1}}) - \boldsymbol{I}\vec{t}_{C \to O}) = s_j \boldsymbol{R}(q_{C \to O})\vec{t}_{C_i \to C_{i+1}} - \vec{t}_{O_i \to O_{i+1}} \tag{2.38}$$

Rotations around the z-axis commute. This is used to estimate the pitch-roll quaternion by substituting $q_{C \to O} = q_z q_{yx}$, as shown in equation [2.39].

$$q_{O_i \to O_{i+1}} q_{yx} - zyx q_{C_i \to C_{i+1}} = 0 \tag{2.39}$$

There are two constraints on the unknown q_{yx}. The first constrain: the x-value and y-value of the pitch-roll quaternion multiplied with each other equals the negative z-value multiplied with the w-value of the same pitch-roll quaternion. The second constrain: the transposed pitch-roll quaternion multiplied with the original one equals one. These constrained are mathematically expressed in equation [2.40] respectively [2.41].

$$q_{yx_x} q_{yx_y} = -q_{yx_z} q_{yx_w} \tag{2.40}$$

$$q_{yx}^T q_{yx} = 1 \tag{2.41}$$

With at least two or more motions a $4n \times n$ matrix $\mathbf{T}$ [2.43] is build:

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{S_1}^T & ... & \boldsymbol{S_n}^T \end{pmatrix}^T \tag{2.42}$$

where $\mathbf{S}$ is a 4×4 matrix resulting of equation [2.38]. With the unitary matrix $\mathbf{U}$ the singular value decomposition is found: $\boldsymbol{T} = \boldsymbol{USV}^T$. The null space of $\mathbf{T}$ is encompassed by the last two columns of $\mathbf{V}$: the last two right-singular vectors $\vec{v_3}$ and $\vec{v_4}$, as displayed in equation [2.43].

$$q_{yx} = \lambda_1 \vec{v_3} + \lambda_2 \vec{v_4} \tag{2.43}$$

Finally q_{yx} is obtained by using the constraints [2.39] and [2.40] to solve for λ_1 and λ_2.

Due to planar motion, the z-component of the camera-odometry translation is unobservable. So the third row from equation [2.38] can be removed, resulting in equation [2.44].

$$\begin{pmatrix} \cos\beta - 1 & -\sin\beta \\ \sin\beta & \cos\beta - 1 \end{pmatrix} \begin{pmatrix} t_x \\ t_y \end{pmatrix} - s_j \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \vec{t}_{O_i \to O_{i+1}} = 0 \tag{2.44}$$

Here, α is the yaw, $\vec{t}_{O_i \to O_{i+1}}$ denotes the first two elements of the translation vector $\vec{t}_{O_i \to O_{i+1}}$ and $\begin{pmatrix} p_1 & p_2 \end{pmatrix}^T$ are the first to elements of the vector $\boldsymbol{R}(q_{yx})$. With $\mathbf{J} = \begin{pmatrix} \cos\beta - 1 & -\sin\beta \\ \sin\beta & \cos\beta - 1 \end{pmatrix}$ and $\mathbf{K} = \begin{pmatrix} p_1 & -p_2 \\ p_2 & p_1 \end{pmatrix}$ equation [2.44] is rewritten as a matrix vector equation in equation [2.45].

$$\begin{pmatrix} \boldsymbol{J} & \boldsymbol{K} \end{pmatrix} \begin{pmatrix} t_x \\ t_y \\ -s_j \cos\alpha \\ -s_j \sin\alpha \end{pmatrix} = -\vec{t}_{O_i \to O_{i+1}} \tag{2.45}$$

Afterwards the $2n \times (2 + 2m)$ matrix $\mathbf{G}$ is build in [2.46], with $n = \sum_{i=1}^{m} n_i$ because of m visual odometry segments with $n_1 \geq 2, ..., n_m \geq 2$ motions each. $\boldsymbol{J}_i^j$ and $\boldsymbol{K}_i^j$ are the respective versions of $\mathbf{J}$ respectively $\mathbf{K}$ to the ith motion in the visual odometry segment j.

$$\boldsymbol{G} = \begin{pmatrix} \boldsymbol{J}_1^1 & \boldsymbol{K}_1^1 & ... & 0 & 0 & ... & 0 & 0 \\ ... & ... & ... & 0 & 0 & ... & 0 & 0 \\ \boldsymbol{J}_{n_1}^1 & \boldsymbol{K}_{n_1}^1 & ... & 0 & 0 & ... & 0 & 0 \\ 0 & 0 & ... & \boldsymbol{J}_1^j & \boldsymbol{K}_1^j & ... & 0 & 0 \\ 0 & 0 & ... & ... & ... & ... & 0 & 0 \\ 0 & 0 & ... & \boldsymbol{J}_{n_j}^j & \boldsymbol{K}_{n_j}^j & ... & 0 & 0 \\ 0 & 0 & ... & 0 & 0 & ... & \boldsymbol{J}_1^m & \boldsymbol{K}_1^m \\ 0 & 0 & ... & 0 & 0 & ... & 0 & 0 \\ 0 & 0 & ... & 0 & 0 & ... & \boldsymbol{J}_{n_m}^m & \boldsymbol{K}_{n_m}^m \end{pmatrix} \tag{2.46}$$

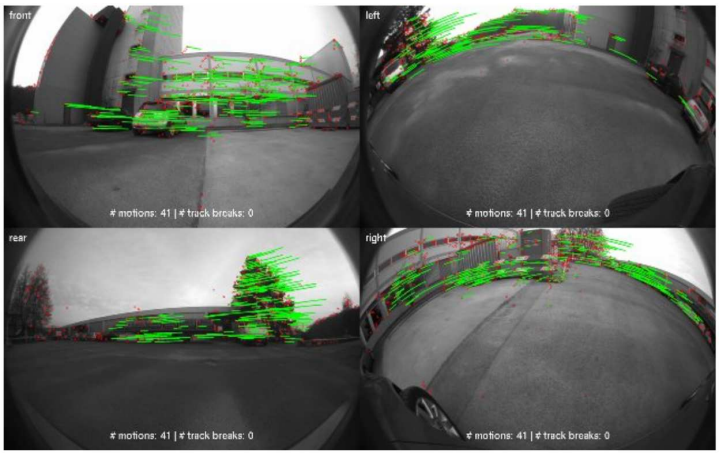


Figure 2.19: Inlier feature point tracks [HLP13]

Using Matrix **G** [2.46] equation [2.45] is extended and rewritten to involve the motions of the visual odometry segements in equation [2.47]:

$$\boldsymbol{G}\begin{pmatrix} t_x \\ t_y \\ -s_0\cos\alpha_0 \\ -s_0\sin\alpha_0 \\ \dots \\ -s_m\cos\alpha_m \\ -s_m\sin\alpha_m \end{pmatrix} = -\begin{pmatrix} \vec{t}_{O_0\to O_1} \\ \dots \\ \vec{t}_{O_n\to O_{n+1}} \end{pmatrix} \tag{2.47}$$

Now, to find the solution to $\vec{t}_{C\to O} = \begin{pmatrix} t_x & t_y \end{pmatrix}^T$, s_j and α_j, the least square motion is used. This estimates the scale s_j for each visual odometry segment besides the translation vector $\vec{t}_{C\to O}$, but still there are m hypotheses of α. The best hypothesis that minimizes the cost function [2.48] is chosen, before the estimate of s_j, α, q_{yx} and $\vec{t}_{C\to O}$ are refined by using non-linear optimization to minimize the just mentioned cost function C [2.48].

$$C = \sum_{i=0}^{n_j}((\boldsymbol{R}(q_{O_i\to O_{i+1}}) - \boldsymbol{I})\vec{t}_{C\to O} - s_j\boldsymbol{R}(\alpha)\boldsymbol{R}(q_{yx})\vec{t}_{C_i\to C_{i+1}} + \vec{t}_{O_i\to O_{i+1}}) \tag{2.48}$$

3D point triangulation

In the third step, all features of every frame which are visible in the current frame and the last two frames, and do not correspond to a previously initialized 3D scene point are found for each camera while iterating through

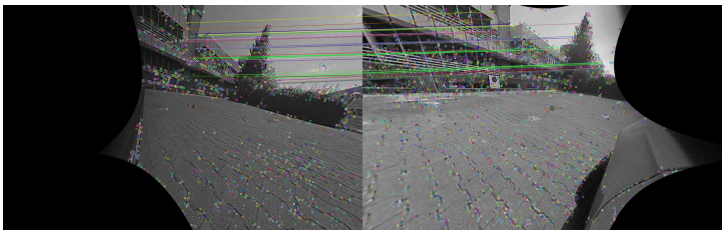


Figure 2.20: Inlier feature point correspondences between rectified images

every frame. Each feature correspondence in the last and current frame is triangulated. If a threshold of 3 pixels for the reprojection error of the resulting 3D scene point in the second last frame does not exceed, this 3D scene point is associated with the corresponding feature in the second last frame. Afterwards, each feature track is associated with the same 3D scene point which the first three features already correspond to. To optimize the extrinsic parameters and 3D scene points, bundle adjustment is run via the Ceres solver [AM12]. This minimizes the image reprojection error across all frames for all cameras.

Finding inter-camera feature point correspondences

As the fourth step, a local frame history for each camera is retained by iterating over each odometry pose of increasing timestamp. For every possible pair of cameras at each odometry pose the features in the first camera's current frame are matched against those in the second camera's frame history. This is done to find the frame in the history that provides the highest number of inlier feature points correspondences. The image pair is rectified on a common image plane which corresponds to the average rotation between the first camera's pose and the second camera's pose before the feature matching. By using the current extrinsic estimate and odometer poses, the camera poses are calculated. The rectification is done to ensure a high number of inlier feature point correspondences. In figure [2.20] an example of a rectified image pair between two cameras is shown. By reprojecting 3D scene points in the map corresponding to the first camera into the first rectified frame and equally for the second camera, the subset of inlier feature correspondences that correspond to 3D scene points found in the map. If a threshold of 2 pixels is exceeded by the image distance between the feature points and corresponding projected 3D scene point, the feature correspondence is rejected. This is done for each inlier feature correspondence.

Loop closures

The penultimate step is loop closures detection. This is done for enhancing the robustness of the just applied SLAM algorithm [Loo]. The loop closure

problem consists in detecting when the vehicle or robot has returned to an already visited location after having discovered new areas for a while. It is possible to increase the precision of the current pose estimate. In this case, it is also a mutual verification between the poses calculated by CamOdoCal and the odometry poses. Following is done for each frame for each camera. Features of the corresponding image are converted into a bag-of-words vector and this vector is added to a vocabulary tree, whereby a DBoW2 [GLT12] implementation is used. The n most similar images are found. Then the matched images, which belong to the same monocular visual odometry segment and whose keyframe indices are within a certain range of the query images keyframe index are filtered, to avoid unnecessary linking of frames whose features may already belong to common feature tracks.

Full bundle adjustment

Lastly full bundle adjustment is performed, optimizing all extrinsic parameters, odometry poses and 3D scene points.
Because the z-component of the camera-odometry translation is unobservable due to planar motion, the relative heights of the cameras are only estimated, hence a hand measurement is recommended.

2.5 Used libraries and software

This section introduces the libraries and software used in this thesis.

2.5.1 ROS

ROS stands for **R**obot **O**perating **S**ystem and is a modular framework for robotic software projects. It provides package management, IPC (inter-process communication) and hardware abstraction. The communication between the components is message-based. Using the publish-subscribe pattern, each component (or node) can subscribe to so-called topics and push message to them. The structure of a message is defined static. It composes different fields of primitive types, arrays of primitive types and other messages. ROS also offers so-called nodelets, that act similar to nodes. Just like when comparing threads to processes, nodelets have the advantage over nodes of zero copy costs between nodelets of the same nodelet handler. So there are no costs of copying or serialization between nodelets of the same nodelet handler, despite running different algorithms. Recordings of ROS nodes, for example of images captured by a camera, are called rosbags. Hence the working group autonomous cars of this thesis is mainly using ROS, it is mandatory using it for this thesis. All implementations created for this thesis have the option to be run as a nodelet or as a node.

2.5.2 OpenCV

OpenCV [Opea] stands for **Open** Source **C**omputer **V**ision Library. It is an open source library for machine learning and computer vision. The source code of OpenCV is available at GitHub [Oped]. Hence it is easy to modify the code for specific needs. It was built to provide a common infrastructure to accelerate the use of machine perception and a common infrastructure for computer vision. Well-established companies like Google, Microsoft, Intel, Sony or IBM employ the library. There are C++, C, Python, Java and MATLAB interfaces available and the library supports Microsoft Windows, Linux, Android and Mac OS. Thanks to taking advantage of MMX, SSE instructions when they are available alongside with CUDA and OpenCL interfaces, mostly real-time vision applications are possible. This thesis uses OpenCV as a dependency of CamOdoCal [2.4.2], its intrinsic camera calibration implementation [2.3.3] and its algorithms to rectify images which were introduced in version 3.0 alongside many other functionalities for fisheye camera lenses.

2.5.3 LibPCAP

LibPCAP [Libc] is a portable C/C++ library for network traffic capture. Functionally, it is the sibling of its Windows pendant WinPCAP [Win].

PCAP stands for **p**acket **cap**ture and is used to get direct access to packets from the network interface. Famous tools using LibPCAP are TCPdump and Wireshark [Wir]. LibPCAP purpose in this thesis implementation is establishing a connection between the installed camera system [2.1] and ROS [2.5.1] via TCP/IP.

2.5.4 LibJPEG-turbo

Due to our focus on image calibration, an efficient image processing process is obviously indispensable. Libjpeg-turbo [Libb] is a huge improvement over the widely used libJPEG [liba]. It is a JPEG image codec, which uses SIMD instructions like MMX and SSE2 to accelerate baseline JPEG compression and decompression on modern systems like x86, x86-64 and ARM. LibJPEG-turbo is generally two up to six times faster as libJPEG, or on other systems, it is as least equal. It even outperforms other rivals using proprietary high-speed JPEG codes in many cases, thanks to its highly optimized Huffman coding routines. LibJPEG-turbo has the traditional libJPEG API, as well as the less powerful but more straightforward TurboJPEG API implemented. In this thesis it is used for decompressing the compressed JPEG images, the cameras are sending over the network.

2.5.5 Docker

Docker [Doca] is a software container platform. When using container, everything required to make a piece of software run is packed into isolated containers. Because of that, it is not necessary to install all required dependencies on a running real existing machine with the possibility to generate conflicts. This way a container grants stability. While virtual machines bundle a full operating system, Docker containers only bundle the required libraries and settings required to make the software work. In consequence, the software will run the same, regardless of where it is deployed. Docker is available for Linux, Microsoft Windows and Mac OS. All in all Docker containers are efficient, lightweight and self-contained systems. Hence Docker containers are a logical choice for this thesis because ROS [2.5.1] using only a trimmed version of OpenCV [2.5.2] can conflict with a full installation of OpenCV, that is required by CamOdoCal [2.4.2].

2.5.6 MATLAB

Similar to OpenCV [2.5.2] MATLAB [Mat] is a software environment for mathematical calculations, widely used in science and engineering, particularly for machine learning, signal processing, image processing, computer vision, robots and more. The name *MATLAB* stems from its optimization for matrix calculations (**Mat**rix **Lab**oratory). Nonetheless, in the meanwhile MATLAB has also been used in statistics, economic optimization and

modeling, or biological problems. MATLAB has two advantages concerning this thesis: multiple available calibration libraries like Mei's toolbox [2.3.1] and Scaramuzza's toolbox [2.3.2] are implemented using MATLAB and were thus easy to adopt. Also, MATLAB is a rigorously tested environment used in many production settings. Thus it is both reliable for future use and scalable when later used on larger machines/clusters. But it has also two disadvantages: its license costs and that both toolboxes cannot be run completely automatically. Therefore MATLAB is only used to get additional insights but not as part of the implementation.

Chapter 3

Implementation

After introducing the fundamentals of camera calibration in the previous chapter [2] the upcoming chapter covers the made implementations in order to achieve this thesis goals [1.3] in chapter [1]. At the beginning of this thesis the cameras were installed and wired, but there was no possibility to get the camera's pictures and consequently no link to the used framework. So the first essential part of the implementation is to write a camera driver [3.1]. Since the cameras are sending compressed JPEG-Images over the network [2.1] and decompression is time-consuming and produces a high computational load it is advisable to implement an efficient ROS node [2.5.1] only for decompression [3.2]. These are the preparations of the thesis.
The actual camera calibration is done by an easy-to-use camera calibration script [3.3]. The calculated intrinsic and extrinsic camera parameters are then used to rectify the fisheye images [3.4], provide those images and to create a surround view [3.5] of the *MIG*.

3.1 The camera driver

When using the cameras for the very first time, their VLAN(virtual local area network) configuration must be adjusted to prevent interferences inside the existing LAN of the *MIG*. Conveniently this can easily be done via the integrated web interface. Once the BroadR-Reach cameras are switched on, they are constantly sending DHCP(dynamic host configuration protocol) requests.
Under the usage of the libPCAP library [2.5.3] the driver responses to this request with specific network packets. Because libPCAP is a C library the camera driver is written in C++. As a result, every camera gets its own fixed IP address and starts sending compressed JPEG pictures. This is happening through one ROS node for each camera with adjusted configurations which contains the IP address, the MAC(Media access control) address, an URL to the camera info file, the cameras name for each camera and the network

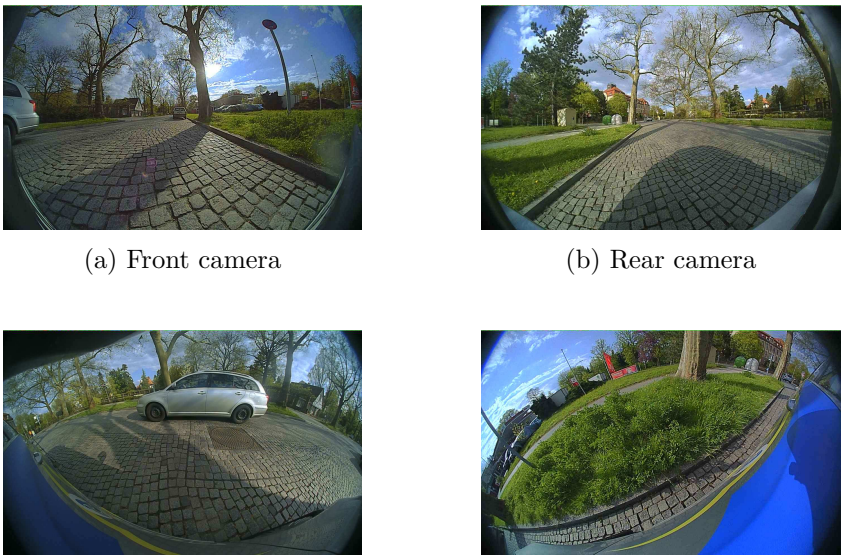
interface to use. The MAC address is the cameras constant hardware address and, because the TCP-IP protocol is used, the destination of the DHCP response packet.
The camera info file comprises the following information about each camera:

- Image width
- Image height
- Camera matrix
- Distortion model
- Distortion coefficients
- Rectification matrix(stereo cameras only)
- Projection matrix

This information are published with the aid of the ROS camera info manager. Other ROS nodes can subscribe to this topic to get the depicted intrinsic calibration.
Due to the allocation of the IP-addresses, each camera is constantly sending compressed JPEG-Packets. One single image cannot be sent within one packet because of TCP-IP limitations. Here one image consists of roughly ninety packets. Therefore the driver differs between the start of a new image with a specific header or the rest of an image and appends the received data until another start packet is received. This is only possible because the images of each camera are sent in the correct order thereby sorting the packets is not required. As soon as an image is complete, the driver publishes it in a specific ROS camera name space, allowing other ROS nodes to subscribe and utilize the image data further. Figure [3.1] displays the result of the driver.

3.2 Decompression of received compressed JPEG images

As mentioned in section [2.1] and [3.1] the cameras immediately start sending compressed JPEG images once IP addresses have been assigned to them. The camera driver [3.1] makes this compressed images available to every subscribed ROS node. Unfortunately, each of those nodes must decompress the images if they want to proceed with for example any OpenCV [2.5.2] operation. It is most likely that the camera images will be used in many different ways like for pedestrian or traffic light detection. For this intention, there would be a ROS node each, which would need to decompress each image by itself. The cameras are sending thirty images each second so there is



(a) Front camera (b) Rear camera

(c) Left camera (d) Right camera

Figure 3.1: The original fish eye images of each camera

already much computation needed to decompress the images for one single ROS node. With each further ROS node this would linearly increase.

For decompressing compressed images widely libJPEG [liba] is used, a C library for reading and writing JPEG Images files, whose latest stable version is dated from the year 1998. It is also integrated into ROS. That is why it is recommendable to write a ROS node only for decompression, again making usage of the ROS C++ API. This way, not every other ROS node has to handle this, and the linear increasing computation power needed for every ROS node is gone. Instead of using libJPEG, an improvement namely libJPEG-turbo [2.5.4] is applied. As a consequence, every single decompression task is generally two till six times as fast as under the utilization of libJPEG.

3.3 Extrinsic and intrinsic camera calibration script

One of the main goals of this thesis is the proper calibration of the fisheye camera system installed in the *MIG*. In the previous chapter three tools for calculating the intrinsic camera parameters and one framework for calculating the extrinsic camera parameters were discussed. For future calibrations a camera calibration script is advisable. The complete camera calibration script has some requirements such as:

1. Compatibility with ROS [2.5.1]

2. Easy to use, with as little user interaction as possible
3. Flexible calculation booth, intrinsic and extrinsic parameters or only one of these
4. Preferably open source usage

Since ROS [2.5.1] is used in this thesis working group [1.2] and also in many other robotic projects the first point of the enumeration is self-explaining. There are two ROS-APIs available: the C++ and the Python-API. The script should be a user-friendly as possible, consequently unlike the other parts of this thesis implementations, the camera calibration script is written in Python. Compiling and starting a Python script is only one step; simply run *python filename.py*, which is much more convenient than working with C++.
Camera calibration is an important but complicated process as can be seen in the comprehensive fundamentals chapter [2]. Most commonly users just want to get the images out of the cameras and proceed with them, be it simple displaying or further computation. To utilize computer vision algorithms calibration parameters are required. It is not mandatory that the user has to know how the parameters are calibrated. The importance of simple usage is huge, if there are too many obstacles in the way to adequate calibrations, the user might use vague results or even give up completely.
For intrinsic calibration with the toolboxes of Mei [2.3.1] or Scaramuzza [2.3.2] the user would have to record the images with the calibration patterns, select them manually, in the case of Mei toolbox manually select every corner, start the toolbox and run multiple other tasks. This is way too much interaction, especially when doing it for multiple cameras. That is one reason why the OpenCV intrinsic camera calibration [2.3.3] is chosen. It automatically detects the calibration patterns over a list of image files and calculates the intrinsic parameters. Still, there are requirements like an installation of OpenCV, which often interferes with the ROS installation. The list of images is also needed and would have to be prepared by the user, besides the configuration describing details of the used calibrated pattern.
As for the extrinsic calibration, especially in the case of CamOdoCal [2.4.2], there are many dependencies to fulfill before the process of extrinsic calibration can start. A look of the frameworks GitHub page [Cam] shows the following required dependencies: *BLAS*, *Boost >= 1.4.0*, *Eigen3*, *glog*, *OpenCV >= 2.4.6*, *SuiteSparse >= 4.2.1* and optionally *CUDA >= 4.2*.

Consequently, there would be a lot of work to do before starting the automatic extrinsic calibration. To relieve the user, the camera calibration script utilizes Docker, which is presented in detailed in subsection [2.5.5]. Briefly, Docker is an open source software, which can be used to isolate operating system virtualizations in containers. For example, it is commonly

used to run web servers inside of containers. One advantage of Docker is that all dependencies and possible conflicts are managed directly inside the container and not the actual user's operation system. This prevents conflicts because if any errors would occur, they would only happen in the container. The camera calibration script deploys Docker to create the environment that required to run the OpenCV intrinsic camera calibration and the extrinsic via CamOdoCal. Luckily, Docker is very performant, so the calculation time does not increase. Having no visual output available is one disadvantage when using Docker.
Now that the dependencies of the tools the calibration script uses for the actual calibration are done by Docker, the data management is another important point for the proper usage of these tools. Specific file name patterns, file formats and correct directory structures or links to them need to be heeded. This also handled by the calibration script.
Overall the users work flow, besides installing Docker [2.5.5], Python and ROS [2.5.1], is as follows:

1. For each camera create a rosbag (recording of ROS nodes containing (de-)compressed image files via *record -o /folder/camera_0_intrinsic.bag -e "/sensors/broadrreachcam_front"*. These images must show always the same calibration pattern [2.15] in different positions and angles. The rosbag files must be ascending named *camera_0_intrinsic.bag* up to *camera_9_intrinsic.bag* and be placed in the camera calibration script root folder.

2. Create a single flexible named rosbag file: *rosbag record -o /folder/filename -e "/sensors/broadrreachcam_(.*)/image_compressed/compressed | /sensors/applanix/gps_odom"*. It must contain about 5 minutes of simultaneous recordings of all camera images and odometry poses while driving around in a feature rich, flat and quiet area. It is important to drive in circles in a small area, so that every sight has been seen by each camera ideally multiple times. A successful driven path can be seen in figure [3.2].

3. Adjust in the file *configuration.yml* the following entries:

    - ros_image_topics - the ROS topics publishing the wanted images
    - ros_odometry_topic - the ROS topics publishing the wanted odometry poses
    - max_images - the maximal amount of images to use
    - BoardSize_Width - number of inner corners per a item row. (square, circle)
    - BoardSize_Height - number of inner corners per a item column. (square, circle)

- Square_Size - the size of a square in some user defined metric system (pixel, millimeter)
- Calibrate_Pattern - the type of input used for camera calibration. One of: *CHESSBOARD*, *CIRCLES_GRID* or *ASYMMETRIC_CIRCLES_GRID*
- ref_camera_height the reference camera's(= camera_0) height

4. run the command *python complete_calibration.py* in the camera calibration script root folder

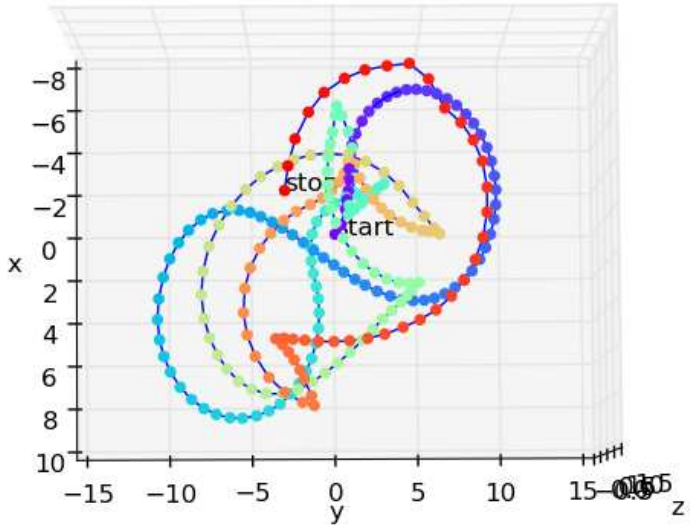


Figure 3.2: Visualization of a successful driven path used by CamOdoCal for extrinsic camera calibration

In the end for each camera the intrinsic calibration files generated by OpencV and the log file written by *CamOdoCal* are placed in the results folder. The intrinsic calibration files are XML files containing general image parameters such as the image's width and height, information about the used calibration pattern and the desired camera matrix alongside with the distortion coefficients. These files can be directly imported by the OpenCV library for further usage such as an image rectifier [3.4]. The *CamOdoCal* log file has a lot of text about the process of the extrinsic calibration, which is useful for debugging. If the framework succeeded, the wanted translation vector and rotation information in form of the classical rotation matrix or as a quaternion are at the end of the log file.
Referring to the fourth point, flexible intrinsic and/or intrinsic parameter calculation, of the camera calibration scripts requirements list [4] there is an option implemented, to choose which parameters should be calculated.

With *python complete_calibration.py intrinsic* only the camera matrix and distortion coefficients are calculated within few minutes. On the contrary via *python complete_calibration.py extrinsic* the rotation matrix and translation vector are calculated within a few hours, but this needs complete intrinsic camera calibration files with file names like *camera_0_intrinsic_calibration.xml* to be placed inside the subfolders *Docker-share/results*. These files must contain the camera matrix and distortion coefficients alongside other information. In terms of the wanted flexibility it is possible to use different calibration patterns, thanks to OpenCV intrinsic calibration, as long as the file *configuration.yml* has been properly adjusted.

Recapitulating the camera calibration script has the following steps:

1. Read the configuration file to determine the ROS image topics and the odometry topic besides the maximal of images to employ.
2. List all rosbag files, that are ascending named *camera_0_intrinsic.bag* up to *camera_9_intrinsic.bag* placed in the root folder of the camera calibration script.
3. Extract all image out of the intrinsic rosbag files if the image topics match with the ones in the configuration file. For easier corner detection the extraction is monochrome.
4. Extract all images and odometry poses in the right format out of the other rosbag
5. Build and run a Docker container to create an environment that satisfies the dependencies of CamOdoCal and the OpenCV intrinsic camera calibration.
6. Edit the source code of the OpenCV intrinsic camera calibration and CamOdoCal that there is no visual output, because this is complicated to realize with Docker and only needed for debugging or getting an better insight of the calibrations process
7. Create a list of images to use for intrinsic calibration for each camera
8. Use this image list and the configuration file to calculate the intrinsic parameters
9. Ensure that all required files are at the correct place, and that for each set of images there is one predecessor and successor odometry pose file
10. Run CamOdoCal to calculate the extrinsic parameters. This process has a high disk space demand. 4 Cameras and 500 pictures each resulted in a total Docker container size of around 11 GB before any calculations and when finished of about 75 GB

11. Copy the resulting files from inside the container to the camera calibrations project folder *Docker-share/results* for user access

3.4 Rectification of distorted fisheye images

By using the camera calibration script [3.3], the camera matrix and distortion coefficients are available. To evaluate these generated parameters, they are used in a specific ROS node to remove the distortion of the fisheye lens. The camera calibration script uses OpenCV [2.5.2] in version 3, in which the support for fisheye lenses, based on the model of [KB06] were added, just as described in [2.3.3]. Alongside the option to calibrate fisheye cameras this version added several new functions and especially an adjusted version of the *initUndistortRectifyMap* function. This function computes the rectification transformation and the joint undistortion applying the intrinsic parameters and represents the results in a form, the *remap* function is capable of handling. Because the intrinsic parameters are not changing over time, the *initUndistortRectifyMap* function is only called once for each camera. Afterwards, the resulting maps are applied for every new incoming image using the *remap* function, which applies a generic geometrical transformation to an image. There is one rectification ROS node for every camera available. Figure [3.3] displays the changes, that made curvy lines in [3.3a] back straight again in [3.3b]. This implementation is the base of the intrinsic camera parameters verification in subsection [4.1.1] in the subsequent chapter results and conclusion [4].

(a) Fisheye image (b) Rectified image

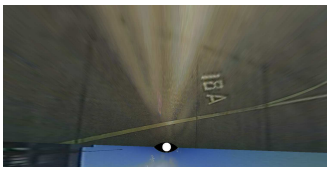
Figure 3.3: The original fish eye and rectified image of the rear camera

3.5 Surround view of the MIG

After successfully calibrating the fisheye camera system installed in the *MIG*, theses parameters will find their application in a surround view. A possible application of this view is the detection of lanes, parking spots or simply the termination of dark spots. A surround view is a compound of all available

top-down views. In order to do this Inverse Perspective Mapping [MBLB91] is needed. It is a coordinate system transformation to transform one perspective to another. Here Inverse Perspective Mapping is used to remove the effects of perspective distortion of the plane surface towards an undistorted top down view. So this part of the implementation does not rely on the rectification node of section [3.4], but its input are the decompressed original fisheye images. In figure [3.4] one single transformation of the original distorted fisheye image towards an undistorted top view image can be seen. The white dot represents the camera's position and every pixel below the horizon going through this point shows the sky and whatever is in the upper area of the original image. Lines that appear to converge near the horizon are straightened.

(a) Original front fisheye image



(b) Undistorted front top view image after perspective transformation

Figure 3.4: The original fish eye and undistorted top view image of the front camera

To generate a single top view in order to later generate a surround view or bird's eye view, the mapping of a point on the road surface (x_w, y_w, z_w) to its projection on the image plane (u, v) needs to be found [TOJG10] [VBPG14]. This requires a rotation about θ, which is a translation along the camera's optical axis, besides a scaling by the camera parameter matrix [3.2]. Figure [3.5] displays the image coordinate system in relation to the world coordinate system.
Equation [3.1] is an expression of this mapping. $\mathbf{K}$ is the camera parameter matrix with $f = px * C_z$ as the focal length, where px is the determined value of the pixels per meter and C_z is the camera's height as part of the cameras calculated translation vector. C_x respectively C_y are the two remaining parts of the camera's position in the world coordinate system. The matrix $\mathbf{T}$ described in [3.3] is the translation matrix, that rotates the image in a way it is looking up and the last matrix $\mathbf{R}$ is the rotation matrix, already calculated by the extrinsic calibration, but the angle θ from figure [3.5] is know it is part of equation [3.4].

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \boldsymbol{KTR} \tag{3.1}$$

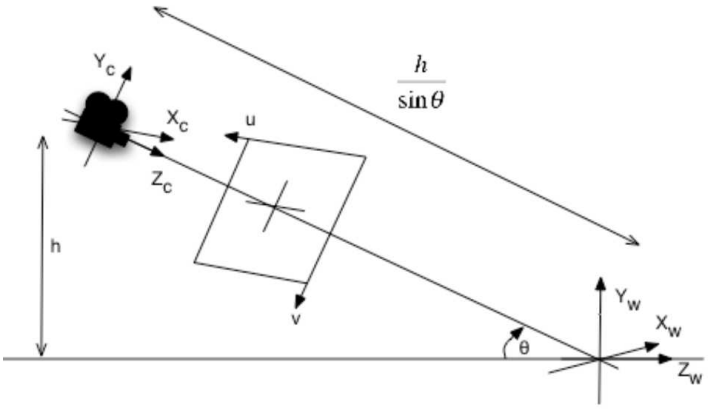


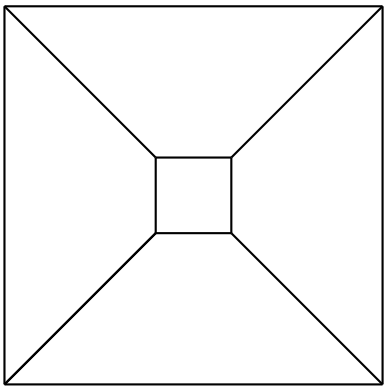
Figure 3.5: Image coordinate system in relation to world coordinate system [TOJG10]

$$\boldsymbol{K} = \begin{pmatrix} f & 0 & C_x \\ 0 & f & C_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

$$\boldsymbol{T} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

$$\boldsymbol{R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix} \tag{3.4}$$

The result of equation [3.1] is then applied with the already in the Rectification ROS node [3.4] used OpenCV function *initUndistortRectifyMaps* along side the intrinsic calibrations of the camera. Hence all camera calibrations are used when generating the surround view, which is why this is a valid verification method.
The just described approach is applied for every new image of every camera. Therewith for every camera, a top view is available, but to get a surround view these must be connected. Here the single top view images are simply copied to a structure similar to figure [3.6a] with the front cameras top view put in the front trapezoid part of the structure, the left camera top view put in the left trapezoid part of the structure and so on. The center shows a vehicle model for better orientation so that the overall image is likewise to the ego vehicle coordinate system [2.2.2]. Also, the positions of the cameras are visualized via a small white circle. The surround view is implemented in C++ and figure [3.6b] is one outcome of this implementation.



(a) Structure of the surround view

(b) Surround view of the *MIG*

3.6 Overview of the Implementation

The final section of this chapter displays an overview over the implementations of this thesis. Figure [3.7] is a sequence diagram showing an abstracted view of the implemented ROS node and their relationships to each other. All of them are implemented in C++. When the cameras are switched on, they are sending DHCP-requests in a loop until they receive a proper reply to their requests. This reply is only sent once. Afterwards they repeatedly send compressed JPEG packets over the network until they are switched off. The driver [3.1] assembles this packets to complete compressed JPEG images and publishes them for each of the cameras. In turn, the decompression ROS node [3.2 is subscribed to the camera driver ROS nodes, decompresses the images and publishes these decompressed JPEG images. For each camera, there is one decompression ROS node. Every ROS node that wants to utilize the images of the cameras is required to subscribe to the decompression node. So there are four single rectification ROS nodes [3.4] that remove the fisheye distortion and one surround view or birds-eye view ROS node 3.5] that is subscribed to all decompression ROS nodes at once, because it creates one single surround view out of all available camera images.
The basic structure of the camera calibration script [3.8] is shown in figure [3.4a]. For each camera, one rosbag is recorded showing multiple images of a predefined calibration pattern vertically held towards the camera. This rosbag is needed for the intrinsic calibration. Another rosbag is recorded for the extrinsic calibration. Here, it must contain the recorded images of all cameras and odometry poses while driving sharp curves and in circles in an environment full of features. The recording is done via the utilization of the *MIG*. Now at any available computer, which has Docker, Python and ROS installed the calibration script extracts the recorded data from the rosbags. Next, a Docker container is created, which has access to the just extracted

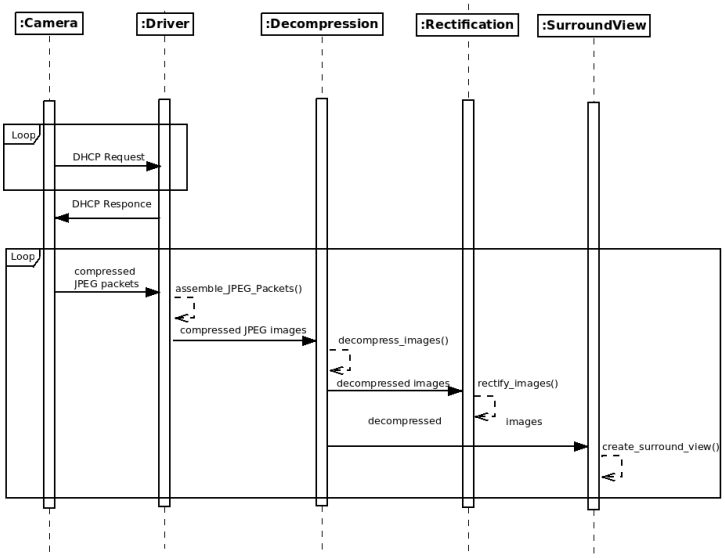


Figure 3.7: UML sequence diagram of the ROS implementation

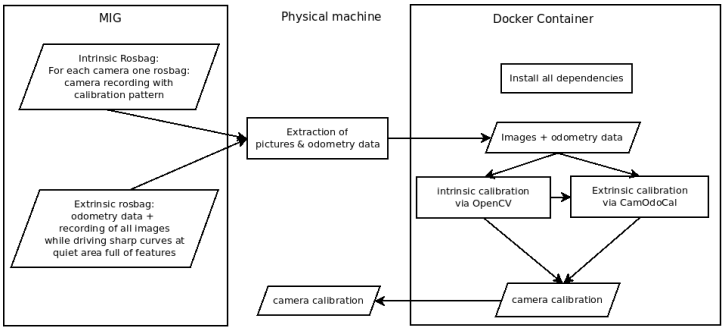


Figure 3.8: Basic structure of the camera calibration script

images and odometry poses. This container installs all dependencies needed by CamOdoCal without having the risk of software conflicts besides some structural tasks. Inside of this container, the OpenCV intrinsic calibration uses the images extracted from the intrinsic rosbags to calculate the intrinsic parameters. Those are employed by CamOdoCal in addition to the images of all cameras and the odometry data of the extrinsic rosbag. CamOdoCal calculates the extrinsic parameters which are then copied back to the physical computer. The same applies to the intrinsic parameters. Afterwards, the Docker container can be destroyed and deleted.

Chapter 4

Results and conclusion

This chapter addresses the received results and their discussion, split in the intrinsic [4.1.1] and extrinsic [4.1.2] parameters that were generated by the camera calibration script. The evaluation of the intrinsic calibration is a visual comparison of the original fisheye images and their distorted version. Whereas the extrinsic calibration evaluation matches the calculated extrinsic parameters with the measured ones. Both calibrations are used in the surround view. That is why the result of this is a benchmark of the whole camera calibration. Moreover, in subsection [4.1.3] the performance in terms of CPU load and how smooth the images are generated, is discussed. After the examination and discussion of the results follows a conclusion of this thesis 4.2, which summarizes the work that has been done. Based on this, the thesis is finalized with a prospect of what could be improved or which works are possible with the outcome of this thesis in the future works section [4.3].

4.1 Results and discussion

First of all, the preconditions to calibrate the fisheye camera calibration system, namely the camera driver [3.1], are successfully implemented as can be seen in several figures like [3.1], [3.3] or [3.4]. Thanks to the driver, the BroadR-Reach cameras [2.1] are sending their compressed JPEG image packets, which are then integrated into the ROS framework. How well the driver performs can be seen in the respective subsection performance [4.1.3]. Same applies for the decompression [3.2] ROS node.
The results of the camera calibration script and the discussion about them [3.3] are split into two subsection: the resulting intrinsic parameters in subsection [4.1.1] and the extrinsic parameters in subsection [4.1.2]. The first mentioned also contains the findings of the rectification implementation [3.4] because it depends on the calculated intrinsic parameters and its results are used as a verification method. As for the extrinsic parameters subsection

[4.1.2], it comprises a visualization of the rotation matrix and the translation vector and the results of the surround view [3.5] which deploys both, the extrinsic and the intrinsic calibration.

4.1.1 Intrinsic parameters

As explained in previous chapters, the installed cameras are delivering fisheye images. That is why straight lines appear twisted. This should be fixed by the rectification applying the calculated intrinsic parameters. To verify these intrinsic parameters, the original fisheye images will be compared to the undistorted counterparts. For the best possible evaluation, these images all show the camera calibration chessboard, because its lines are straight and its corners are all the same size. Placing the chessboard perpendicular to the camera is important. Red rectangles framing the chessboard, with their own corners being placed at the outer corners of the chessboard, were added manually to illustrate the change.

(a) Original front fisheye image (b) Undistorted front image

Figure 4.1: The original fish eye and undistorted image of the front camera with a chessboard calibration pattern

The images in figure [4.1] are showing the original fisheye images captured by the front camera and its undistorted complement. At the very end of the front camera images, the vehicle license plate number and the case of the camera can be seen, thanks to the 190 deg FOV. As expected the corners in the original image [4.1a] appear straight in the center of the image, but when going away from the middle, the distortion is getting obvious. The outer chessboard corners are overlapping the red rectangle in the upper, right hand and mostly on the left-hand side of the chessboard. By contrast, the chessboard fits perfectly inside of the red rectangle in the undistorted image [4.1b], but this comes with the costs of a smaller FOV. The rectification itself causes this zoom-in effect, that makes the trees on the left and right side disappear. Same applies to the upper and lower areas of the original images. The calibration overall succeeded, as also can be seen by reference to the straightened railings on the left and right sides of the image.

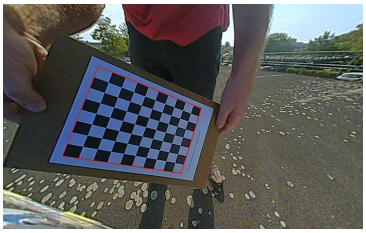
(a) Original rear fisheye image

(b) Undistorted rear image

Figure 4.2: The original fish eye and undistorted image of the rear camera with a chessboard calibration pattern

Analogical to the front cameras intrinsic results are the rear camera extrinsic results. The images do pretty much look alike since the vehicle was moved when capturing these images. The reason is that shadows should not influence the calibration process. At the lower end of the rear cameras picture the vehicle back bumper is visible. Again the chessboard of the original image [4.2a] does not fit into the later added red rectangle and the distortion is getting worse when moving away from the center of the image. Compared to the original image, the rectified one [4.2b] has only straight lines and a chessboard that fits perfect into the red rectangle. The railings and parking marks on the ground are not curvy anymore.

(a) Original left fisheye image

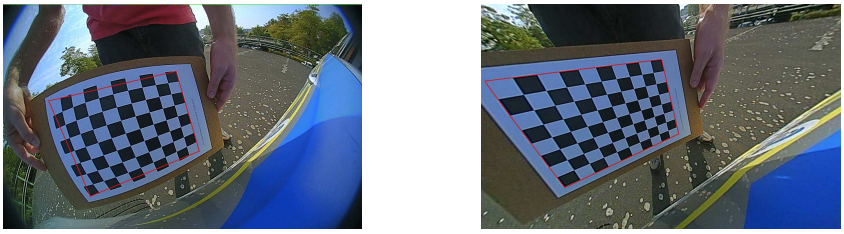


(b) Undistorted left image

Figure 4.3: The original fish eye and undistorted image of the left camera with a chessboard calibration pattern

Coming to the left camera, it is obvious, that the side cameras are not straight oriented like the front and back camera. That is why the horizon curvy and the complete lower left side of the *MIG* is visible in the original fisheye image of the left camera [4.3a]. The chessboard in this figure is placed in the cameras focus, so the upper and right side of the chessboard nearly match with the red rectangles upper and right side. Though when having

a closer look, the corners of the chessboard are distorted, especially on the left and lower side of it. All in all the chessboard does not fit inside the red rectangle and barely any line appears straight. In direct comparison to the rectified image [4.3b] there is a loss of FOV but every line is now straight and the red rectangle is nicely framing the chessboard.



(a) Original right fisheye image (b) Undistorted right image

Figure 4.4: The original fish eye and undistorted image of the right camera with a chessboard calibration pattern

Lastly, the images of the right camera are displayed in figure [4.4]. Just like in the original picture of the left camera the chassis of the vehicle is discernible, and all lines are not straight. Again it is obvious, that the rotation of the right camera differs to all the other cameras. This time, the chessboard is not in the center of the image. That is why the fisheye effect is most visible here and only the left side of the red rectangle matches with the chessboard. Despite the not placing the chessboard in the center, the undistorted version [4.4b] is looking fine. Every line that is straight in reality is also in the rectified version.
Concluding this subsection about the resulting intrinsic parameters, all of them are satisfying. The only lines that are still curved are the same ones like in reality. Unfortunately the disadvantage of the straight lines is the loss of the large FOV as can be seen in figure [2.16].

4.1.2 Extrinsic parameters

Verifying extrinsic parameters is different to the verification of the intrinsic parameters because it is possible to compare the calculated parameters with measured ones. The positions of the cameras were measured with a common yardstick and confirmed through the data already known about the *MIG*.
But the rotation is more difficult. Measurements were done with the Android app smart tools [Sma] regarding the roll and pitch angles. As for the pitch angles, there is another option, specific determining the angles between the horizon of an image and a straight line from the left side start of the horizon towards the right side of the same image height. This is only possible when the images are rectified to get straight lines to compare.

	x	y	z
Front	0.95 m	0.02 m	0.52 m
Left	-0.73 m	0.97 m	0.94 m
Rear	-3.68 m	0.03 m	0.825 m
Right	-0.78 m	-0.95 m	0.965 m

Figure 4.5: Calculated translation vector

	x	y	z
Front	0.93 m	0 m	0.52 m
Left	-0.72 m	0.935 m	0.915 m
Rear	-3.81 m	0.01 m	0.84 m
Right	-0.73 m	-0.94 m	0.92 m

Figure 4.6: Measured translation vector

Table [4.5] contains the calculated translation vector respectively the cameras position. All translation values are given in meters. A point directly on the ground under the front centered axle of the *MIG* is the point of origin, just like in the ego vehicle coordinate system [2.2.2]. The creators of CamOdoCal explained, that the height calculation is a weakness of their framework. Since the height of the front camera is the reference height and has to be entered before the calculation, the measured and calculated value is exactly the same. The heights of the other cameras differ between 1.5 and 4.3 centimeters or 1.7 % up to 4.7%. X-axis values deviate between 2 for the front camera as the point of origin and 13 centimeters for the right and rear camera which is a divergence of 2.4% for the left up to 3.4% for the right camera. Lastly the y-axis value. Here the front and rear cameras have a gap of 2 centimeters and for the left and right camera, it is 3.5 respectively 1 centimeters which is a maximum of 3.7% for the left camera. Overall the calculated translation vector, meaning the cameras positions, are very close to reality with only a few centimeters variance.

In table [4.7[and table [4.8] the calculated respectively measured roll pitch and yaw angles are exhibited, describing the cameras rotations. When the pitch angle is zero, it means that the horizon of the images the camera captured is a line with the same height pixels all over. When looking at the images the cameras created, this angle is expected to be the smallest, with the right camera to have the highest value. Here a roll angle of $-90°$ means, that the camera is neither looking up or down, so the left and right cameras are looking downwards. The value $-90°$ is the initial value because of outstanding transformation. Where the cameras are looking is illustrated by the yaw-angle. If two cameras would look in absolute opposite directions,

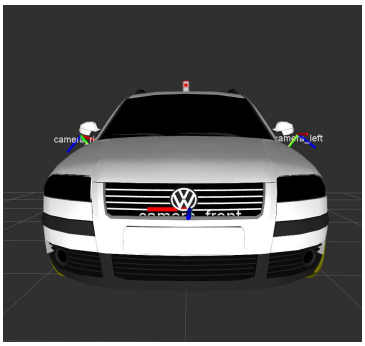
	Roll	Pitch	Yaw
Front	0.4°	−91.26°	−91.73°
Left	1.15°	−126.18°	6.6°
Rear	4.28°	−90.3°	85.85°
Right	11.14°	−138.24°	154.86°

Figure 4.7: Calculated roll pitch and yaw angles

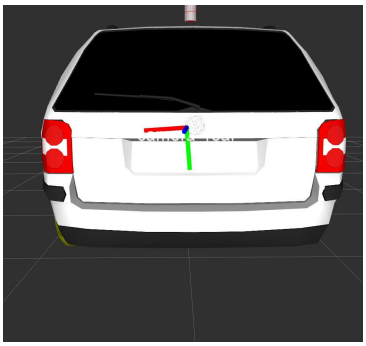
	Roll	Pitch	Yaw
Front	0.05°	−90°	−90°
Left	2.75°	−123°	10°
Rear	4.22°	−90°	90°
Right	17.26°	−135°	155°

Figure 4.8: Measured roll pitch and yaw angles

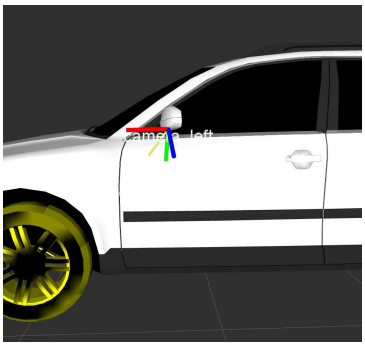
the yaw angle between them would be 180°, so when covering all 4 directions the difference between the front and left or the rear and right and so on, would be 90°.
The calculated and measured results are very close. Yaw angles have the largest variance with 1.73° for the front, 3.4° for the left, 4.15° for rear and 0.14° for the right camera. In average this is a difference of 2.355°. As for the roll angles, the gap is 0.35° for the front, 1.6° for the left, 0.06° for the rear and a large mismatch of 6.12° for the right camera, resulting in an average of 2.0325°. Lastly the pitch angles. The disparity of the front camera is 1.26°, of the left camera 3.18°, of the rear camera 0.3° and of the right camera, it is 3.24°. Consequent this is an average of 1.95°. In total the mean mismatch between the calculated and measured rotation angles is 2.0325° for the row angles, 1.95° for the pitch angles and 2.355° for the yaw angle.
Figure [4.9] shows a screenshot of the ROS 3D-visualization tool *RVIZ*, containing an illustration of the car model and a visualization of the extrinsic parameters through the transformation package. Although the height values were confirmed through measurements, in this representation they had to be increased by 10 centimeters. All other values were untouched in this figure.
At last, the surround view is being discussed. Implementations section [3.5] explains, that for each camera a top view is generated applying both the intrinsic and extrinsic camera parameters and afterwards the results were simply copied into one image. The driven path of the used extrinsic rosbag is displayed in figure [3.2]. A top level of a parking level was the used environment, which can roughly be recognized through the intrinsic calibration images [4.1]. No pedestrians disturbed the recording. Despite there is no



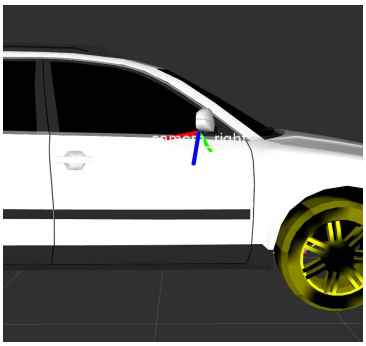
(a) Rviz screenshot front camera



(b) Rviz screenshot rear camera



(c) Rviz screenshot left camera

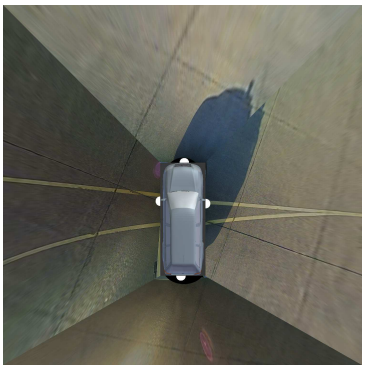


(d) Rviz screenshot right camera

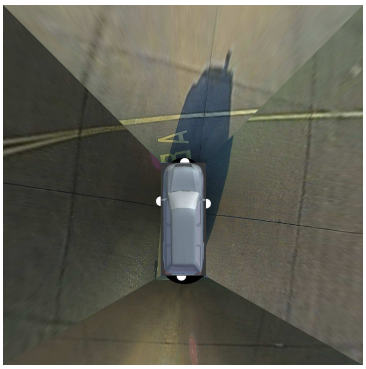
Figure 4.9: Rviz screenshots of extrinsic camera parameters using ROS transformation package

further optimization, the resulting images in figure [4.10] are promising. The white semicircles represent the cameras positions and the black regions are areas the cameras are not able to see, because of the bumper blocking the field of view. Thanks to the intrinsic parameters the lines on the ground of the Tempelhofer Feld are straight. An advantage of using this area is the guaranteed flat level because it is near a former airport. No color calibration has been done in this thesis, explaining the differences in brightness or contrast. In subfigures [4.10a] and [4.10d] it looks like the lines from the left camera continue correctly through the vehicle into the right camera. But the transitions between the cameras with an overlapping field of view are not ideal, especially for the rear camera as can be seen in image [4.10c]. The transition from the front camera towards the left or right camera is looking better, but still not perfect. Figure [4.10b] displays this behaviour. It seems that the camera is placed a little bit too much up front. Overall the presented surround view is a good initial position but still needs some

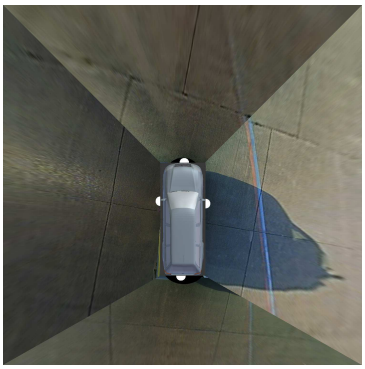
improvements.



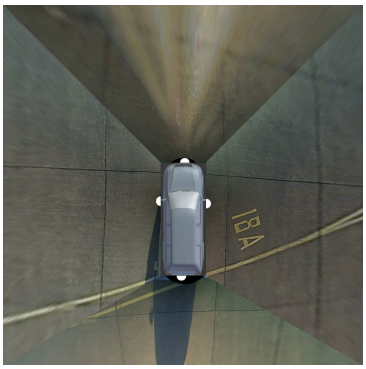
(a) Surround view 1



(b) Surround view 2



(c) Surround view 3



(d) Surround view 4

Figure 4.10: Set of surround view images

4.1.3 Performance of the implementation

After discussing the results in terms of the calculated camera parameter, the upcoming subsection is about how well the implemented ROS nodes and the calibration script are running. Two computers were used, both had Ubuntu 14.04, ROS indigo and besides that the newest updates installed. Benchmarks were made with live data using the Ubuntu command *top* respectively the command *rostopic hz <topic name>* while being connected to the *MIG* and having access to the camera system. The mainly used laptop which was running the ROS nodes has the following hardware:

- Model: Dell XPS 15 9550
- CPU: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz (4 Cores, 8 Threads)

- GPU: Nvidia Geforce 960m (2GB)
- SSD: Samsung PM951 NVMe 512GB
- RAM: 16 GB

In table [4.11] the published frames per second alongside with the CPU usage can be seen. While the camera driver performs very well with a very low CPU usage of about one percent and delivering the full 30 frames per second the cameras are providing; the decompression has a higher CPU demand of between 20% and 30% for each camera. So with this hardware setup, one out of 8 threads was nearly completely busy with the decompression of all cameras compressed JPEG images. Still, the decompression node is capable publishing the decompressed JPEG images with the full frame rate of 30. A much higher CPU load of between sixty and seventy percent is generated by the rectification nodes. It is not a surprise when remembering that there are thirty images every second with a resolution of 1280 in width and 800 in height and every pixel has to be moved to its new position. Despite this, the ROS node is able to publish nearly the full rate of 30 rectified images, being just a little bit below it with about an average of 29 frames per second. The 240% CPU load, meaning that more than two out of eight available threads were completely busy, generated by the surround view makes sense since there are four images rectified like by the rectification node and combined to each other. Unfortunately, the surround view does not reach the maximum possible frame rate. Most probably it is because the cameras are not synchronized and the ROS node waits for new images of every camera before publishing a new surround view. The overall RAM usage of the nodes is very low as each node has a maximum demand of one percent of the available 16 GB.

ROS node	Frames per second	CPU usage	RAM usage
Driver	30	0.7% - 1.3%	0.7% - 1% each
Decompression	30	20% - 30% each	0.1% - 0.2% each
Rectification	29	60% - 70% each	0.3% - 0.4% each
Surround view	24	240%	0.7% - 1%

Figure 4.11: Performance of the implemented ROS nodes

Next is the camera calibration script, which ran on the following hardware:

- Mode: Dell Precision T1600
- CPU: Intel(R) Xeon(R) CPU E31245 @ 3.30GHz (4 cores, 8 Threads)
- GPU: Nvidia Quadro 2000
- HDD: Seagate ST3500413AS 500GB

- RAM: 10 GB

Since as part of the Docker container building process software is installed like OpenCV [2.5.2] and out of every needed rosbag the image and odometry files are extracted and added to the container, there is a huge need for disk space. There were four rosbags used only for the intrinsic camera calibration, each of the size of about 500 MB. The extrinsic rosbag was 2.8 GB large, from which 500 images were used of each camera, leading to a total of 2.9 GB image and odometry files. All in all the created docker container had a size of 10.9 GB. But this was before even starting the calculations. During these, CamOdoCal [2.4.2] combines many images to search for matching features and to look for correlations. This is why the overall container size rises up to 77 GB. Depending on the calculation steps, the CPU usage alternates between 100% and 750%, mostly around 700% after the initialization steps, and the RAM usage raises up to 65%. In total, the camera calibration process with building the whole Docker container took 3 hours and 43 minutes when using this camera calibration script.

4.2 Conclusion

Before the start of this thesis, the BroadR-Reach fisheye camera system was only installed in the autonomous car *MIG*. There was no possibility to gain images out of the cameras, and their positions and orientations were unknown.
So the first goal of this thesis was, to write a driver for the camera system which feeds the used ROS-framework with the recorded images of the cameras. This has been realized with the full available performance of thirty frames per second. Preventing every single ROS node accessing the provided compressed JPEG images to decompress, a separate ROS node only for this purpose was implemented, increasing the overall performance.
Due to the fact, that the installed cameras are equipped with fisheye lenses, their images distortions had to be removed to get straight lines, necessary for computer vision algorithms. Therefore an intrinsic calibration was needed, resulting in an easy-to-use camera calibration script. This script is based on the OpenCV intrinsic camera calibration and CamOdoCal, an open source pipeline based on Visual Odometry and SLAM. Only for each camera, one recording of images showing a calibration pattern is required to gain precise intrinsic camera parameters. Besides that, if the script is fed with a recording of images and odometry data while driving curves in a rich textured environment, the cameras positions and rotations are calculated.
Using the new calculated intrinsic parameters, the distortions of the original fisheye images were removed by a separate ROS node while still being able to process at twenty-nine out of the the full available thirty frames per second.

The calculated extrinsic parameters were verified through measurements made by hand. Both, the intrinsic and extrinsic parameters were applied to create single top views of each camera, which then were combined to a surround view of the *MIG*.
Overall the final outcome of this thesis is a promising base for further works using the installed fisheye camera system, now that it is completely accessible and proper intrinsic and extrinsic camera calibration has been done. This work can be seen as a starting point on top of which most diverse computer vision algorithms could grow.

4.3 Future work

Now, with a properly calibrated camera system, there are many possibilities to make use of it. With an improved surround view parking lots could be recognized like in these works [LW12] [HKH+13]. Another option is the detection of traffic lights [dCN09]. For the safety of surround people a pedestrian detection is an important field presented in several works like [OPS+97], [DWSP12] or [EG09].
Currently, the implementation of this thesis only makes use of the CPU. If the power of graphical process units were used by applying specific API's like CUDA, huge performance improvements are possible. CamOdoCal [2.4.2] already provides this option and a specific base Docker container is also available at Github [Docb].
The presented implementation of the surround view is very simple. As already mentioned, every single camera's top view is just copied to its new position of the surround view. The resulting images reveal an unhandled problem - pending color calibration. This is visible in figure [4.10] because there are huge differences in terms of brightness and contrast between the recorded images. The work [ZAP+14] uses photometric alignment to correct the brightness and color mismatches between adjacent views to achieve seamless stitching. Therefore a global color and brightness correction function for each view is designed. This way the discrepancies in the overlapping regions of adjacent views are minimized. Also, the transition between the single top views is optimized via a loop-up-table which is created after initially running a feature matching algorithm. Altogether, this work demonstrates promising ways to optimize the existing implementation of the surround view.

Bibliography

[AM12] Sameer Agarwal and Keir Mierle. Ceres solver: Tutorial & reference. *Google Inc*, 2:72, 2012.

[Aut] AutonomousLabs. `http://autonomos-labs.com/team/`. Accessed: 2017-04-15.

[BA01] Joao P Barreto and Helder Araujo. Issues on the geometry of central catadioptric image formation. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2001.

[BK13] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2nd edition, 2013.

[BL14] Stuart Bennett and Joan Lasenby. Chess–quick and robust detection of chess-board features. *Computer Vision and Image Understanding*, 118:197–210, 2014.

[Bou15] Jean-Yves Bouguet. Matlab calibration tool, 2015.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[CAD11] Gerardo Carrera, Adrien Angeli, and Andrew J Davison. Slam-based automatic extrinsic calibration of a multi-camera rig. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2652–2659. IEEE, 2011.

[Cam] CamOdoCalGithub. `https://github.com/hengli/camodocal`. Accessed: 2017-04-15.

[dCN09] Raoul de Charette and Fawzi Nashashibi. Real time visual traffic lights recognition based on spot light detection and adaptive traffic lights templates. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 358–363. IEEE, 2009.

[Dis] Distortion. `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`. Accessed: 2017-04-14.

[Doca] Docker. `https://www.docker.com/what-docker`. Accessed: 2017-04-11.

[Docb] DockerCUDA. `https://github.com/NVIDIA/nvidia-docker`. Accessed: 2017-04-25.

[DRMS07] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6), 2007.

[DWSP12] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):743–761, 2012.

[EG09] Markus Enzweiler and Dariu M Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE transactions on pattern analysis and machine intelligence*, 31(12):2179–2195, 2009.

[Ego] EgoVehicleBlueprint. `http://car-blueprints.narod.ru/images/skoda/skoda-fabia-combi.gif`. Accessed: 2017-04-11.

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[FMGJRA14] Eduardo Fernandez-Moral, Javier González-Jiménez, Patrick Rives, and Vicente Arévalo. Extrinsic calibration of a set of range cameras in 5 seconds without pattern. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 429–435. IEEE, 2014.

[GLT12] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[GMR12] Chao X Guo, Faraz M Mirzaei, and Stergios I Roumeliotis. An analytical least-squares solution to the odometer-camera extrinsic calibration problem. In *Robotics and Au-*

tomation (ICRA), 2012 IEEE International Conference on, pages 3962–3968. IEEE, 2012.

[Gro] GrowthImageSensors. `http://www.sensorsmag.com/components/significant-growth-potential-for-image-sensors-automotive-market`. Accessed: 2017-05-01.

[Gup89] KC Gupta. An historical note on finite rotations. *Journal of Applied Mechanics*, 56:139, 1989.

[GWSG11] Daniel Göhring, Miao Wang, Michael Schnürmacher, and Tinosch Ganjineh. Radar/lidar sensor fusion for car-following on highways. In *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*, pages 407–412. IEEE, 2011.

[HD95] Radu Horaud and Fadi Dornaika. Hand-eye calibration. *The international journal of robotics research*, 14(3):195–210, 1995.

[HFP15] Lionel Heng, Paul Furgale, and Marc Pollefeys. Leveraging image-based localization for infrastructure-based calibration of a multi-camera rig. *Journal of Field Robotics*, 32(5):775–802, 2015.

[HKH+13] Sebastian Houben, Matthias Komar, Andree Hohm, Stefan Luke, Marcel Neuhausen, and Marc Schlipsing. On-vehicle video-based parking lot recognition with fisheye optics. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 7–12. IEEE, 2013.

[HLP13] Lionel Heng, Bo Li, and Marc Pollefeys. Camodocal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1793–1800. IEEE, 2013.

[Hof13] Peter Hofmann. Object Detection and Tracking with Side Cameras and RADAR in an Automotive Context. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2013.

[Hor87] Berthold KP Horn. Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, 4(4):629–642, 1987.

[K+99] Jack B Kuipers et al. *Quaternions and rotation sequences*, volume 66. Princeton university press Princeton, 1999.

[KB06] Juho Kannala and Sami S Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1335–1340, 2006.

[KRC+11] Bernd Manfred Kitt, Joern Rehder, Andrew D Chambers, Miriam Schonbein, Henning Lategahn, and Sanjiv Singh. Monocular visual odometry using a planar road model to solve scale ambiguity. 2011.

[liba] libJPEG. `http://libjpeg.sourceforge.net/`. Accessed: 2017-04-15.

[Libb] LibJPEG-turbo. `http://libjpeg-turbo.virtualgl.org/`. Accessed: 2017-04-11.

[Libc] LibPCAP. `http://www.tcpdump.org/`. Accessed: 2017-04-11.

[Loo] LoopClosureDetection. `http://cogrob.ensta-paristech.fr/loopclosure.html`. Accessed: 2017-05-03.

[LW12] Chien-Chuan Lin and Ming-Shi Wang. A vision based top-view transformation model for a vehicle parking assistant. *Sensors*, 12(4):4431–4446, 2012.

[Mat] Matlab. `https://www.mathworks.com/products/matlab.html`. Accessed: 2017-04-11.

[MBLB91] Hanspeter A Mallot, Heinrich H Bülthoff, JJ Little, and Stefan Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological cybernetics*, 64(3):177–185, 1991.

[Med] MediaGateway. `http://www.technica-engineering.de/produkte/media-gateway/`. Accessed: 2017-04-27.

[Meia] MeiCalibrationToolbox. `http://www-sop.inria.fr/icare/personnel/Christopher.Mei/ChristopherMeiPhDStudentToolbox.html`. Accessed: 2017-04-15.

[Meib] MeiProjectionModel. `http://www.robots.ox.ac.uk/~cmei/articles/projection_model.pdf`. Accessed: 2017-04-15.

[Mid14] LNM Middelplaats. Automatic extrinsic calibration and workspace mapping: Algorithms to shorten the setup time of camera-guided industrial robots. 2014.

[MK04] Gerard Medioni and Sing Bing Kang. *Emerging topics in computer vision*. Prentice Hall PTR, 2004.

[Neu16] D. Neumann. Fahrzeugerfassung mithilfe von Stereo-Vision sowie HOG-, LBP-, und Haar-Feature-basierten Bildklassifikatoren. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2016.

[OCa] OCamCalib. `https://sites.google.com/site/scarabotix/ocamcalib-toolbox`. Accessed: 2017-04-15.

[Omn] OmniVision. `http://www.ovt.com/sensors/OV10635`. Accessed: 2017-04-11.

[Opea] OpenCV. `http://opencv.org/`. Accessed: 2017-04-11.

[Opeb] OpenCVFisheye. `http://docs.opencv.org/3.1.0/db/d58/group__calib3d__fisheye.html`. Accessed: 2017-04-30.

[Opec] OpenCVFishZoom. `http://answers.opencv.org/question/36032/how-to-draw-inscribed-rectangle-in-fish-eye-corrected-image-using-opencv/`. Accessed: 2017-04-30.

[Oped] OpenCVGithub. `https://github.com/opencv/opencv`. Accessed: 2017-04-26.

[OPS+97] Michael Oren, Constantine Papageorgiou, Pawan Sinha, Edgar Osuna, and Tomaso Poggio. Pedestrian detection using wavelet templates. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 193–199. IEEE, 1997.

[OSV08] Anton Obukhov, Konstantin Strelnikov, and Dmitriy Vatolin. Fully automatic ptz camera calibration method. In *Proc. of Graphicon*, pages 122–127, 2008.

[pat] patternAsymCircles. `http://docs.opencv.org/2.4/_downloads/acircles_pattern.png`. Accessed: 2017-04-15.

[Rak16] Sindre Raknes. 3d robot vision using multiple cameras. Master's thesis, NTNU, 2016.

[Rol] RollPitchYaw. `https://www.grc.nasa.gov/WWW/K-12/airplane/rotations`. Accessed: 2017-04-13.

[Rot14] Simon Sebastian Rotter. FSwarm Behaviour for Path Planning. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2014.

[Sca07] Davide Scaramuzza. *Omnidirectional vision: from calibration to robot motion estimation*. PhD thesis, ETH Zurich, 2007.

[Sch] Florian Michael Schaukowitsch. Pano .net.

[Sma] SmartToolsAndroid. `https://play.google.com/store/apps/details?id=kr.aboy.tools&hl=de`. Accessed: 2017-04-24.

[SMS06] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A toolbox for easily calibrating omnidirectional cameras. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5695–5701. IEEE, 2006.

[Söd15] Anna Söderroos. Fisheye camera calibration and image stitching for automotive applications, 2015.

[Sze10] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[Tec] TechnicaEngineering. `http://shop.technica-engineering.de/index.php?id_product=18&controller=product`. Accessed: 2017-04-11.

[TMHF99] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment?a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[TNM12] Kosuke Takahashi, Shohei Nobuhara, and Takashi Matsuyama. A new mirror-based extrinsic camera calibration using an orthogonality constraint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1051–1058. IEEE, 2012.

[TOJG10] Shane Tuohy, D O'cualain, E Jones, and M Glavin. Distance determination for an automobile environment using inverse perspective mapping in opencv. In *Signals and Systems Conference (ISSC 2010), IET Irish*, pages 100–105. IET, 2010.

[TR13] Ernesto Tapia and Rojas Raúl. A note on calibration of video cameras for autonomous vehicles with optical flow. 2013.

[VBPG14] SM Vaitheeswaran, MK Bharath, Ashish Prasad, and M Gokul. Leader follower formation control of ground vehicles using dynamic pixel count and inverse perspective mapping. *International Journal of Multimedia and its Applications*, 6(5):1–11, 2014.

[Win] WinPCAP. `https://www.winpcap.org/`. Accessed: 2017-04-27.

[Wir] Wireshark. `https://www.wireshark.org/`. Accessed: 2017-04-11.

[ZAP+14] Buyue Zhang, Vikram Appia, Ibrahim Pekkucuksen, Yucheng Liu, Aziz Umit Batur, Pavan Shastry, Stanley Liu, Shiju Sivasankaran, and Kedar Chitnis. A surround view camera solution for embedded systems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 662–667, 2014.

[Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.