Evelyn Gutschier, BSc.

# Initial 6-DOF Camera Pose Estimation from known 3D Geometric Primitives

**MASTER'S THESIS**

to achieve the university degree of
Diplom-Ingenieurin

Master's degree programme
Computer Science

submitted to

**Graz University of Technology**

Advisor

Dipl.-Ing. Dr.techn. Clemens Arth
Institute for Computer Graphics and Vision

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg
Institute for Computer Graphics and Vision

Graz, Austria, Feb. 2019

## AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

# Abstract

For Augmented Reality, 6 DOF pose estimation is an essential prerequisite. This work focuses on the task of monocular optical tracking of known objects, based on the technique of natural feature detection. An object's shape, measures and texture are being required as a priori knowledge. We compare two common approaches to retrieve the pose: homographies, which let us retrieve a pose from 2D-2D point correspondences, and Perspective-$n$-Point (PnP), which works on 2D-3D point correspondences. We then investigate the most convenient ways to refine the pose: we compare the use of an optimization library, Ceres Solver, with own implementations of Gauss-Newton method and Levenberg-Marquardt method.

Even though homographies are not as versatile as PnP, they show better performance in general and higher accuracy for coplanar scenes. While the Ceres Solver library functions perform well, our own optimization functions turn out as 10 times more efficient for our purpose while achieving the same accuracy.

**Keywords:** Pose Estimation, Homography, P3P, Pose Refinement

# Kurzfassung

Für Augmented Reality ist eine Lagebestimmung über sechs Freiheitsgrade eine notwendige Voraussetzung. Diese Arbeit konzentriert sich auf die Aufgabe des monokularen optischen Trackings bereits bekannter Objekte mittels Detektion natürlicher Merkmale. Die Form eines Objekts, seine Maße und Textur müssen vorher bekannt sein. Wir vergleichen zwei verbreitete Ansätze um die Lage zu bestimmen: Homographien, durch welche wir die Lage via 2D-2D Punktkorrespondenzen bestimmen können, und Perspective-$n$-Point (PnP), das 2D-3D Punktkorrespondenzen erfordert. Wir untersuchen die günstigsten Methoden um die Lagebestimmung zu verbessern: wir vergleichen die Verwendung der Ceres Solver Optimierungsbibliothek mit eigenen Implementierungen des Gauss-Newton- und Levenberg-Marquardt-Verfahrens.

Obwohl Homographien nicht so vielfältig einsetzbar wie PnP sind, zeigen sie im Allgemeinen eine bessere Laufzeit und höhere Genauigkeit für planare Szenen. Während die Ceres Solver Programmbibliothek eine gute Laufzeit aufweist, stellen sich unsere eigenen Implementierungen bei gleicher Genauigkeit als zehnmal effizienter für unsere Zwecke heraus.

**Stichworte:** Lagebestimmung, Homographie, P3P, Lageverbesserung

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Tracking in AR

In the beginning of Augmented Reality (AR), tracking was done using mechanical tracking or GPS, which suffered from limited practicability or accuracy. A major breakthrough was done by the introduction of visual markers: this cleared the path for vision-based tracking, which is now of more importance than ever since cameras are cheap and available on almost any device. Vision-based tracking is an essential task in robotics, Computer Vision and Computer Graphics, especially in Virtual Reality (VR) and AR: it is used for inside out tracking (determining the camera/tracker's position in a static environment) as well as outside in tracking (determining the position of an object in relation to the static camera/tracker). While markers are a convenient solution, it is not always feasible to use them, as every environment would need to be prepared first, their visibility is not wanted or it is simply not possible to prepare the environment, *e.g.* when navigating through a city. Therefore, modern approaches focus on natural feature tracking. Natural features include points, lines or texture, while the vast majority of algorithms are based on point-based features due to their stability and repeatability.

A big new challenge arose with the popularity of mobile systems: although they have limited performance, they became an attractive target for AR applications since they are widely available, relatively inexpensive and offer a new small form factor. While many mobile systems provide as well an Inertial Measurement Unit (IMU), inertial sensors have several shortcomings: they cannot be used to track moving objects other than the device itself, their measurements are often noisy and inaccurate, and they are prone to drift. Although they can not be used for tracking on their own, they can still be useful to refine or improve vision-based tracking in certain situations or overcome tracking loss for inside out tracking. The combination is known as *Sensor Fusion*. The first ones to introduce a purely vision-based real-time 6 Degrees of Freedom (DOF) natural feature tracking system on mobile phones were Wagner et al.in 2008 [77], who used a stripped-down version of

Scale-Invariant Feature Transform (SIFT)[40] features and Ferns [55] to track planar targets. While at this point they did not take into account the underlying geometry of objects, in 2009 they released a video where they are extending their method to cylinders [75].

## 1.2 Pose Estimation

For solving the tracking problem in general, the ultimate goal is to fully describe our camera with respect to a local environment or a specific target over a sequence of frames. Physical cameras are most often modelled by a pinhole camera model, which describes a perspective projection between 3D points $X \in \mathbb{R}^3$ and 2D image points $x \in \mathbb{R}^2$ by casting a ray from $X$ towards the center of projection $C$ (through which all projection rays go), $x$ is where the ray hits the image plane, as seen in Figure 1.1.

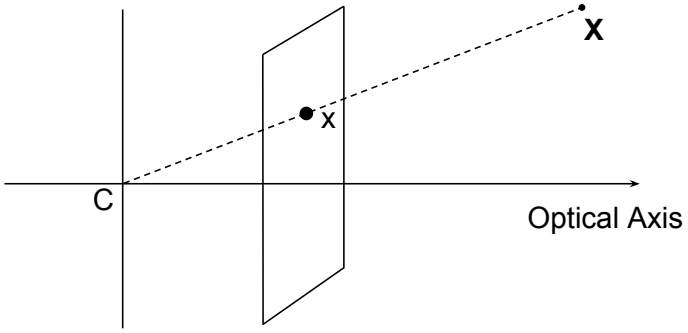



**Figure 1.1:** A pinhole camera projects 3D point X onto image point x at the point where the viewing ray of X hits the image plane.

The camera model can be described by a 3x4 matrix P, which projects homogeneous 3D world coordinates $\mathbf{X} = [x, y, z, 1]^T \in \mathbb{P}^3$ onto homogeneous 2D image coordinates $\mathbf{x} = [u, v, 1]^T \in \mathbb{P}^2$:
Hereby the matrix P has 11 DOF and can be split into intrinsic camera parameters $K$ and extrinsic camera parameters $R$ and $t$:

$$\mathbf{x} = K \cdot [R \mid t] \cdot \mathbf{X}$$

The intrinsic camera matrix $K$ follows the form

$$K = \begin{pmatrix} p_x & s & u_0 \\ 0 & p_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

where $(u_0, v_0, 1)^T$ marks the principal point, where the z-axis of the camera intersects the image plane, and $p_x$ and $p_y$ describe the ratio between the camera lens' focal length (the

distance between the camera center and the image plane) and the width and height of a pixel. The skew parameter s is only used when the optical axes are not orthogonal and is zero for most modern cameras.
Assuming the intrinsic parameters are already known (or calculated beforehand via camera calibration), we are now interested in the extrinsic parameters: they consist of a 6 DOF camera pose, 3 DOF representing the camera's position in world space and 3 DOF representing its orientation (yaw, pitch, roll). They are represented by a rotation matrix $R$ and a translation vector $t$ and express the coordinate transformation from 3D world space to 2D projective space, as seen in Figure 1.2. Note hereby that the matrix $[R \mid t]$ can as well be split into a projection matrix $\Pi$ and a quadratic transformation matrix $T$:

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$T = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

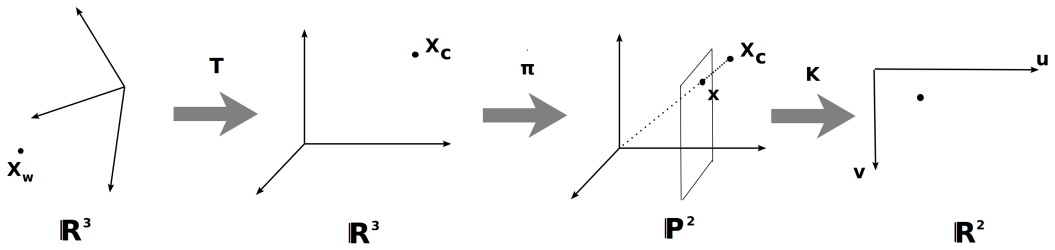



**Figure 1.2:** Geometric interpretation of a camera matrix P. While $R$ and $t$ are used to translate world space coordinates to camera space coordinates, the perspective projection translates the coordinates into projective space and the matrix $K$ translates them into actual 2D coordinates within a camera frame.

**Prerequisites for Pose Calculus** A lower bound of how many correspondences are needed to calculate P is given by the number of DOF we want to solve for: while Direct Linear Transformation (DLT) solves for all elements of the matrix and therefore requires at least six correspondences (each correspondence contributing with two equations), Perspective-$n$-Point (PnP) solutions exist for as few as three 2D-3D point correspondences, with a fourth correspondence being needed to resolve ambiguity issues[1]. The

[1]For the remainder of this thesis, we will deal with point correspondences only. For a more extended list of line and plane-based constraints, the interested reader is referred to the work of Ramalingam et al.[59]

basic assumption in determining the pose of a camera with respect to the environment given established 2D-3D correspondences is, that from the set of correspondences, only a hand full are correct. Therefore, any pose estimation algorithm has to be wrapped into a robust outlier detection technique, i.e. Random Sample Consensus (RANSAC)[18] or similar algorithms. In order to deal with noise in the data, common approaches can be separated into closed-form solutions that take into account all point correspondences at once in order to minimize the error, and iterative solutions, where the error is reduced with each iteration.

## 1.3 Thesis Goals

In this work we followed the approach of Wagner et al. [80] in determining the camera pose with respect to a known target, but we additionally took into account the underlying geometry of objects, as many real-life objects resemble primitive geometric shapes. We specifically consider cuboids, cylinders and spheres to represent different groups of shapes. Knowing the shape of an object, its 2D template and measurements (e.g. length, height, width) in advance, we can map features found on the template into 3D.

In *tracking by detection*, pose estimation is done repeatedly for each camera frame independently. This can become very difficult and computationally expensive on large amounts of natural features, *e.g.* outdoors. In order to lower the computational costs, this task can be split into *initial pose detection* and *incremental tracking* in subsequent frames based on a previous pose [80]. Incremental tracking takes into account the expected position or movement of the camera or object using the calculated camera pose of a prior frame. We assume the incremental tracking to be easier to solve and to be computationally less expensive than detection. Therefore, in this work we only focus on the initial pose detection problem. Even this problem is typically split into two steps, the calculation of an *initial pose estimate* followed by further *pose refinement*.

For the initial pose estimate, we use SIFT features on both the textured templates of the geometric primitives and on the camera frames to receive 2D-3D point correspondences. Depending on the (given) type of shape we then calculate the pose either via homographies, if the geometric primitives consists of mostly planar sides (*i.e.* cuboids), or via PnP methods if few to no planar sides are available (*i.e.* cylinders and spheres). In practical applications we will have to deal with outliers and noise. In order to deal with noise, the problem is formulated as a non-linear least squares problem. Common approaches to solve them include the Gauss-Newton and Levenberg-Marquardt method, Levenberg-Marquardt being a linear combination of the Gradient Descent method and Gauss-Newton method. Since these methods cannot offer global convergence, a good first pose estimation is already needed. So even though we can remove outliers to a certain extent in the optimization, it is preferable to do so at the first pose estimate

already. Both homographies and PnP methods therefore need to be wrapped into a RANSAC scheme [18] to deal with the outliers.

For the pose refinement, powerful open source libraries are available: We used Ceres solver, an open source C++ library for solving various optimization problems, including non-linear least squares problems [1]. One of its many features is automatic differentiation, which can be valuable for software prototyping. However, in doubt of the capabilities of its automatic differentiation feature, we also calculate the derivatives ourselves to use them in the analytic derivative feature of Ceres Solver, and further implement a Gauss Newton function to compare it in terms of performance to the one derived by the library. Finally, we also look into Lie groups hoping to take a "shortcut" on calculating the tedious actual derivatives.

As a result of our approach, we show that an own implementation of either Gauss-Newton or Levenberg-Marquardt outperforms the Ceres Solver library functions by a factor of 10, and that using textured templates of geometric primitives is reasonable. We give several exemplary results of the developed system and provide proof of its plausibility. We conclude by bringing our approach to mobile systems, namely to Android on a Samsung Galaxy S7 mobile phone, in order to test its practicability.

## 1.4 Outline

In Chapter 2 we discuss common tracking methods, alternatives to monocular optical tracking, their advantages and disadvantages, their practicability for tracking on mobile devices and mention some approaches for each. We also go into more detail for some of the underlying concepts. Chapter 3 first describes our prerequisites and how we generate our templates. We then describe our method in detail, first how we get an initial pose estimate in Section 3.2 and then how we further optimize it in Section 3.3. In Chapter 4 we are showing what test data we generated and used the results of our evaluations, while in Chapter 5 we present our practical results on mobile systems. In Chapter 6 we are drawing our conclusion of the former chapters.

# 2 Related Work

In the following, we give an overview about related work in field of camera pose estimation and tracking. For brevity, we will mainly focus on the aspect of pose initialization, as this is most relevant for our work.

## 2.1 Various Tracking Hardware

While in general there exist various types of tracking systems, most of them are only feasible for active tracking and require either the signal emitter or the sensor to be mounted on the target:

- Mechanical Tracking: It uses robotic armatures where the position can be calculated from known lengths and angles at the joints. While its accuracy is rather high, its working space is limited, the devices are usually expensive and encumber the movement of the user [53]. An example device is shown in Figure 2.1a.

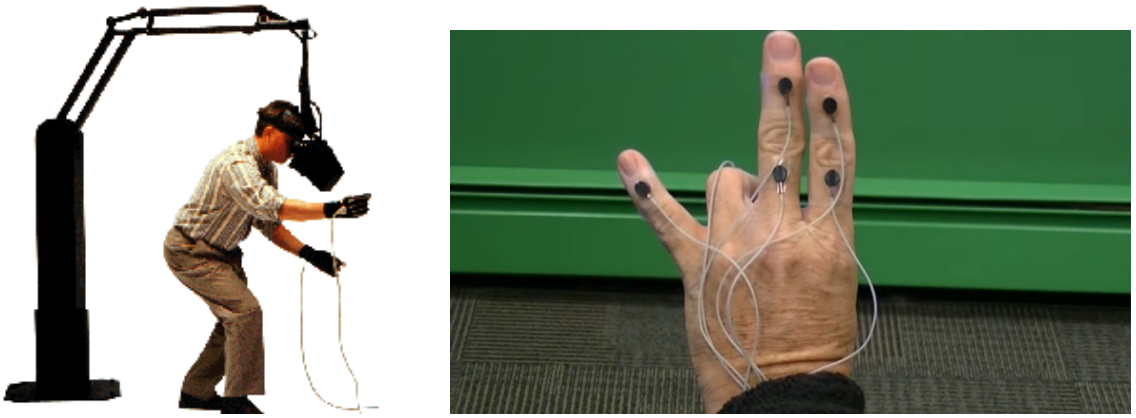
- Magnetic Tracking: Magnetic fields are used to determine the position. They do not need a direct line of sight, have a high update rate and can track several targets at once, but they are susceptible to environment noise: metallic objects in the surrounding can cause interference and its working volume is rather small too. Figure 2.1b shows finger tracking using magnetic tracking.

- Ultrasonic Tracking: High frequency sound waves are used for either time-of-flight or to measure the phase coherence. The time-of-flight method's update rate is limited by the low speed of sound and is affected by factors like temperature or pressure, while the coherence based one can accumulate error over time.

- Global Positioning System (GPS): First used for an AR system in 1997 [15], GPS tracking has since been used in many applications, especially for outdoor tracking [35]. However, it still suffers from low accuracy and is therefore mostly used in

hybrid tracking systems. More accurate GPS systems with an accuracy of 5 cm are theoretically available, but are highly expensive.

- Inertial Tracking: While using gyroscopes and accelerometers offer a large working space and do not require any line of sight, gyroscopes are sensitive to vibration [53] and both suffer from drift in time. Inertial sensors cannot be used for initialization of the pose as they only measure the differences, so they need to be combined with other methods like optical trackers or ultrasonic trackers.



**(a)** Mechanical Tracking **(b)** Magnetic Tracking

**Figure 2.1:** A binocular omni-orientation monitor (BOOM) [a] and a magnetic finger tracker by Polhemus [58].

[a]Image from http://www8.informatik.umu.se/ jworth/2Techniques

**Hybrid Tracking** There have been various approaches of *hybrid tracking*, also known as *sensor fusion*, which combine optical tracking with other tracking types: e.g. combinations of optical tracking and GPS [61], which focus on outdoor tracking and help to overcome signal loss of GPS in case of occlusions or being in buildings, or combinations with inertial sensors [50] [84].

## 2.2 Optical Tracking

Not only is optical tracking with RGB cameras able to provide higher accuracy than other approaches, but the required hardware is also considerably cheap. Since most AR devices need visual video background, some camera is usually required already. For Augmented Reality (AR) on mobile devices the most preferred approach is using a single camera, which opens access to AR for almost every modern mobile device.

**Infrared Tracking** Infrared (IR) tracking can either be done via retro-reflective markers, structured light [67] or by time-of-flight sensors [24]. IR cameras became quite affordable in the past years, like the Microsoft Kinect [86], which uses structured light and led to the KinectFusion tracking system [48]. The general downside of infrared cameras is that they are strongly influenced by other sources of infrared light or other high intensity lights, which drastically lowers their outdoor applicability.

**Stereo Cameras** Tracking with stereo cameras showed to improved robustness to noise and occlusions over RGB cameras only [56], as they combine standard monocular optical tracking with additional depth information from stereo clues. However, stereo cameras are relatively rare in mobile devices.

**Marker-based Tracking** Early markers were circular with contrastic [29] or color-coded rings [47], of which four distinctive matches were needed for a correct pose. Square markers were introduced in 1997 [34], so each corner gave a point correspondence. Some patterns like those of ARToolKit [30] have a pattern to distinguish between different markers, so that one square marker is enough to estimate a pose. It became a very popular technique for markers and spawned a lot of successors, e.g. ARToolKitPlus [79]. Markers are a computationally inexpensive way of tracking for AR that works even with a low quality camera, which makes them especially interesting for AR on mobile devices. It is independent of textures, repetitive patterns and specularity but comes at the cost of having to prepare the environment first.

**Natural Feature Tracking** Since having to prepare the environment before tracking is not feasible and often not even possible, a common approach is to track natural features instead. While features can be corners, edges or blobs (i.e. regions of similar properties), only point detectors (i.e. corners and blobs) are suitable for accurate tracking as their position in an image can be measured exactly. Ideally feature detectors should be scale, rotation and affine invariant, repeatable, robust to changes in illumination, robust to noise or blur, distinctive and, of course, be efficient in both calculation time and storage space. While corner features are typically calculated fast, they are often not very distinctive and thus harder to match. Blob detectors on the other hand take more time to calculate but are more distinctive. Popular corner feature detectors of the last years include FAST [62] and Oriented FAST and Rotated BRIEF (ORB) [65], popular blob feature detectors include Scale-Invariant Feature Transform (SIFT) [41] and Speeded Up Robust Features (SURF) [2].

**Conventional Neural Networks** There have also been attempts to use Conventional Neural Networks (CNNs) for object tracking. While Crivellaro et al. [7] received accurate results and robustness to occlusions, it also showed that these approaches are rather computationally intensive: they achieved 4 frames per second (fps) on a current desktop

computer in 2015. However, with new generations of microprocessors in mobile phones and a general increase in computational power, their practicability is growing quickly.

**Model-free Tracking** Trackers can be divided into model-based and model-free ones. While model-based tracking refers to a priori knowledge of an object, which can consist of its texture, a CAD-model or a representation of a scene by simple geometric shapes, model-free tracking generates this information in run-time. Model-free tracking is closely related to Simultaneous Localization and Mapping (SLAM), which describes the problem of generating a map of an unknown environment while at the same time keeping track of the device's current location. Some well known approaches include PTAM [31] [32], ORB-SLAM [46] and LSD-SLAM [14].

In order to track objects that a user can interact with, it is best to use model-based approaches. Although using model-free approaches offer a lot of flexibility, it would be necessary to decide in run-time which parts of a scene belong to the object to track and which not (e.g. the user's hand), which would, for example, make it harder for an algorithm to keep tracking an object's backside, thus making it less robust and reliable.

## 2.3 Model-based Optical Tracking

A priori knowledge of a rigid object can be represented in various ways:

- Point Features: As already mentioned earlier [refer to Natural Feature Tracking]...
- Edges: Edge-based approaches need a priori information of the outlines of an object such as a CAD model of it. Edge filters can be used to find the outlines of objects and match them to saved models [83], other approaches use additional knowledge of the textures [60]. Edge-based approaches work best for man-made objects where an exact 3D model is known, otherwise an accurate 3D scan is required.
- Keyframes: Another way to create save knowledge of a scene in an offline stage is to save a number of keyframes. Vacchetti et al. [74] were able to overcome drastic aspect changes that way.

## 2.4 Perspective-n-Point

Given correspondences between 3D world points and their 2D projections in an image (of a calibrated camera), we want to calculate the 6 Degrees of Freedom (DOF) camera pose from them. Since each point correspondence contributes 2 equations, the absolute minimum of correspondences needed in order two get a finite number of solutions is 3. This still results in up to 4 possible solutions. For coplanar points unique solutions can be found for $n \geq 4$, while in the general case $n = 4$ and $n = 5$ can still have multiple

solutions [18], so $n \geq 6$ is required for a unique solution.
When dealing with noisy data, which is usually the case with real world measurements, an exact solution does not exist and an error function needs to be defined (and minimized). There are two important error functions used in the literature, the *image space error* and the *object space error*:

**Image Space Error** measures the difference between an observed 2D image point and the 2D projection of the expected 3D point:

$$E_{IS} = \sum_i \left|\left| \mathbf{x}_i - \frac{K \cdot [R\,t] X_i}{(K \cdot [R\,t] X_i)_z} \right|\right|^2$$

If the intrinsic camera parameter matrix $K$ has been calculated beforehand, we can measure the differences in the normalized image plane (at distance 1 from the camera center) by replacing $\mathbf{x}_i$ with $\bar{\mathbf{x}}_i = K^{-1}\mathbf{x}_i$:

$$E_{IS} = \sum_i \left|\left| \bar{\mathbf{x}}_i - \frac{[R\,t] X_i}{([R\,t] X_i)_3} \right|\right|^2$$

$$= (u - \frac{R_{11}x + R_{12}y + R_{13}z + t_x}{R_{31}x + R_{32}y + R_{33}z + t_z})^2 + (v - \frac{R_{21}x + R_{22}y + R_{23}z + t_y}{R_{31}x + R_{32}y + R_{33}z + t_z})^2.$$

The image space error is also used for bundle adjustment. While bundle adjustment is closely related to the pose estimation problem, the main difference is that bundle adjustment tries to optimize for both the pose and the 3D coordinates.

**Object Space Error** refers to the distance between a 3D point $X$ in camera space and the back-projected ray $\mathbf{g}$ of the corresponding 2D image point $v$. The function $g(X)$ returns the closest point on the ray $\mathbf{g}$ to the point $X$. Since $\mathbf{g} = \lambda\mathbf{v}$, there exists $\bar{\lambda}$ such that

$$g(X) = \bar{\lambda}\mathbf{v}.$$

Also, since the projection vector is orthogonal to $\mathbf{v}$, it also has to hold that

$$(g(X) - X)^T\mathbf{v} = 0.$$

Combining these two constraints we get

$$\begin{aligned}
(\bar{\lambda}\mathbf{v} - X)^T\mathbf{v} &= 0 \\
\bar{\lambda}\mathbf{v}^T\mathbf{v} - X^T\mathbf{v} &= 0 \\
\bar{\lambda}\mathbf{v}^T\mathbf{v} &= X^T\mathbf{v} \\
\bar{\lambda} &= \frac{X^T\mathbf{v}}{\mathbf{v}^T\mathbf{v}}
\end{aligned}$$

$$
\begin{aligned}
g(X) &= \frac{X^T\mathbf{v}}{\mathbf{v}^T\mathbf{v}}\mathbf{v} \\
&= \mathbf{v}\frac{X^T\mathbf{v}}{\mathbf{v}^T\mathbf{v}} \\
&= \mathbf{v}\frac{\mathbf{v}^T X}{\mathbf{v}^T\mathbf{v}} \\
&= \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}X \\
&= VX.
\end{aligned}
$$

$V$ is called a line-of-sight projection matrix and does an orthogonal projection of a scene point onto the viewing ray of an image point $v$. If $X$ lies on the ray, it holds that

$$X = VX$$

and expressed in world space coordinates we get

$$[R\,t]X_W = V[R\,t]X_W$$

The object space error can therefore be expressed as

$$E_{OS} = \sum_i ||(I - V_i)([R\,t]X_i)||^2 .$$

The difference between image space and object space error is illustrated in Figure 2.2.
Hoewever, as shown by Schweighofer and Pinz [69], the magnitude and position of local minima for image space error and object space error are very similar. While the object space error is often used when calculating a closed form solution, the image space error is usually preferred for iterative methods. We chose to minimize over the image space error, as it assumes the noise to happen during image processing. Additionally, minimizing the image space error results in a better visual alignment of virtual objects rendered onto real objects [82].

**Iterative Methods** usually focus on minimizing an error function for an arbitrary number of points [42] [51] [69], which typically give more accurate results but can suffer from poor initialization and convergence problems. It also comes at the price of higher computational costs. DeMenthon and Davis presented PosIt [8], an iterative algorithm that focuses on orthographic projections and which was later improved to handle coplanar points [51], as they noticed that the cost function can have two local minima, with the solutions being mirror images with respect to a plane that is parallel to the image plane. Schweighofer and Pinz [69] analyzed these two minima for the perspective camera, and

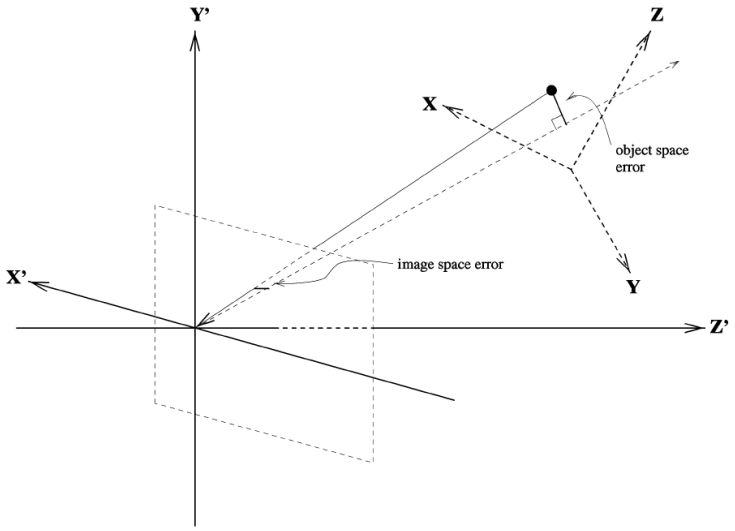



**Figure 2.2:** Illustration of image space error and object space error. [42]

proposed another method for it.

**Closed form solutions** want to solve the problem by solving for a linear system. Fischler and Bolles [18] noted that there is at maximum 4 solutions to the P3P problem and introduced a Random Sample Consensus (RANSAC)-based method (see Section 2.6) over sets of three points where they checked for consistency. Solving the P3P problem typically involves solving for the roots of a polynomial system [38] and can lead to up to four solutions. While for four coplanar non-collinear points there is a unique solution [18], for four non-coplanar it can still be ambiguous. Gao et al. [23] made a full analysis over all solutions to the P3P problem, which will be used in our work. Lepetit et al. [38] were the first ones to propose a non-iterative solution in linear runtime: their EPnP algorithm handles arbitrary numbers of $n \geq 4$ points by building the weighted sum of the null eigenvectors. Other approaches that focused on improving EPnP for the cases of $n = 3, 4, 5$ include Kneip et al. [33], Li et al.'s RPnP [39] and OPnP by Zheng et al. [87]. While closed form solutions generally give good results and are of course faster than iterative methods, the quantities they measure are not geometrically meaningful [27], they cannot take into account all required constraints [28] and so their results are less accurate than those of iterative methods. However, they are often used as an initialization for iterative non-linear optimization methods.

**Perspective-n-Line** approaches are trying to calculate the pose from line correspondences, with the motivation being that they would perform better on man-made structures with repetitive patterns. Notable works include Pribyl et al. [57] and Zhang et al. [85],

however, the accuracy was lower than that of the point based approaches.

**Direct Linear Transformation** The Direct Linear Transformation (DLT) method [71] is a closed form solution and results in a unique solution. Since the matrix P has 11 DOF, the DLT focuses on solving for all of them by solving a linear system. Each point correspondence will contribute with 2 equations, so DLT will require at least 6 point correspondences. The solution to the linear system can be found by a Singular Value Decomposition (SVD) of the matrix A. The DLT does not make use of the additional orthogonality constraints of the rotation matrix R, has reduced stability and requires a higher number of correspondences than necessary.

## 2.5 Outlier Removal

A standard method to remove outliers is RANSAC [18]: by repeatedly randomly picking a subset of the set of correspondences, each time a pose is calculated and the number of inliers for this pose counted, the highest amount of inliers "wins". The number of necessary iterations N can easily be calculated from the proportion of outliers $\epsilon$, the subset size s and the requested probability of success:

$$N = \frac{log(1-p)}{log(1-(1-\epsilon)^s)}$$

In practice N is usually multiplied by ten to increase robustness [21]. However, RANSAC is non-deterministic, meaning that a reasonable result can only be achieved with a certain probability. Since the number of necessary iterations grows exponentially with the number of correspondences needed for a pose, using a minimal point set gives the best results runtime-wise. On the other hand, if the data is very noisy, performance is shown to be better if using more point correspondences than minimally required [64]. Over the years several alternative approaches to RANSAC have been proposed: MLESAC [73] maximizes the likelihood instead of the number of inliers, PROSAC [6] ranks the correspondences before picking sets from them, preemptive RANSAC [49] uses preemptive scoring of the motion hypothesis and has a fixed number of iterations, which makes it popular for real-time applications.

## 2.6 Pose Refinement

For methods with lower accuracy it is reasonable to minimize the error function that is based on the over-determined system of point correspondences. Common methods to solve these non-linear least squares problems include Gauss-Newton method and Levenberg-Marquardt method.

**Gauss-Newton** The Gauss-Newton method is used to solve non-linear least squares problems and is based on the Newton method, which, starting at a given initial point, approximates the function with a quadratic function, as its minimum is easy to calculate. This step is repeated until the minimum or a stop criterion is reached. While the Newton method requires the Hessian matrix (and therefore costly second order derivatives), the Gauss-Newton method avoids calculating the Hesse matrix by approximating it with $2J^TJ$, with $J$ being the Jacobian matrix. For the conventional Gauss-Newton method convergence is not guaranteed, with bad initialization it is even possible for it to diverge. In classical Gauss-Newton, the step size towards direction $\delta$ is always 1, a good value can however also be found by doing line search along this direction, which can be shown to have guaranteed convergence [44]. Empiric results showed that the initial estimate should not exceed 10 percent of object scale for translation and 15 degrees for each Euler angle [42] [26].

**Levenberg-Marquardt** Levenberg-Marquardt is a combination of Gauss-Newton method and gradient descent method: It uses the advantages of gradient descent method to get into the vicinity of the minimum, even in cases where Gauss-Newton would diverge, and then smoothly switches over to using Gauss-Newton. The original Levenberg-Marquardt method has the form

$$(J^TJ + \lambda I)\delta = J^Tr.$$

Hereby, $\lambda$ is called a damping parameter and controls the influence of the gradient descent direction, $\delta$ is the update parameter that we want to solve for. Because the matrix $A = (J^TJ+\lambda I)$ is dominated by its diagonal for large values of $\lambda$, Marquardt [45] later proposed another version, where the problem is scaled before being solved, which is supposed to lead to faster convergence. Over the years, several other versions have been created by different authors, one of them was by Fletcher [19], who instead of the identity matrix added a scaled version of the diagonal of the matrix $J^TJ$:

$$(J^TJ + \lambda diag(J^TJ)\delta = J^Tr.$$

Generally, the Levenberg-Marquardt method increases the chance of local convergence and prohibits divergence.

**Robust Estimation** When dealing with outliers in an optimization problem, outliers can heavily influence the outcome, especially when the error is only calculated as quadratic error (also known as least-squares). A *cost function* $\rho$ defines the contribution per residual to the error function:

$$E = \sum_i \rho(e_i)$$

Common cost functions include

**Squared error** Squared error assumes noise to be Gaussian distributed. It grows without bound in case of outliers. It is defined as: $\rho(e) = e^2$.

**Huber** The Huber cost function is a mix of least squares and L1 norm and is defined as

$$\rho(e) = \begin{cases} \frac{1}{2}e & \text{for } |e| \leq k \\ k|e| - \frac{1}{2}k^2 & \text{for } |e| > k \end{cases}$$

where the tuning constant k represents its resistance to outliers, with smaller values of k producing more resistance. For normally distributed error a value of $k = 1.345\sigma$ produces an efficiency of 95 % [20] while still handling outliers. Instead of calculating $\sigma$ as the standard deviation of the residuals, another robust measure of spread is $\sigma = MAR/0.6745$, with MAR referring to the median absolute residual [20]. In general, smaller values of k gives more robustness to outliers, but can result in lower efficiency for normally distributed errors. The Huber cost function grows without bound too, but less rapidly than least squares.

**Tukey's biweight function** The Tukey's biweight (or bisquare) function levels off when the error exceeds its tuning constant k:

$$\rho(e) = \begin{cases} \frac{k^2}{6}(1 - (1 - (\frac{e}{k})^2)^3) & \text{for } |e| \leq k \\ \frac{k^2}{6} & \text{for } |e| > k \end{cases}$$

where $k$ typically refers to 4.685 $\sigma$ with $\sigma$ being the standard deviation.

Tukey's function is non-convex, while least-squares and Huber cost function increase without limits with the size of the residual, Tukey's function is cut off for $|e| > k$ and rejects outliers more strongly.

### 2.6.1 Pose Representation

For these iterative methods we will need the Jacobians for error minimization. The 3x3 rotation matrix has 9 elements but only 3 DOF. There are many ways to parameterize rotations, some common ways include:

**Rotation Matrices** are the most intuitive representation of rotations. While they are convenient for transforming a point, building the inverse or composing two rotations, they are not well suited for calculating derivatives, as they have 9 elements but only three DOF and the additional constraint $R \cdot R^T = I$ has to be ensured. Some works that still used rotation matrices for their approach include Olsson and Eriksson [52] or Briales and Gonzales-Jimenez [5]. The orthormality constraints are typically added via Lagrange multipliers.

**Euler Angles** split the rotation into three elemental rotations around the orthogonal axes of the Cartesian coordinate system. It was proven by Leonhard Euler in 1775 that every orientation can be reached. The order of the angles is not fixed, so there are twelve possible combinations. Even within a specific order of angles, the angles leading to a result are not unique, as a negative angle can lead to the same result.
Euler angles representation suffers from singularities called "gimbal locks": a continuous subspace that corresponds to the same rotation, so changes within this subspace will not affect the rotation, while changes of two Euler angles will result in the same rotation, thus losing one degree of freedom. Even though the Euler angle representation provides a convenient Jacobian (the matrix of partial derivatives), they usually are not a good choice for optimization problems, as the discontinuity around these singularities can lead to instabilities and jumps. A common workaround for the gimbal lock problem is to change the representation when coming close to singularities [9].

**Axis-angle** representation describes the rotation by an axis by an angle $\theta$. It can either be stored as a unit vector and angle, or the angle can be multiplied onto the unit vector, so it consists of three values on total. Like Euler angles, it is not a unique representation as every multiple of $2\pi$ will lead to the same rotation.
Another synonym for axis-angle representation is exponential map, as rotation matrices can be calculated as the matrix exponentials of skew symmetric matrices:

$$exp([\omega]_\times) = \sum_{n=0}^{\infty} \frac{1}{n!}[\omega]_\times^n,$$

where $[\omega]_\times$ is a skew symmetric matrix, also known as cross product matrix.

$$[\omega]_\times = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}.$$

As a closed-form solution this can be rewritten as the Rodrigues' formula:

$$R = I + \frac{sin\theta}{\theta}[\omega]_\times + \frac{1 - cos\theta}{\theta^2}[\omega]_\times^2$$

For small values of $\theta$ it is usually recommended to use the Taylor series expansions to approximate $sin\theta$ and $cos\theta$. The exponential map has singularities where $||\omega|| = 2n\pi$ for $n \geq 1$, then there is no rotation happening at all. In order to avoid them, one can replace $\omega$ whenever coming close to a singularity by $(1 - \frac{2\pi}{||\omega||}\omega$ to get the same rotation [36]. When differentiating the rotation matrix with respect to the axis-angle vector $\omega$, the Jacobian becomes rather complicated with numerous uses of the trigonometric functions. A more compact formula for the Jacobian was given by

Gallego and Yezzi [22]:

$$\frac{\partial R}{\partial \omega_i} = \frac{\omega_i [\omega]_\times + [\omega \times (I - R)e_i]_\times}{||\omega||^2} R$$

**Quaternions** are less intuitive than the former representations, but powerful and therefore popular: they do not suffer from discontinuities, they represent rotations unambiguously and their partial derivatives of the matrix R with respect to the quaternion parameters do not involve trigonometric functions. A quaternion can be represented as a 4-dimensional vector, consisting of a real element and three imaginary elements, so given a 4-dimensional vector $q = [q_0, q_1, q_2, q_3]^T$, the general form of the quaternion would be

$$q = q_0 + iq_1 + jq_2 + kq_3.$$

The product of two quaternions can be conveniently calculated by using a matrix notation for the left one:

$$q \cdot p = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

The partial derivatives of R with respect to the elements of $q$ can be calculated like this:

$$D_0 = \frac{\partial R}{\partial q_0} = 2 \begin{bmatrix} q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}$$

$$D_1 = \frac{\partial R}{\partial q_0} = 2 \begin{bmatrix} q_1 & q_2 & q_3 \\ q_2 & -q_1 & -q_0 \\ q_3 & q_0 & -q_1 \end{bmatrix}$$

$$D_2 = \frac{\partial R}{\partial q_0} = 2 \begin{bmatrix} -q_2 & q_1 & q_0 \\ q_1 & q_2 & q_3 \\ -q_0 & q_3 & -q_2 \end{bmatrix}$$

$$D_3 = \frac{\partial R}{\partial q_0} = 2 \begin{bmatrix} -q_3 & -q_0 & q_1 \\ q_0 & -q_3 & q_2 \\ q_1 & q_2 & q_2 \end{bmatrix}$$

Since quaternions still have 4 elements while we want to optimize over 3 DOF, we would need to further reduce them. We can achieve this by using the axis-angle

representation and converting them to quaternions when needed:

$$q = (cos\frac{\theta}{2}, sin\frac{\theta}{2}(\omega_1, \omega_2, \omega_3))$$

We can then calculate the derivatives of R with respect to $\omega$ the following way:

$$\frac{\partial R}{\partial \omega_i} = \sum_{j=0}^{3} \frac{\partial R}{\partial q_j} \frac{\partial q_j}{\partial \omega_i}.$$

However, $\partial q_j / \partial \omega_i$ will still include trigonometric functions and the calculations will end up being as cumbersome as if using axis-angle representation directly.

### 2.6.2 Lie Groups

**SO(3)** Rotation matrices in 3D space fulfill the group axioms (closure, associativity, identity element and inverse element). Since the group operations of multiplication and inversion are smooth, they further are a Lie group, called SO(3) (special orthogonal group of order three). The inverse of a rotation matrix is its transpose, the neutral element is the identity matrix.

$$R \in SO(3)$$

$$R \cdot R^T = I$$

Each Lie group has an associated Lie algebra and can locally be replaced by it. A Lie algebra is a vector space that is isomorphic to the tangent space of the unity matrix. The Lie algebra $\mathfrak{so}(3)$ is the set of all 3x3 skew-symmetric matrices. Every element $\omega_x$ can be described as a linear combination of the Lie algebra's basis elements, the so-called generators, and can be seen as a tangent vector.

$$\omega \in \mathbb{R}^3$$

$$\omega_x = \omega_1 G_1 + \omega_2 G_2 + \omega_3 G3 \in \mathfrak{so}(3)$$

$$\omega_x = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}$$

Hereby the group generators have the following form:

$$G_i := \frac{\partial}{\partial \omega_i} exp(\omega_x)\Big|_{\omega=0} = [e_i]_x$$

$$G_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, G_2 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, G_3 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The advantage of using the tangent space is, that it has 6 dimensions, which is exactly the DOF the group transformations (i.e. rotation matrices) have [13].

Instead of calculating the actual derivatives of the matrix R with respect to its axis-angle parameters, we can instead define the updated $R'$ to be $R' = exp([\omega]_\times) \cdot R$. The initial value $exp([\omega]_\times)$ equals the identity matrix. If we consider the rotation of a vector x by R as $y = Rx$, the partial derivatives with respect to R can be replaced the following way:

$$\begin{aligned}
\frac{\partial y}{\partial R} &= \frac{\partial R}{\partial R} x \\
&= \frac{\partial}{\partial \omega}\bigg|_{\omega=0} (exp([\omega]_\times) \cdot R) \cdot x \\
&= \frac{\partial}{\partial \omega}\bigg|_{\omega=0} exp([\omega]_\times) \cdot (R \cdot x) \\
&= \frac{\partial}{\partial \omega}\bigg|_{\omega=0} exp([\omega]_\times) \cdot y \\
&= (G_1 y \,|\, G_2 y \,|\, G_3 y) \\
&= -[y]_\times
\end{aligned}$$

Since this is not exactly the derivative with respect to R itself, we are actually not optimizing over the rotation matrix R itself, but instead over a matrix that is modifying R. One of the first ones to propose this method were Taylor and Kriegman in 1994 [72] and later it became a popular practice [10] [11] [31].

**SE(3)** The SE(3) group is the group of rigid transformations in 3D space, which consist of rotation and translation. The elements of the group look like the following:

$$C = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \in SE(3),$$

where R is an element of the SO(3) group and t a 3-dimensional translation vector. The SE(3) group has 6 dimensions. The corresponding Lie algebra is called $\mathfrak{se}(3)$ and its group generators consist of the generators of the SO(3) group as well as the basis vectors of $\mathbb{R}^3$ for the translation vector.

$$G_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, G_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, G_3 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$G_4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, G_5 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, G_6 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

We define the elements of $\mathfrak{se}(3)$ as $\delta = (\upsilon\, \omega)^T$, $\upsilon, \omega \in \mathbb{R}^3$, and further

$$exp(\delta) = exp \begin{pmatrix} \omega_\times & \upsilon \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} exp(\omega_\times) & V\upsilon \\ 0 & 1 \end{pmatrix},$$

with

$$V = \sum_{n=0}^{\infty} \frac{1}{(n+1)!} \omega^n.$$

Obviously the neutral element of $\mathfrak{se}(3)$ would be $\delta_0 = (0, 0, 0, 0, 0, 0)^T$. For the rotation and translation of a vector x, we want to "ignore" the last line of a matrix C for convenience reasons and define the multiplication as

$$y = f(C, x) = Rx + t.$$

Like before with the SO(3) group, we again want to avoid calculating the actual derivatives of the matrix C but instead take the derivatives of a perturbation matrix that gets multiplied with C:

$$\frac{\partial y}{\partial C} = \frac{\partial}{\partial \delta}\Big|_{\delta=0} \left( exp(\delta) \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \right) \cdot \begin{pmatrix} x \\ 1 \end{pmatrix} \tag{2.1}$$

$$= \frac{\partial}{\partial \delta}\Big|_{\delta=0} exp(\delta) \cdot \left( \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} \right) \tag{2.2}$$

$$= \frac{\partial}{\partial \delta} \cdot y \tag{2.3}$$

$$= (G_1 y \,|\, G_2 y \,|\, G_3 y \,|\, G_4 y \,|\, G_5 y \,|\, G_6 y) \tag{2.4}$$

$$= \begin{pmatrix} I & -[y]_\times \end{pmatrix} \tag{2.5}$$

## 2.7 Tracking on Mobile Devices

Due to lower computational power and a wider variety of scenarios, AR on mobile devices is still challenging. At first there were only marker-based toolkits like ARToolkit, Studierstube [76], ARToolkitPlus or ARTag [17]. The goal of 6 DOF natural feature tracking in real-time was first accomplished by Wagner et al. [77], who used heavily modified SIFT features [41] and Ferns on planar targets. They later improved their approach further with

a template-matching-based tracker [78], which looks for known features in predicted areas of the frame according to the last known camera pose and thus reduce the computational costs. Lepetit and Fua [37] used a decision tree instead of descriptor matching, Ozuysal et al. [54] used random ferns instead, which were both faster than regular descriptor matching but instead required more memory. Some approaches for mobile devices were also based on inertial sensors [25] [81], but suffered from the typical drift problems.
While SIFT and SURF features provide some of the best results, as they are invariant to rotation, scale and affine transformation (to viewpoint changes of up to 60 degrees), they are also some of the most computationally expensive ones. By making use of the parallel hardware of graphics processors, Schulz et al. [68] were able to increase their performance. Some binary features like FAST [63] or ORB [66] (FAST features with BRIEF descriptors) can be computed as much as two orders of magnitudes faster than SIFT, but are not affine invariant. Affine invariance is important for pose detection as it improves matching results in situations of bigger viewpoint changes - which occur a lot when matching a flattened template model to an image of the object.

# 3 Methodology

In this chapter we want to explain the steps of our method in detail and the reason we chose them. An overview over our method is given in Figure 3.1. As can be seen, the task consists of three parts: first we need to represent the object with a template so we can do feature detection and feature matching on it in order to receive 2D-3D correspondences needed for Perspective-$n$-Point (PnP) methods. Next, a PnP method is used to retrieve a pose estimation from the correspondences. Finally, the pose is refined using optimization methods. Each step will be described in detail in the following subsections.

## 3.1 2D Templates and their Mapping

Our goal is to cover a wide variety of shapes, so we chose three specific ones to each represent a whole category of shapes: cuboids, to represent polyhedra, cylinders for developable surfaces, and spheres for undevelopable surfaces (they will be explained in the upcoming subsections). The required input information consists of a 2D template of the object and the objects measurements (e.g. width, height, length). For each shape type we separately built a function to bijectively map points from the 2D template to 3D coordinates.

### 3.1.1 Cuboids

We chose cuboids to represent polyhedra in general. Polyhedra are three-dimensional objects with planar polygonal faces that are only connected along their edges. Polyhedra are not necessarily edge-unfoldable, not even convex polyhedra [3][43]. Other examples of polyhedra would be all pyramids and prisms. For cuboids we chose the well known Latin cross template, as seen in Figure 3.2. Additionally, specification of width, height and length of the cuboid is required.

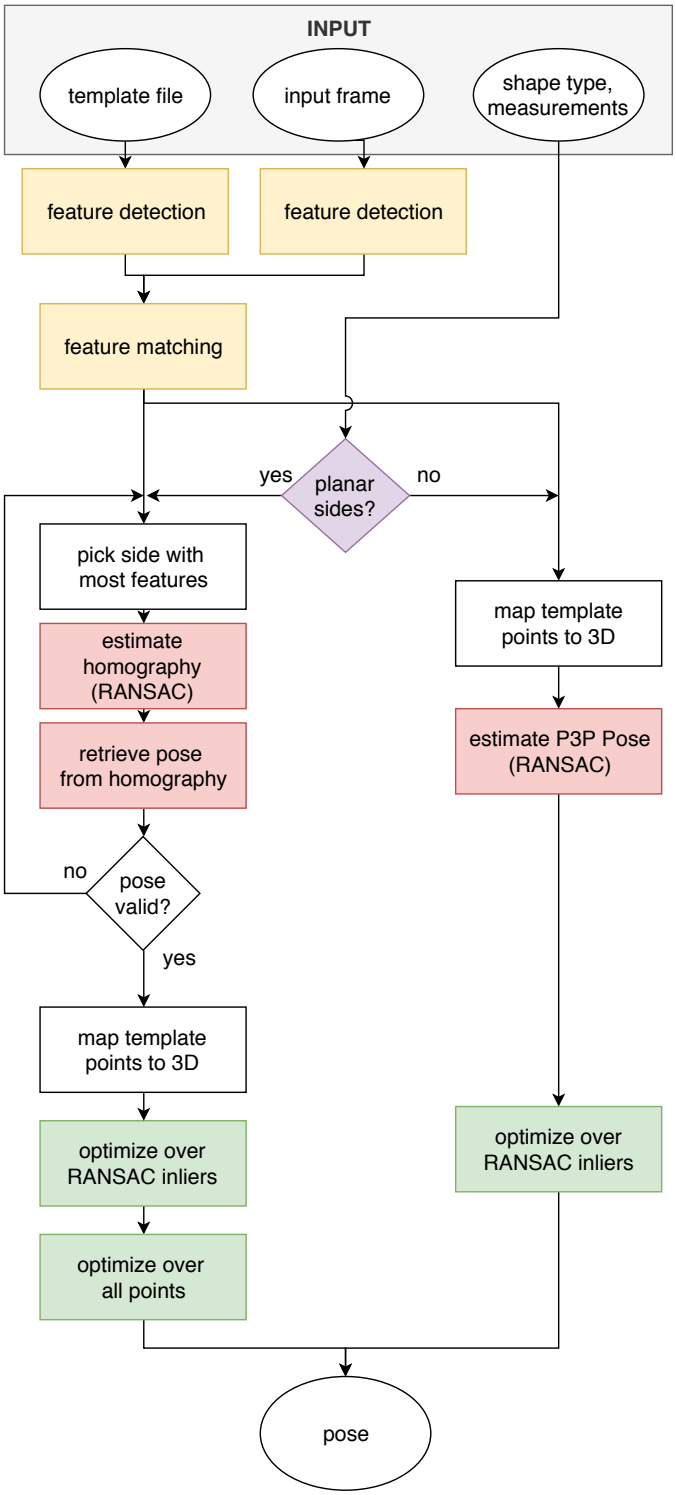



**Figure 3.1:** An overview over the method we use, either for polyhedra (i.e. cubes) or all others (i.e. cylinders and spheres).

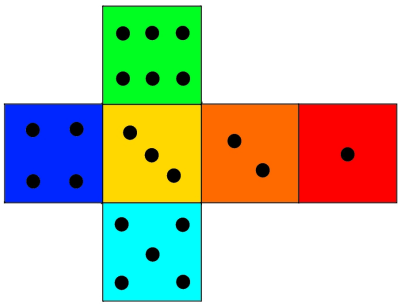


**Figure 3.2:** 2D template of a cuboid.

### 3.1.2 Cylinders

Cylinders were chosen to represent developable surfaces in general. Developable surfaces include all surfaces where curvature only occurs along at most one direction. Other examples of developable surfaces would be cones and truncated cones. The cylinder template we used follows the intuitive pattern of splitting it into its lateral area, top and bottom, as seen in Figure 3.3. The required measurements of a cylinder are height and radius.

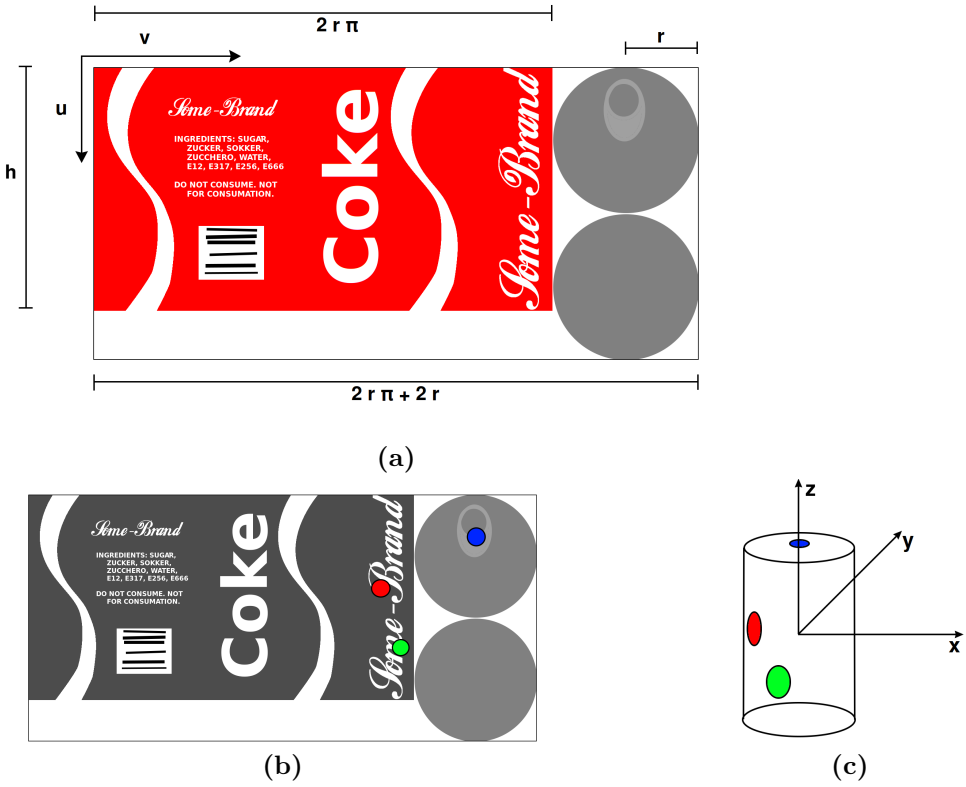
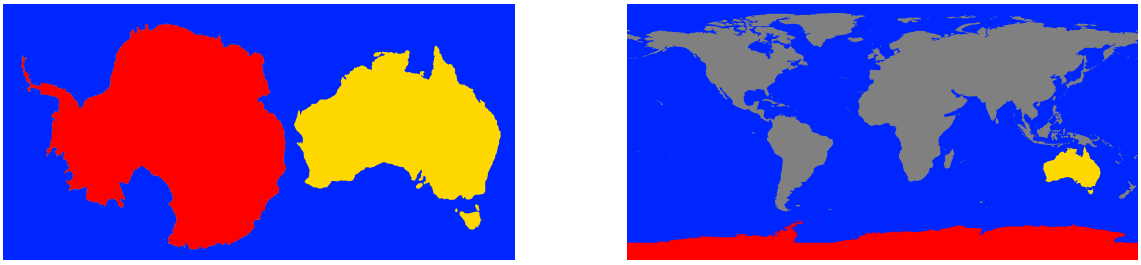



**Figure 3.3:** Cylinder template: The features are detected on the cylinder's lateral area, top and bottom and projected accordingly into 3D space.

### 3.1.3 Spheres

Spheres are a special case, as they are an undevelopable surface. Undevelopable surfaces have curvatures along more than one direction (non-zero Gaussian curvature) and therefore cannot be mapped to a plane without distortion. While this will still work fine for noiseless artificial data, distortion like stretching might affect our feature detection and feature matching results for real world objects. A common example of sphere mapping would be cylindrical projection. As seen in Figure 3.4 this can lead to enormous distortion: while Antarctica has approximately 1.7 times the area of Australia, it becomes multiple times its size if it is mapped to a plane via cylindrical projection. The closer we get to the poles, the stronger the stretching.



**(a)** Size comparison of Antarctica and Australia

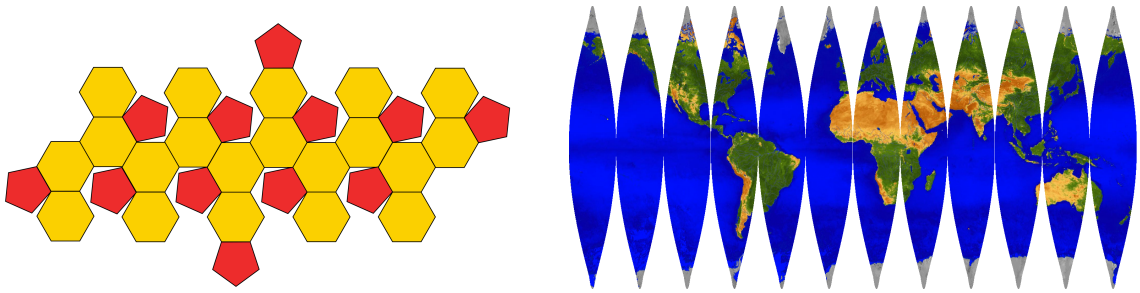
**(b)** Equidistant cylindrical projection

**Figure 3.4:** Equidistant cylindrical projection: while Antarctica actually has roughly 1.7 times the size of Australia, it gets stretched enormously if mapped to 2D.

We therefore want to choose a mapping with little stretching. While this could theoretically be achieved by using a polyhedron with a large number of faces, the increasing number of cuts along the edges would at the same time reduce our chances to find good features. There is only a limited number of semi-regular convex polyhedra (called Archimedean solids, an example can be seen in Figure 3.5a and each net would require a different mapping function. Our choice fell instead on a gore map consisting of 12 gores. An example template can be seen in Figure 3.5b. Its advantage is that the number of gores could easily be increased or decreased in order to make it fit even better to a sphere or to reduce the number of cuts, and that its mapping function is intuitive.

In order to map points $u, v$ from the 2D template onto their corresponding 3D position $(x, y, z)$, our function consists of the following steps:

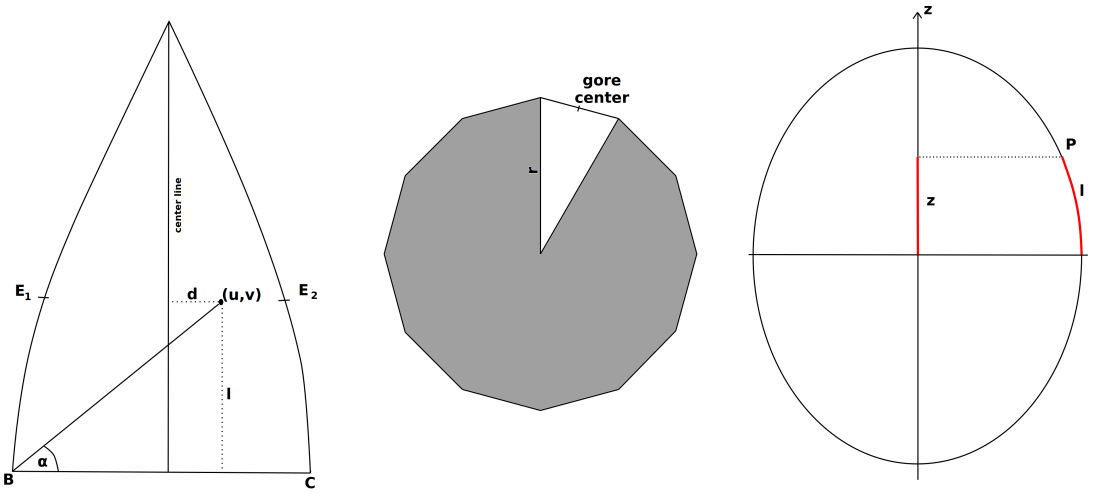
1. Determine which (half-)gore the coordinates $(u, v)$ are on.
2. Calculate
   (a) the distance $l$ between $(u, v)$ and the equatorial line (line BC in Figure 3.6a)
   (b) the distance $d$ between $(u, v)$ and the current gore's center line



**(a)** Net of a 32-sided Polyhedron **(b)** Gore Mapping

**Figure 3.5:** The net of a 32-sided polyhedron, known as truncated icosahedron" [a], and a gore model of 12 gores [b].

[a] Image from Wikimedia Commons
[b] Source of world map: NASA




**(a)** One of the gores **(b)** Equatorial plane **(c)** The ellipse the gore center line is following (exaggerated)

**Figure 3.6:** A sketch of what our sphere mapping looks like: The sphere is cut into gore sections and from the 2D position within that gore we can map a point into 3D world space.

3. Given distance l from the equator, calculate its corresponding z-coordinate in world space: We know that the center line of each gore follows a slight ellipse (Figure 3.6b), the distance l corresponds to its arc length (Figure 3.6c). Since the arc length on the ellipse is calculated by elliptic integrals which cannot be expressed by elementary functions, we cannot easily solve for the corresponding z-coordinate. So we instead calculated the arc length for various z-values and built a look-up table, with z-values

picked more densely around the pole areas. For values in between, the result is interpolated.

$$\epsilon = \frac{b}{a} = \cos\left(\frac{\pi}{\#gores}\right)$$

$$l = \int_0^z \sqrt{1 + \epsilon^2 \frac{x^2}{a^2 - x^2}}\, dx$$

4. Map the point of the gore's center line which is of the same height into world space: Given the z-coordinate, we can easily calculate the position of the gore's outlines at this height, which lie directly on the sphere and therefore follow a circle. The point on the center line is calculated as the mean of the next two gore outline points of this height.

5. Add distance $d$ to the center line point along its interpolation. We now reached the point $P$ on the gore model in world space corresponding to template point $(u, v)$.

6. Project $P$ onto the actual sphere: since our origin lies in the center of the sphere, it's sufficient to scale $P$ to the radius of our sphere.

As specifications for the mapping function, the radius and the number of gores are required.

## 3.2 Initial Pose Estimation

As mentioned in Section 2.4, iterative pose estimation methods typically require an initial pose in order to step-wise improve it further. A convenient way for planar surfaces is to use homographies. Unfortunately this only works for object that actually have planar surfaces. A more general solution is P3P, which takes any 2D-3D point correspondences.

### 3.2.1 Homographies

While a homography typically only represents the projection of a plane in one image to another one, we can as well use it to retrieve the camera pose: Consider the plane $Z = 0$ in 3D world space:

$$\bar{\mathbf{x}} = K^{-1}\mathbf{x} = [R\,t]\mathbf{X}$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$
$$= H \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Therefore,

$$H = \lambda K [r_1 r_2 t]$$
$$K^{-1} H = \lambda [r_1 r_2 t].$$

Given the homography between a plane $Z = 0$ in world space and the image we can reconstruct R and t: since the column vectors $r_1$ and $r_2$ are unit vectors, we retrieve $\lambda$ from the first two columns of $K^{-1}H$ (or their average in case they are not equal). The third column of $R$ can then be calculated as the cross product of the other two columns, as rotation matrices are orthonormal. Even though the first two columns might not always be truly orthogonal due to noisy point correspondences, this can be corrected by normalizing them and re-positioning them to 45 degrees from their common bisector.

Since we can only use points of one side of the object, we obviously choose the side with the most feature matches on it. In case we fail to retrieve a valid pose from that side, we try again with the next best one. To decide if a pose can be considered valid, we check if the rotation matrix R is orthogonal ($R^T \cdot R = I$) and if the object would even be in front of the camera (meaning the positive z-axis in camera space would have to point towards the origin in world space).

For convenience, we use an OpenCV function to calculate homographies. OpenCV [4] is an open source library for various computer vision applications. It offers a variety of feature detection methods, that will come in handy for our practical application, so we decide to also use its built-in Random Sample Consensus (RANSAC)-based function *findHomography*, as RANSAC is an ideal way to deal with outliers.

```
Mat cv::findHomography ( InputArray      srcPoints,
                         InputArray      dstPoints,
                         int             method = 0,
                         double          ransacReprojThreshold = 3,
                         OutputArray     mask = noArray(),
                         const int       maxIters = 2000,
                         const double    confidence = 0.995
                       );
```

Hereby we used the method parameter CV_RANSAC. Various values for the reprojection threshold, number of iterations and confidence parameters were tested during our evaluations and will be described in detail in the next chapter. We modified the original OpenCV function by removing its last step, a Levenberg-Marquardt optimization over all inliers, as we only needed the RANSAC and homography step itself. Since we can only use a coplanar subset of the point correspondences to find a homography, other sides of an object might be ignored. To add their information into our calculations, we will perform two optimization steps after a homography was found: once over all RANSAC inliers and once including points from other sides.

### 3.2.2 P3P Method

For all object shapes that contain few to none planar sides, e.g. cylinders or spheres, we cannot use homographies and therefore had to fall back to PnP methods. We chose a RANSAC-based OpenCV function, *solvePnPRansac*:

```
bool cv::solvePnPRansac ( InputArray    objectPoints,
                          InputArray    imagePoints,
                          InputArray    cameraMatrix,
                          InputArray    distCoeffs,
                          OutputArray   rvec,
                          OutputArray   tvec,
                          bool             useExtrinsicGuess = false,
                          int           iterationsCount = 100,
                          float            reprojectionError = 8.0,
                          double           confidence = 0.99,
                          OutputArray   inliers = noArray(),
                          int           flags = SOLVEPNP_ITERATIVE
                        )
```

This function uses the RANSAC algorithm over subsets of four point. For each subset the function solvePnP is called, which is based on the paper of Gao et al. [23]. The approach offers a complete solution classification for sets of three point correspondences, where each set can have up to four possible solutions. The fourth point is then used to determine the solution with the smallest reprojection error. The final step of RANSAC would be to re-estimate the pose on all inliers. The OpenCV version 3.4.1 of solvePnPRansac does so by applying EPnP on all inliers. Since we strictly wanted to compare P3P to homographies, we removed that. The re-estimation will instead be done by our own pose refinement. As parameters we used the flag SOLVEPNP_P3P, no extrinsic guess and varying values for iteration count, reprojection error and confidence, their evaluation will be shown in the next chapter.

## 3.3 Pose Refinement

In practice, point correspondences are noisy and an exact solution usually does not exist. We therefore want to minimize the error between the 2D observed points in a frame and

their 3D positions on the model.

$$E_{IS} = \sum_i \left|\left| \bar{\mathbf{x}}_i - \frac{K \cdot [R\,t] X_i}{(K \cdot [R\,t] X_i)_z} \right|\right|^2$$

For convenience we multiplied both sides with the inverse of the camera intrinsics K, so we only need to apply it to the observed points ones.

$$\begin{aligned} E &= \sum_i \left|\left| \mathbf{x}_i - \frac{[R\,t] X_i}{([R\,t] X_i)_3} \right|\right|^2 \\ &= (u - \frac{R_{11}x + R_{12}y + R_{13}z + t_x}{R_{31}x + R_{32}y + R_{33}z + t_z})^2 + (v - \frac{R_{21}x + R_{22}y + R_{23}z + t_y}{R_{31}x + R_{32}y + R_{33}z + t_z})^2 \end{aligned} \tag{3.1}$$

Robustness to outliers can additionally be achieved by choosing another cost function than quadratic error in order to cut off the costs an outlier will give, as described earlier in Section 2.6. Robust cost function will often resemble quadratic error around zero but cut off the costs for higher distances. We expect the majority of outliers to already be eliminated by RANSAC, but to be safe we decide to use Huber cost function for all optimization methods we are going to use.

**Parameterization** A rule of thumb is to preferably choose a parameterization which makes the error function as close as possible to being linear, as this will lead to fewer local minima and faster convergence [27]. However, for calculating the partial derivatives of the error function with respect to the elements of the rotation vector r and the translation vector t, ur intuitive approach is to use the Rodrigues formula to express R by r:

$$\theta = ||r||^2$$

$$\bar{r} = r/\theta$$

$$R = cos\theta I + (1 - cos\theta)\bar{r}\bar{r}^T + sin\theta \begin{bmatrix} 0 & -\bar{r_3} & \bar{r_2} \\ \bar{r_3} & 0 & -\bar{r_1} \\ -\bar{r_2} & \bar{r_1} & 0 \end{bmatrix}$$

For the error function

$$E = \sum_{i=1}^{n} ||K^{-1}x - [R\,|\,t] \begin{bmatrix} X \\ 1 \end{bmatrix} ||^2,$$

the partial derivatives are given by the Jacobian matrix

$$J = \nabla_{r,t}(RX + t)^T.$$

Inserting one formula into another, the elements of the Jacobian quickly become very big and we will therefore only include them in Appendix A. Considering that the vector

$(RX + t)$ is a homogeneous vector, we can normalize it by applying a matrix $V_{(RX+t)}$ to the 3x6 Jacobian, with V being defined as

$$V_{\mathbf{t}} = \begin{bmatrix} \frac{1}{t_z} & 0 & -\frac{t_x}{t_z^2} \\ 0 & \frac{1}{t_z} & \frac{t_y}{t_z^2} \end{bmatrix}.$$

While this Jacobian leads to correct results, we also implement derivatives according to Lie groups, as previously mentioned in Section 2.6.1. Not only are the formulas much easier, they also do not need any trigonometric functions. Using Lie groups, the Jacobian can be calculated as

$$J = \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & [RX + t]_\times \\ 0 & 0 & 1 & \end{bmatrix}$$

with

$$[\mathbf{v}]_\times = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}.$$

**Ceres Solver with Automatic Derivatives** Our first approach was to use the Ceres Solver library [1] and its automatic differentiation, where the derivatives are calculated by expressing the function as a combination of simple functions (e.g. addition, multiplication, logarithm, cosine), so that the derivative can be calculated by the library via the chain rule. We defined the error function as shown in Equation 3.1. Of the solvers offered by Ceres Solver, we picked Levenberg-Marquardt. As a cost function we chose least squares, as we did not expect heavy outliers due to RANSAC.

**Ceres Solver with Analytic Derivatives** The next step was to use the same error function but instead of using the automatic differentiation we wanted to hand the function the derivatives ourselves. As with automatic derivatives, we picked Levenberg-Marquardt as solver.

### 3.3.1 Gauss-Newton Method

As our next step we wanted to implement a straightforward optimization method in order to compare it to the library methods. Our choice fell on Gauss-Newton method. Even though Gauss-Newton method, compared to Levenberg-Marquardt method, has the disadvantage of potentially diverging if the starting point is far from the optimum, we assume that this might not be an issue if our pose estimates are good enough. Gauss-Newton iteratively updates a pose parameterization $x$ as follows: consider the error vectors between

observed 2D points and their 3D model points and the residuals $E_i$ as their norm.

$$v_i = \mathbf{x}_i - \frac{[R\,t]X_i}{([R\,t]X_i)_3}$$

$$E_i = ||v_i(x)||^2 = v^T v$$

Gauss-Newton approximates the error vectors $v_i$ as a function of the pose parameters $x$ and a perturbation vector $\delta$ by a first-order Taylor expansion:

$$\begin{aligned} v_i(x \oplus \delta) &\approx v_i(x) + \frac{\partial v_i(x)}{\partial x}\delta \\ &= v_i(x) - J_i\delta \end{aligned} \tag{3.2}$$

with $\oplus$ referring to the addition of the preferred pose parameterization (e.g. vector addition for angle-axis representation, left multiplication for Lie groups). In order to minimize the residuals, we then need to differentiate with respect to $\delta$.

$$\begin{aligned} E &= \sum_i E_i \\ &= \sum_i (v_i(x) - J_i\delta)^T (v_i(x) - J_i\delta) \\ &= v(x) - J\delta)^T (v(x) - J\delta) \end{aligned} \tag{3.3}$$

$$\frac{\partial E}{\partial \delta} = -2(v - J\delta)^T \cdot J$$

$$v^T J - \delta^T J^T J = 0$$

$$\delta^T J^T J = v^T J$$

$$J^T J\delta = J^T v$$

$$\delta = (J^T J)^{-1} J^T v$$

This can then easily be solved via Cholesky decomposition. Since the features and their observations are independent of each other, the matrix $J^T J$ can be accumulated:

$$J^T J = \sum_i J_i^T J_i$$

$$J^T v = \sum_i J_i^T v_i$$

The Gauss-Newton method is not guaranteed to converge, so we need to either stop when the update falls below a convergence threshold or when a maximum number of iterations is reached.

### 3.3.2 Levenberg-Marquardt method

To make sure we can later determine whether any differences between our implementation of Gauss-Newton method and the Ceres Solver version of Levenberg-Marquardt method arise from actually comparing the library to an own implementation, or from comparing two different optimization methods, we decided to additionally implement Levenberg-Marquardt method. Of the various versions existing of Levenberg-Marquardt, we chose the version described by Ethan Eade [12] as Levenberg method. Similar to Gauss-Newton, we set up the linear system

$$(J^T J + \lambda I)\delta = J^T r.$$

Hereby, the damping parameter $\lambda$ controls the influence of the gradient descent direction. Starting with an initial damping parameter of value 1, we solve this linear system for $\delta$ and update our parameter vector as

$$x' = x \oplus \delta.$$

Note that in the case of using Lie group representation, $\oplus$ refers to a multiplication of group elements. After the update step, we test if the value of the error function has actually been decreased. It it decreased, the update step is accepted and the damping factor is decreased by a factor $a$, making the Levenberg-Marquardt method work more similar to Gauss-Newton.

$$x \leftarrow x'$$
$$\lambda \leftarrow \frac{1}{a}\lambda.$$

If the proposed update step did not decrease the value of the error function, the update is rejected and $\lambda$ gets increased by a factor $b$, which increases the influence of the gradient descent method.

$$\lambda \leftarrow b\lambda.$$

For factors $a$ and $b$, we chose $a = 2$ and $b = 10$. The method ends, when either a maximum number of iterations is reached, the value of $\lambda$ gets so high that it endangers the numerical stability or the update steps become too small.

### 3.3.3 Robust Estimation

If we additionally want robustness against outliers, a cost function $\rho$ needs to be added:

$$\begin{aligned} \frac{\partial E_{robust}}{\partial x} &= \sum_i \rho(E_i) \\ &= \sum_i \frac{\partial \rho(E_i)}{\partial E_i} \frac{\partial E_i}{\partial x} \end{aligned} \tag{3.4}$$

So in order to calculate the update $\delta$, we need to add the derivative of the cost function $\rho$ as a weight function $W_i$. This leads to the update

$$\delta = (\sum_i W_i J_i^T J_i)^{-1} (\sum_i W_i J_i^T v_i)$$

For the least squares cost function the weight would be 1, for the Huber cost function it is

$$\rho(E) = \begin{cases} \frac{1}{2}E^2 & E \leq k^2 \\ k\sqrt{E} - \frac{1}{2}E^2 & E > k^2 \end{cases}$$

$$W_H(E) = \begin{cases} 1 & E < k^2 \\ \frac{k}{\sqrt{E}} & E \geq k^2 \end{cases}$$

# 4 Experiments

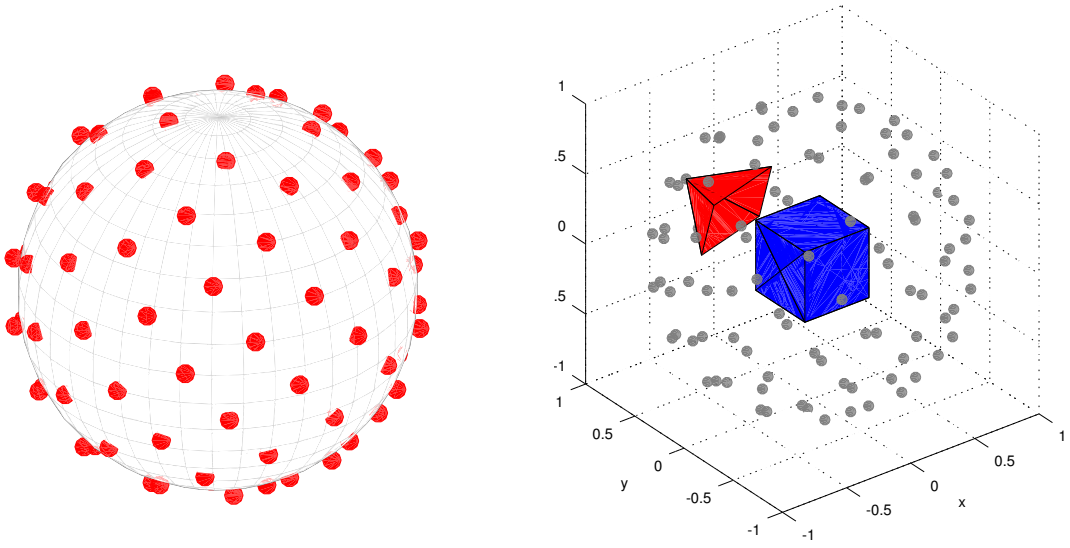
In order to evaluate our method, we need test data. Ideally we would have a database of images of primitive shapes with given textures/templates, size and exact pose ground truth. Since we do not have that at hand, for optimal accuracy we instead generate artificial test data. Another advantage of artificial data is, that our results do not rely on our choice of feature detectors, feature description and feature matching methods and we can control the amount of noise and outliers directly.

## 4.1 Test Data

First we need to generate poses to cover our object from all sides, so we generate a certain number of points (n = 100) on the unit sphere. We choose a spherical Fibonacci lattice, as can be seen in Figure 4.1a. Not only are the points somewhat evenly distributed (mutually equidistant points on a sphere are not possible for most numbers n), they also never have the same latitude or longitude, which makes the resulting poses more unique. We use these 3D points as the camera positions in world space for our ground truth. We further need to specify the camera's orientation, so we set the camera's z-axis facing towards the object and the camera's x-axis as its upward-pointing normal in world space, as seen in Figure 4.1b.

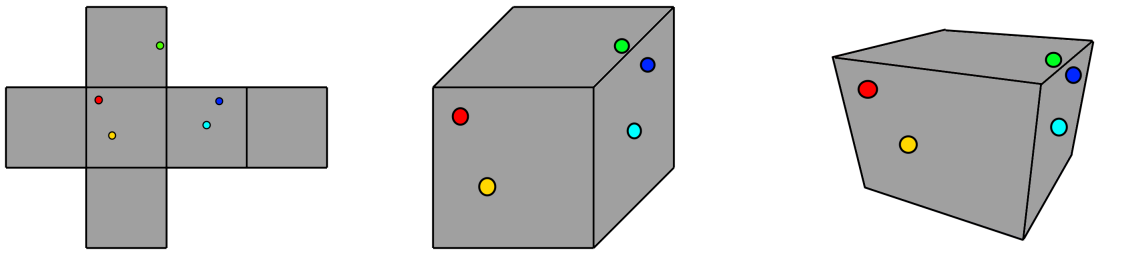
We further need artificial feature positions and matches, so for each pose we select random points on the template, map them onto the 3D model of the object, check if they would even be visible from the current pose (and discard them if not), and apply the projection matrix P to get their coordinates in the artificial image. This is displayed in Figure 4.2.

In order to simulate outliers, a certain (varying) amount of the image coordinates are chosen at random within the image (as features could in a real picture be




**(a)** Camera positions **(b)** Camera orientations

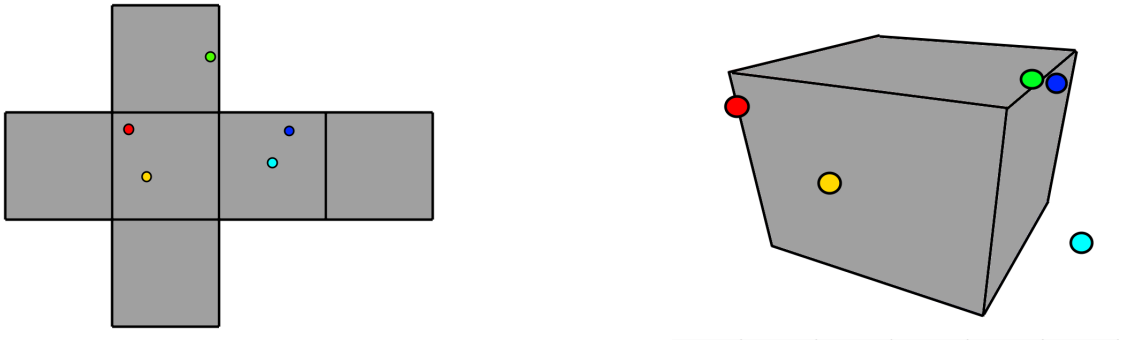
**Figure 4.1:** Evenly distributed camera poses facing the object.



**(a)** Random template points **(b)** Mapped onto 3D model **(c)** Projected into camera frame

**Figure 4.2:** Random points on the template get projected into 3D space and into the camera image.

detected anywhere on the background as well). To simulate noise, Gaussian noise is being added to the image feature coordinates. As the final result we have artificial matches of feature points on the template and in the frame, as pictured in Figure 4.3.



**(a)** Features on template **(b)** Features in the frame

**Figure 4.3:** Matched feature points on the template and in the camera image.

Our artificial testdata now consists of:

- the object type (i.e. cuboid)
- the object parameters (i.e. length, width, height)
- n camera poses (as rotation matrix R and translation vector t)
- n · m feature points on the texture
- n · m feature points in the frame

We generate test data sets for the range of

- outlier level from 0 % to 50 %
- normally distributed noise with standard deviation from 0 to 1.6 (image size 640 × 480)
- number of features from 10 to 100
- special case: only one side of the object visible
- special case: random points (= 100 % outliers)

## 4.2 Test Metrics

We decide for the following test metrics:

**Translation Error:** $e_{trans} = \frac{||t_{GT}-t||}{||t||} \cdot 100$
As in [16], the translation error represents the difference in camera positions in world space in relation to the distance to the object.

**Rotation Error:** $\max_{k=1}^{3}\{arccos(r_{k,GT}^{T} \cdot r_k)\}$
As in [16], the rotation error represents the maximum of the three Euler angles between the two poses.

**Success Threshold:** We empirically choose a translation error of up to 10 to still be accepted as a solution and a rotation error of up to 5. For the first pose estimate we want to be more generous, as the optimization step might still correct the result, so we choose a multiple of it: translation error of up to 30 and rotation error of up to 10 are considered a 'success'.

**Specifications:** We run our following tests on a computer with Intel Core i7-4771 @ 3.50GHz CPU, running Ubuntu 16.04, with GCC version 5.4 and OpenCV 3.4.1.

## 4.3 Functionality Testing

**Ground truth** The very first step is to generate test data without noise or outliers to test whether or not our method returns the correct ground truth and thus to verify that it is working correctly.
We start by running the homography-based and the P3P method both without any optimization. With the number of Random Sample Consensus (RANSAC) iterations high enough and the confidence value set to 0.995, the only parameter left to choose is the inlier threshold. While in theory it should be close to zero, we get slightly better results for the homography-based method when using more forgiving thresholds of 0.01 and above, caused by numerical errors. For the P3P method small thresholds beneath 0.00001 works best. Maximal, mean and median errors of both methods are listed in Table 4.1. This covers both translation and rotation errors.

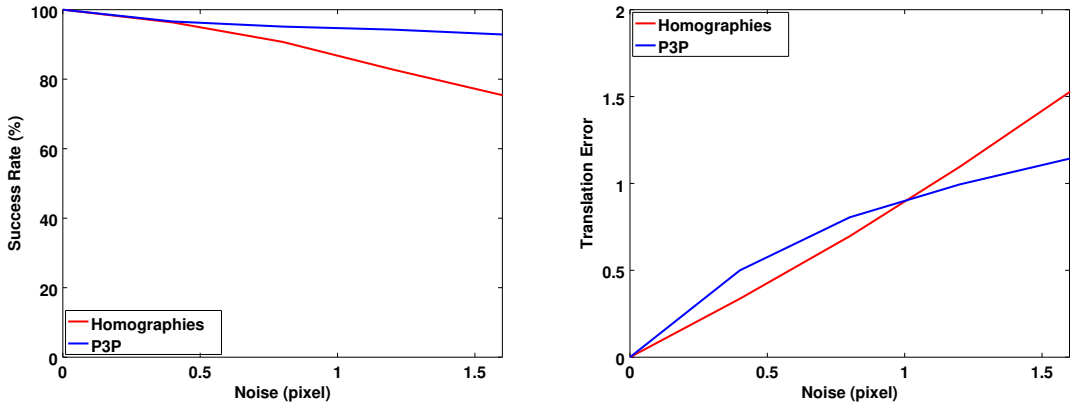
To verify that using any of the chosen optimization methods (i.e. Ceres Solver with automatic derivatives, Ceres Solver with analytic derivatives, Gauss-Newton method, Levenberg-Marquardt method, with either derivatives of the rotation matrix or using Lie groups) can only reduce the error further, we re-run the same tests including them. For simplicity, and as the results are very close to each other, Table 4.1 shows the maximal error occurring with any of these six combinations.

| **Homographies** | Maximal Error | Median Error | Mean Error |
|---|---|---|---|
| Translation Error, no Opt. | 0.0111 | $e^{-06}$ | $e^{-05}$ |
| Translation Error, with Opt. | $e^{-06}$ | $e^{-12}$ | $e^{-09}$ |
| Rotation Error, no Opt. | 0.0098 | $e^{-05}$ | $e^{-05}$ |
| Rotation Error, with Opt. | $e^{-06}$ | $e^{-07}$ | $e^{-06}$ |

| **P3P** | Maximal Error | Median Error | Mean Error |
|---|---|---|---|
| Translation Error, no Opt. | $e^{-05}$ | $e^{-06}$ | $e^{-06}$ |
| Translation Error, with Opt. | $e^{-09}$ | $e^{-10}$ | $e^{-10}$ |
| Rotation Error, no Opt. | $e^{-05}$ | $e^{-06}$ | $e^{-06}$ |
| Rotation Error, with Opt. | $e^{-06}$ | $e^{-07}$ | $e^{-07}$ |

**Table 4.1:** Functionality tests on data with no noise and no outliers.

**Robustness to Noise** In order to analyze the influence of noise we use test data with various levels of Gaussian noise but with no outliers. As expected, the success rate drops with the amount of noise while the error rate goes up. The P3P method shows better results than homographies, which is not surprising considering it uses points of all sides.




**(a)** Success Rate **(b)** Median Translation Error

**Figure 4.4:** The effect of different noise levels on the success rate and median translation error on both the homography method and the P3P method.

**Robustness to Outliers** Outliers should obviously be handled by RANSAC, so a quick test on various outlier levels without noise reveals that for $k = 100$ iterations the results hold as expected until approximately 80 %, as shown in Figure 4.5, and drops according to the formula: given 100 correspondences and 90 % outliers for the homography approach, we would have 10 inliers in total, which would in the worst case be spread over the three visible cube sides with only one of them containing 4 inliers. Thus it would take on average 3.9 million iterations to find the only correct solution in that scenario.

**Correctness of Optimization** We test the correctness of the various optimization options by applying them on poses we receive from a first pose estimation on various noise levels, at the absence of outliers. Figure 4.6 shows the success rate and median translation error rate after a first pose estimate has been done via P3P method. As expected, all versions of our optimization increase the success rate to almost 100 % and are almost equal. The median translation error drops accordingly and is shown representative for one of the methods (namely Gauss-Newton in combination with Lie groups) in more detail in Figure 4.7, where we show the area between the lower and upper quartile of the median translation error and median rotation error. This area corresponds to the middle 50 % of the errors.

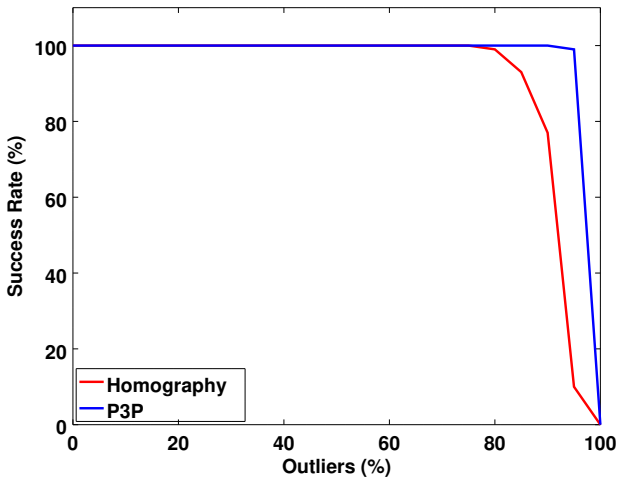
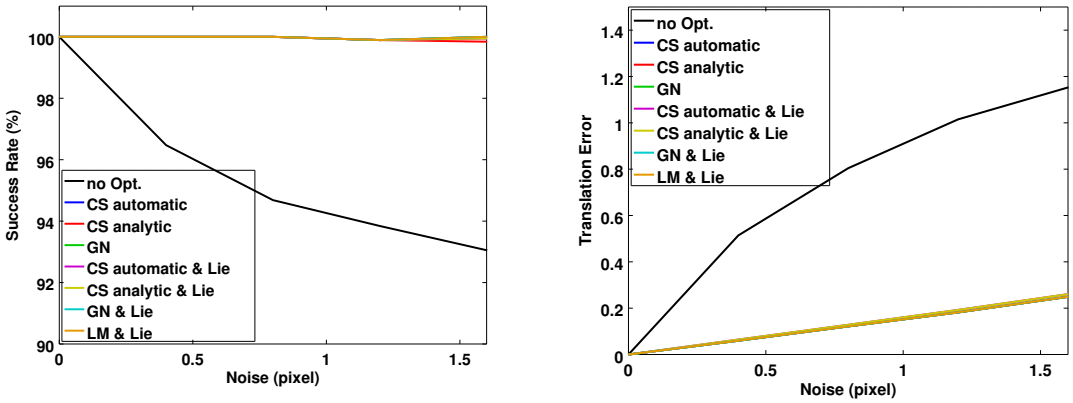



**Figure 4.5:** The number of correct results for test data without noise. Given enough iteration steps and at least 4 inliers, RANSAC will always find a correct solution. The homography method reaches its limit earlier because it requires 4 inliers being on the same side of a cube, while for P3P they may be spread over the whole cube.

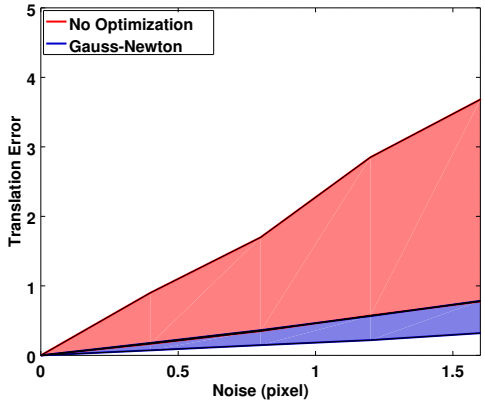
## 4.4 Evaluation

**Homographies vs. P3P: Accuracy and Robustness** At first we are interested to compare the success rate of homography and the P3P method without any additional optimization. Using test data varying between 10 and 100 features, Gaussian noise between

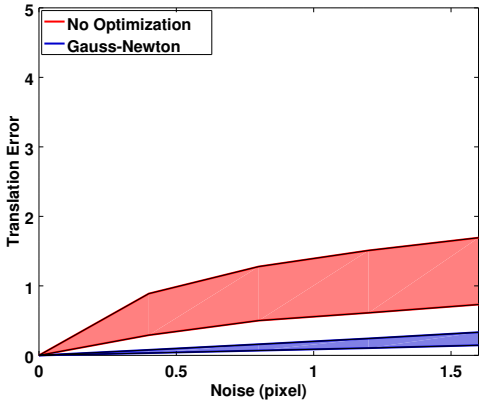



(a) Success Rate (b) Median Translation Error

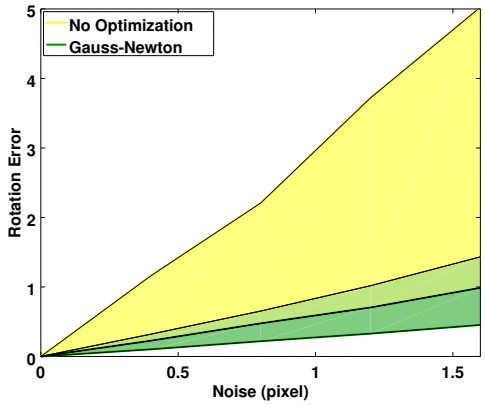
**Figure 4.6:** The different optimization functions work correctly on a first P3P pose estimation over various outlier levels: the success rate goes up to almost 100 % while the error rate is reduced significantly.



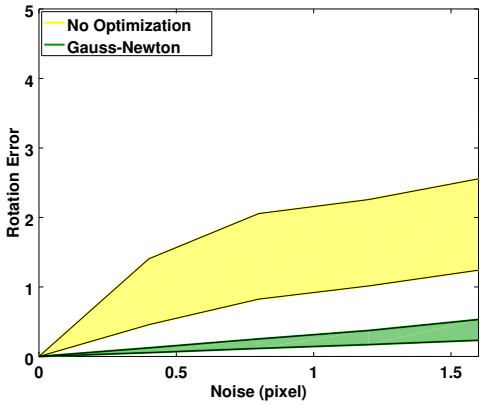

**(a)** Translation Error for Homographies




**(b)** Translation Error for P3P




**(c)** Rotation Error for Homographies

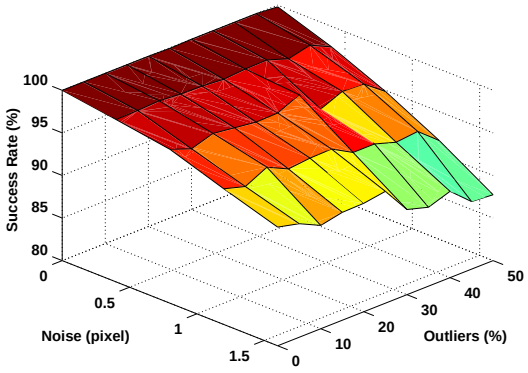



**(d)** Rotation Error for P3P

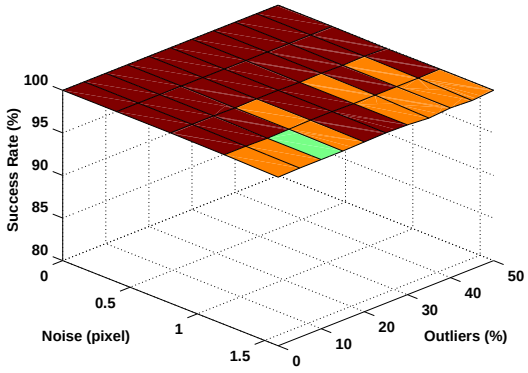
**Figure 4.7:** The influence of Gaussian noise on both homography and P3P approach. No outliers. The areas show the interval between the lower and upper quartile of the errors, making the red/blue/yellow/green colored areas cover the middle 50 % of the results.

0.0 and 1.6 and outliers from 0 % to 50 %, we immediately notice that the homography method fails to find any pose in 2.3 % of the cases. This is simply due to the fact, that 10 features spread across potentially 3 visible sides will worst case only leave 4 features on one side to calculate a homography from - not enough if any of them happen to be an outlier. So to make sure the homographies even get a chance to work, from now on we only consider test data with at least 30 features. We choose the RANSAC parameters generously in order to hopefully catch any acceptable pose: maximum iterations of 20000, confidence of 0.999 and reprojection threshold of 5.0, which is about 3 times our maximum standard deviation. The results can be seen in Figure 4.8. Comparing the success rate and the median translation and rotation errors, P3P clearly shows superior performance: its success rate is much better while the median errors are less sensitive to noise. We are looking at median errors instead of average errors because the average error could be

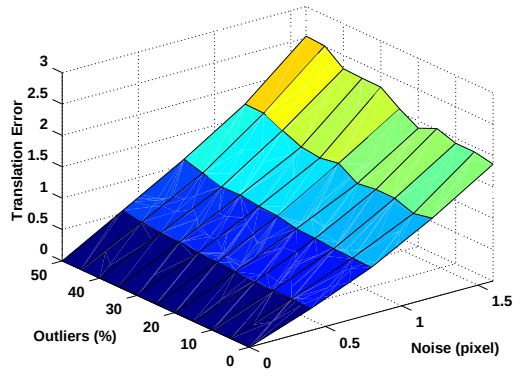
influenced heavily by even a single bad results.



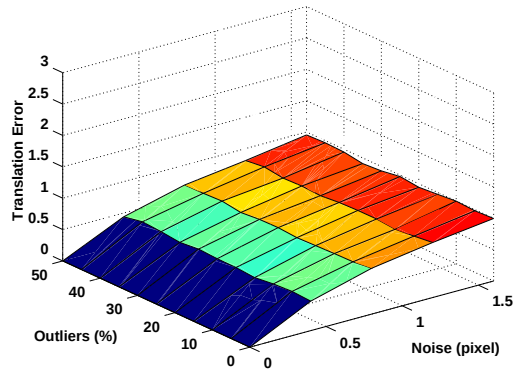

**(a)** Success Rate Hom.



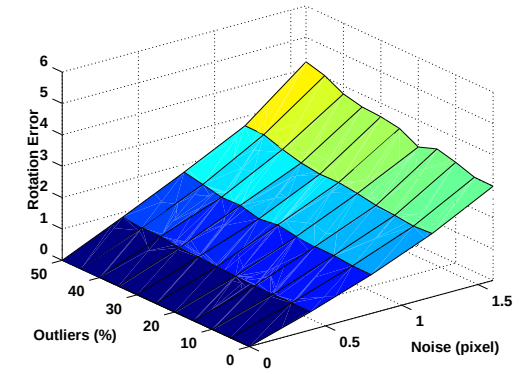

**(b)** Success Rate P3P



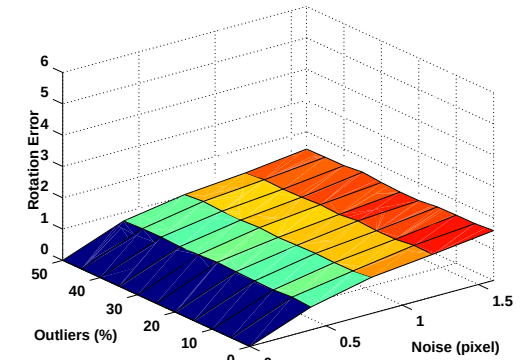

**(c)** Median Translation Error Hom.




**(d)** Median Translation Error P3P




**(e)** Median Rotation Error Hom.




**(f)** Median Rotation Error P3P

**Figure 4.8:** Success rate, median translation error and median rotation error of both the homography and P3P method. Without optimization.

However, this could be expected, as the P3P method gets a lot more features 'to work with' than homographies, which only receive features of one side to work with. But what

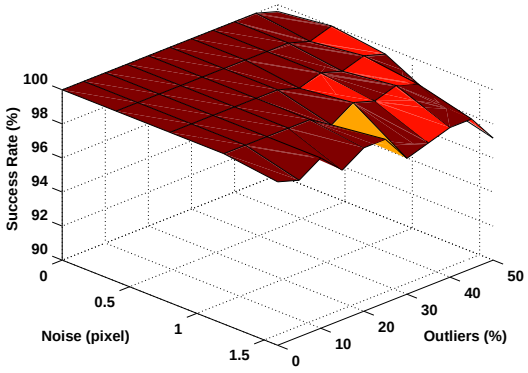
if both had the same amount of features available? To simulate this, we run both methods on a data set consisting of only poses, where just one side of the object was visible. The results can be seen in Figure 4.9. It seems that here homographies do much better: the influence of noise is a lot lower on them for both success rate and errors.

**Homographies vs. P3P: Runtime** For the runtime, we first want to check if the run time of our pose estimation methods significantly depends on the amount of noise and outliers. We therefore run a test with fixed RANSAC values over various amounts of noise and outliers: As can be seen in Figure 4.10, both methods clearly depend on both the amount of noise and outliers. On the first look it seems that homographies have a lower run time than P3P and one might assume that that is again caused by the fact that the homography method receives less point correspondences than the P3P method if more than one side of the object is visible. To test that, we run the same test for poses where only one side of the cube was visible and both methods therefore receive the same amount of point correspondences. However, in Figure 4.11 it is shown that in fact homographies clearly outperform P3P for coplanar points.

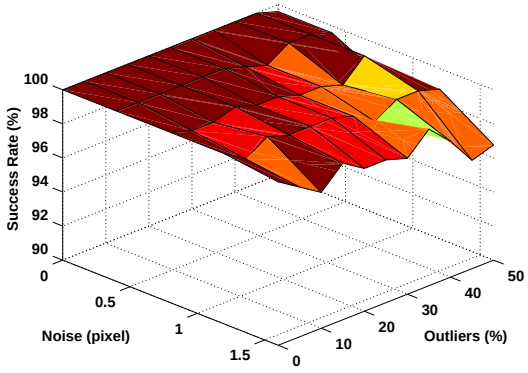
To get more details on the run time itself, we examine their worst case scenario: We choose a set of random point (which simulates absence of the object) to have a closer look at the RANSAC parameters. We expect the run time to be unaffected by RANSAC confidence and reprojection threshold values, but to grow linearly with the number of RANSAC iterations. As Figure 4.12a shows, the P3P method in fact grows linearly while the homography method becomes constant at some point. This is a result of our test data having a maximum of 100 features, which leads to on average 16.6 (random) points per side. For each iteration of RANSAC, 4 of those points are chosen to calculate a pose, resulting in 4/16.6 inliers. In the OpenCV implementation of RANSAC, the maximum number of iterations is re-evaluated at each iteration via the RANSAC formula

$$k = \frac{log(1-p)}{log(1-w^n)},$$

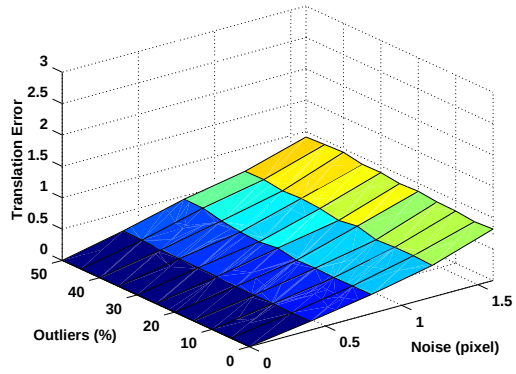
with k being the number of iterations, p the confidence, w the number of inliers and n the number of model points. So for 4 inliers out of 16.6 points, the new maximum number of iterations would become about 2045. Therefore, the run time of homographies becomes almost constant in Figure 4.12a. To simulate a bigger number of point correspondences, we let both RANSAC methods run without recalculating the number of iterations. As can be seen in Figure 4.12b, the run time then grows linearly for both. It also shows that for random data, where the findHomography method has to be called for each side of the cube, the total run time of the homography method is 3 times as high as for P3P. The run time of one function call is approximately half of P3P.



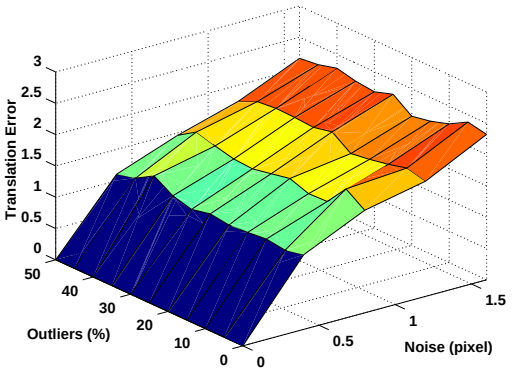

**(a)** Success Rate Hom.



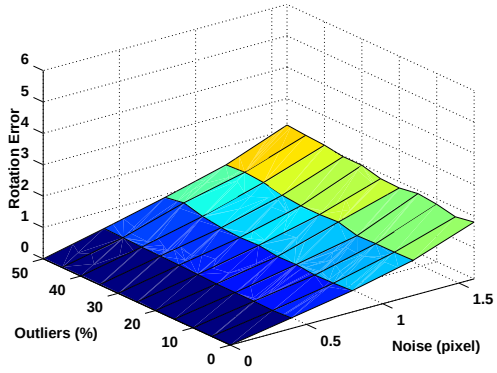

**(b)** Success Rate P3P



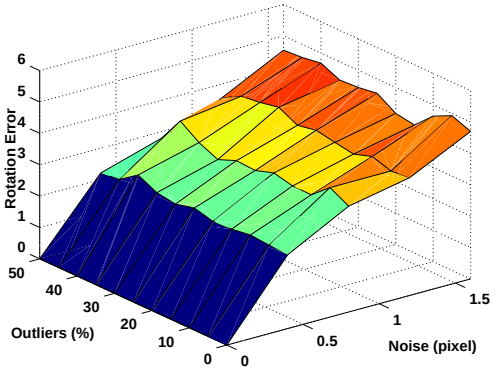

**(c)** Translation Error Hom.




**(d)** Translation Error P3P



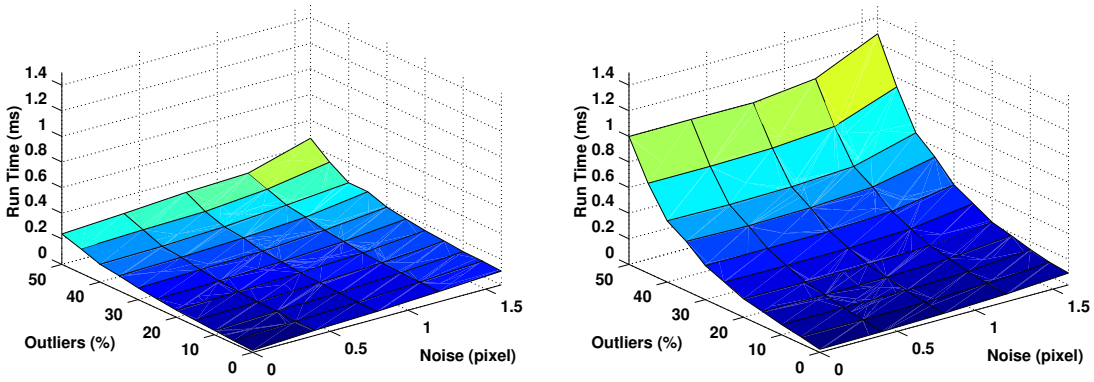

**(e)** Rotation Error Hom.




**(f)** Rotation Error P3P

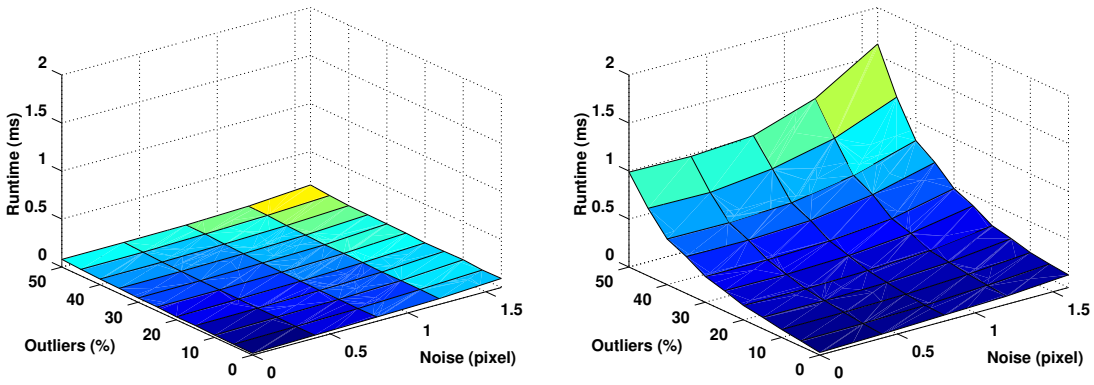
**Figure 4.9:** Success rate, median translation error and median rotation error of both the homography and P3P method if only one side of the object is visible. Without optimization.

**Optimization: Accuracy** We already showed previously that all optimization methods are reducing the error significantly with the results being very similar to each other. We now want to have a closer look into their accuracy. We run our test over all noise and




**(a)** Homographies **(b)** P3P

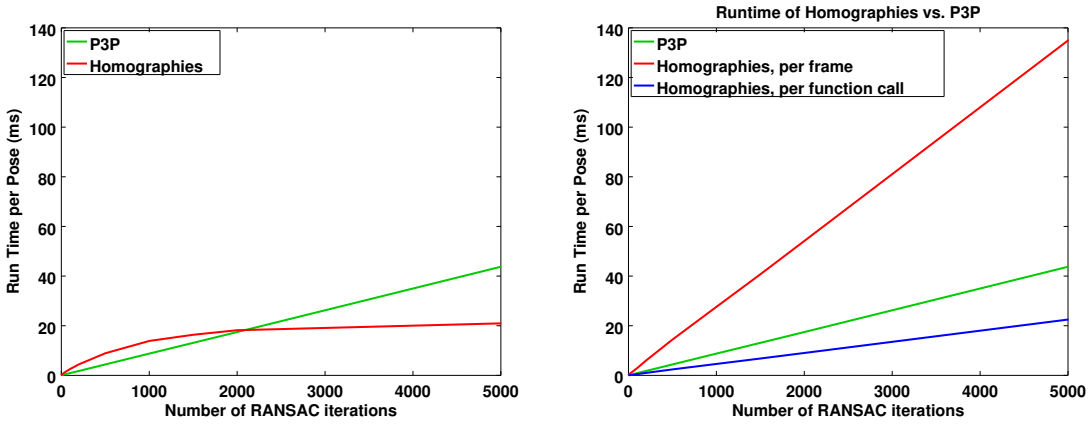
**Figure 4.10:** We tested the influence of noise and outliers on the run time of homographies and P3P.




**(a)** Homographies **(b)** P3P

**Figure 4.11:** The influence of noise and outliers on the run time of homographies and P3P for scenes where only one side of the object was visible to the camera.

outlier levels and build the sum of residuals, considering only RANSAC inliers. Like this, we compare the result of the error function before and after optimization. The results are shown in Table 4.2. The Ceres Solver functions as well as our Gauss-Newton implementation are close to their minimum after only two iterations. Levenberg-Marquardt is known to potentially have slower convergence than Gauss-Newton, which might as well be a result of us using a relatively simple implementation of it. While Gauss-Newton can potentially even diverge if the starting point is far from the minimum, this does not seem to be a problem in our case, the pose estimates seem to be a good starting point that is close enough to the minimum already.




**(a)** Hom. vs. P3P **(b)** no iteration recalculation

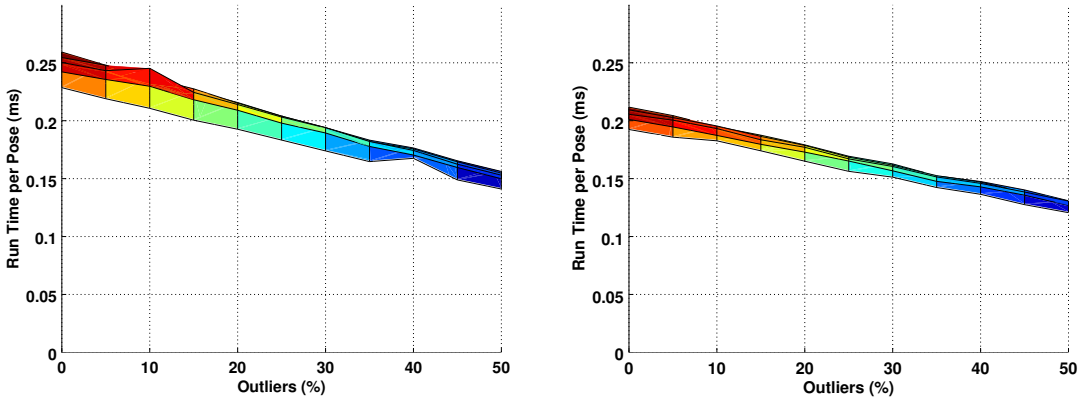
**Figure 4.12:** The run time of homographies and P3P on random points. Once with OpenCV's RANSAC limiting the number of iterations, once not.

**Optimization: Efficiency** Now that we know, that optimization only requires very few steps, we would like to compare their run time, so we run the same tests for two iteration steps. It quickly becomes apparent, that the run time of the Ceres Solver functions is a multiple of the self implemented ones, so the Ceres Solver's run time is shown separately in Figure 4.13 with a different scaling. Even though Ceres Solver is for sure highly efficient, this happens most likely due to the fact that we only need very few iterations and the data has to be converted to fit into the Ceres Solver functions, while our own implementations are able to use data types that are very efficient for our tasks.

Apparently, the run time for the optimization step is lower, the higher the amount of outliers is. This makes sense as RANSAC should by now already have removed all/most outliers and the optimization step has fewer point correspondences to deal with. Overall,

| # iterations | CS automatic | CS analytic | GN | GN + Lie | LM |
|---|---|---|---|---|---|
| 0 | 1.5374 | 1.5374 | 1.5374 | 1.5374 | 1.5374 |
| 1 | 0.7448 | 0.7448 | 0.7449 | 0.7441 | 0.8472 |
| 2 | 0.7426 | 0.7426 | 0.7426 | 0.7426 | 0.7698 |
| 3 | 0.7426 | 0.7425 | 0.7426 | 0.7426 | 0.7491 |
| 5 | 0.7426 | 0.7425 | 0.7426 | 0.7426 | 0.7428 |
| 10 | 0.7426 | 0.7425 | 0.7426 | 0.7426 | 0.7426 |
| 100 | 0.7426 | 0.7425 | 0.7426 | 0.7426 | 0.7426 |

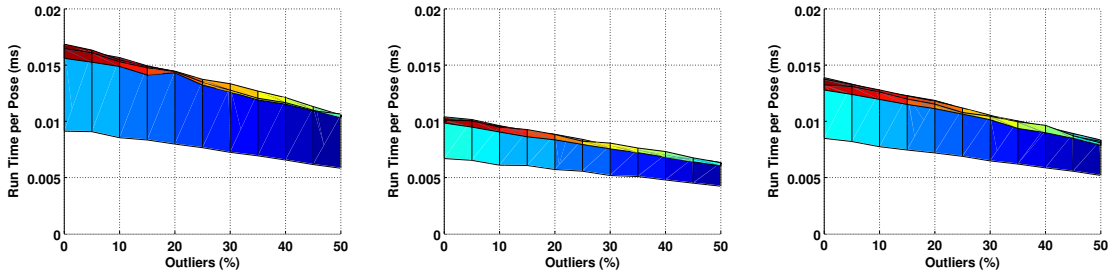
**Table 4.2:** The sum over the residuals over 1000 poses for different optimization options: Ceres Solver with automatic derivatives, Ceres Solver with analytic derivatives, Gauss-Newton with derivatives of the R matrix, Gauss-Newton with Lie groups, Levenberg-Marquardt with Lie groups. All but Levenberg-Marquardt come very close to their minimum in only one iteration step.




**(a)** Automatic Derivatives **(b)** Analytic Derivatives

**Figure 4.13:** The run time of the Ceres Solver functions for a maximum of 2 iteration steps. The blue lower values in the "background" occur for noise-free test data.

the Ceres Solver function with analytic derivatives is slightly faster than Ceres Solver with automatic derivatives.



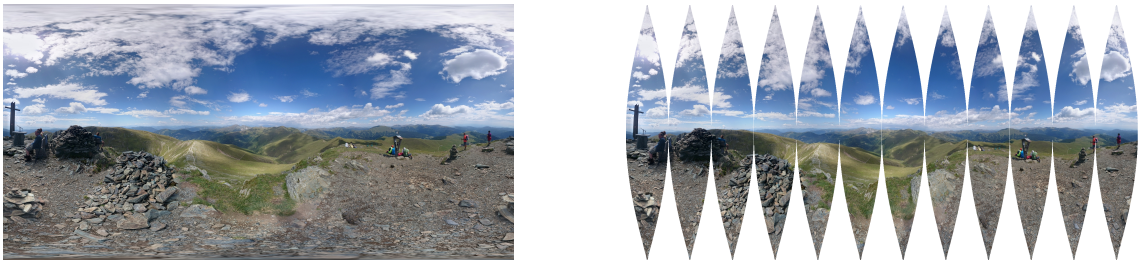

**(a)** GN **(b)** GN + Lie **(c)** LM

**Figure 4.14:** The run time of the optimization step for Gauss-Newton method with derivatives of matrix R, Gauss-Newton method with Lie groups, and Levenberg-Marquardt with Lie groups. The lower "background" values occur for noise-free test data.

In Figure 4.14, we then compare the run time of Gauss-Newton method with derivatives of matrix R, Gauss-Newton method with Lie groups, and Levenberg-Marquardt with Lie groups. Again, they clearly depend on the number of outliers and therefore the number of point correspondences. Although Levenberg-Marquardt has a similar performance to Gauss-Newton, keeping in mind that it needed more iterations to receive the same residual, its run time is unfavorable. Gauss-Newton on the other hand is doing well and clearly profits from replacing the cumbersome derivatives of the rotation matrix R by Lie groups.

# 5 Practical Application

It turns out difficult to find ready-made well-textured spheres. So in order to test our code on spheres, we use the mapping function described in Chapter 3 to convert any cylindrical sphere mapping to 3D and then to gore mappings, which enables us to print them out and craft them. This gives us access to a variety of spherical objects from regular 360 degree panoramic images (as seen in Figure 5.1) without interruptions along the gores, as it would happen if we would just cut them from the images directly. Since they of course cannot be perfectly round spheres, we accept some distortion: for our preferred number of 12 gores, the distortion is at its maximum at 3.4 % of the radius (for a sphere diameter of 10 cm, that would be 1.7 mm at its maximum).



**(a)** Input image **(b)** Output image

**Figure 5.1:** We use a tool to create sphere gores from cylindrical projections.

## 5.1 Desktop Computer

We test our code on real pictures using OpenCV's Scale-Invariant Feature Transform (SIFT) and Oriented FAST and Rotated BRIEF (ORB) features. As a rule of thumb, Fraundorfer and Scaramuzza [21] suggest using 1000 features for 640 x 480-pixel images. Although the resolution we use was 1920 x 1080 pixel, we as well found 1000 features to

work best. In order to get more distinct features, we use a k-nearest neighbor (KNN) matcher to get the two best matches and compare them by Lowe's ratio test [41]: if the best match is not significantly better than the second-best, reject it.

We run our code on a desktop computer with an Intel Core i7-4771 @3.50GHz processor and 8 GB of RAM. As library versions we use OpenCV 3.4.1, Ceres Solver 1.13.0, Eigen 3.3.4, as well as Sophus library. As shown in Figure 5.2, feature detection and feature matching take over the biggest shares of the total run time. While our improvements were well visible on our artificial data, using OpenCV feature detection and feature matching lets the effect vanish. In total, we achieve frame rates of 15 fps for ORB features and 6 fps for SIFT features. Examples of the results including feature matches can be seen in Figures 5.3 and 5.4.

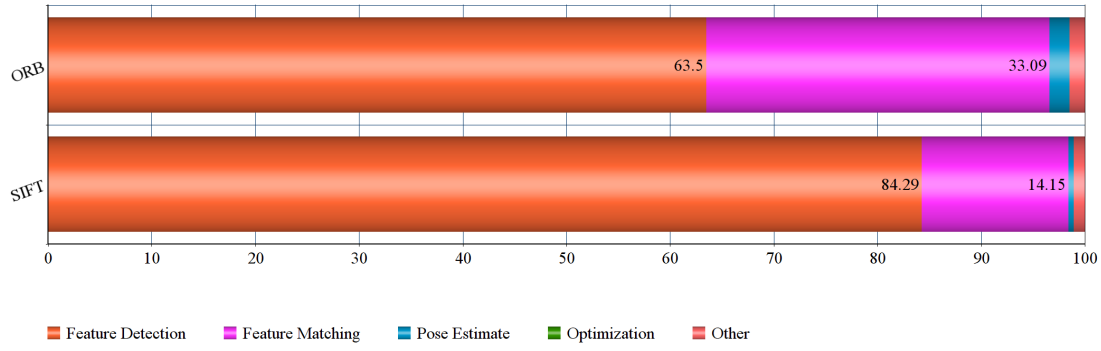



**Figure 5.2:** The shares of the total run time for both SIFT and ORB features: using SIFT features the pose estimation share is almost vanishing (0.5 %), for SIFT features it is at less than 2 %.

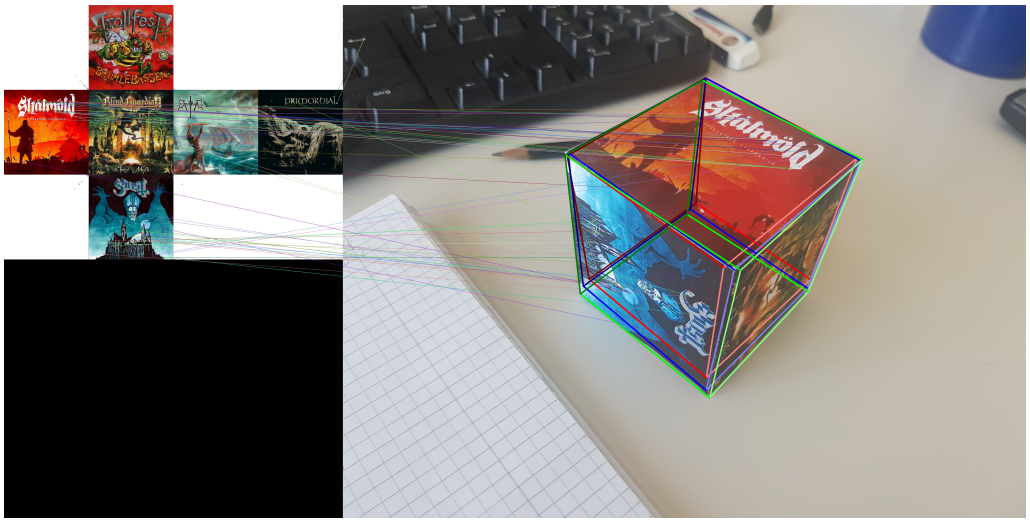



**Figure 5.3:** Results for a cube: red outlines show the pose achieved via homographies, blue outlines show the refined pose using all Random Sample Consensus (RANSAC) inliers, green outlines show the final result.

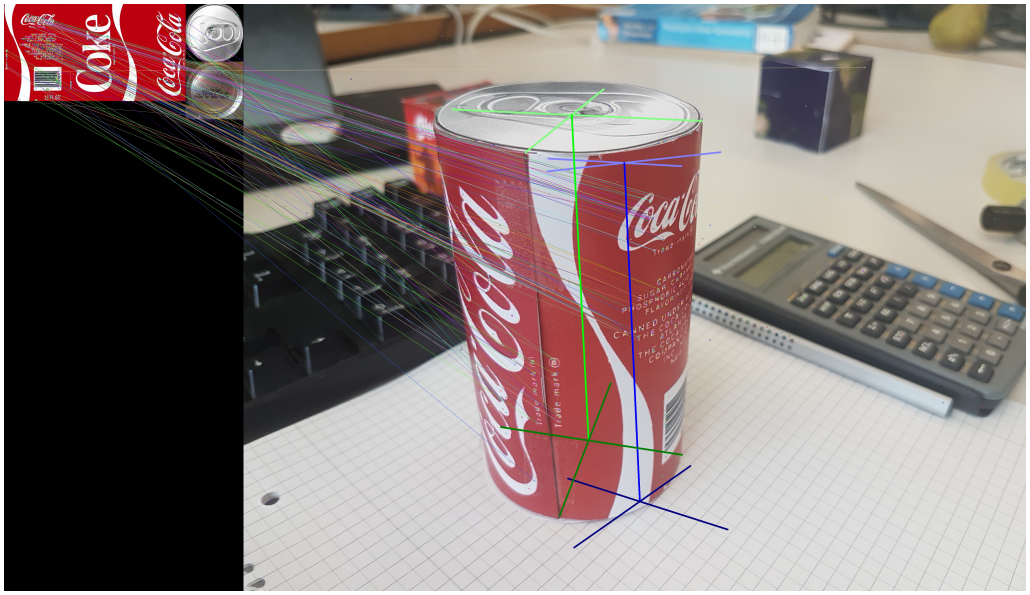



**Figure 5.4:** Results for a cylinder: blue lines show the initial pose calculated via P3P, green lines show the refined pose.

## 5.2 Mobile Device

Finally, we bring our code to mobile devices and present an Android application. We use Android Studio 3.1 to develop it, and include OpenCV 3.4.1 library and its xfeatures2d classes for SIFT and SURF features, Eigen library 3.3.4, as well as Sophus library [70]. We integrate our C++ code from the desktop version as native code using the Android Native Development Kit (Android NDK).

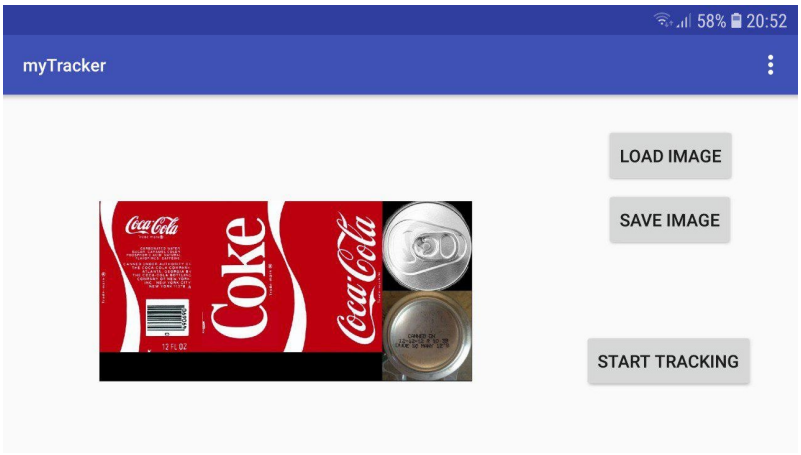



**Figure 5.5:** Android application: start screen.

We use a minimalist start screen to handle saved templates, as can be seen in Figure 5.5 and add a settings menu (Figure 5.6) to easily change our parameters, e.g. the type of features we want to use or whether or not to use homographies when tracking cuboids. A screen shot of the tracking screen can be seen in Figure 5.7. In Figures 5.8 and 5.9 we show a section of a video, where we can see how the refined pose fits well while the unrefined pose jumps from frame to frame.

For the optimization step we always use our implementation for Gauss-Newton in combination with Lie groups, as this has proven to be the most efficient option. We are testing our setup on a Samsung Galaxy S7, featuring a Samsung Exynos 8890 Octa CPU and 4 GB of memory. Similar to what we observed on the Desktop computer, the bottleneck for the run time is again the feature detection. Using ORB features, we achieve 6 fps, using SIFT features our frame rate goes down to around 1 fps.

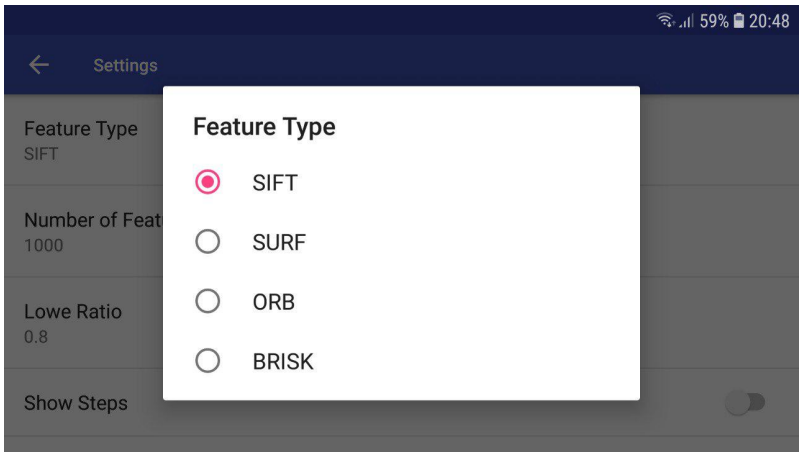



**Figure 5.6:** Android application: settings menu.

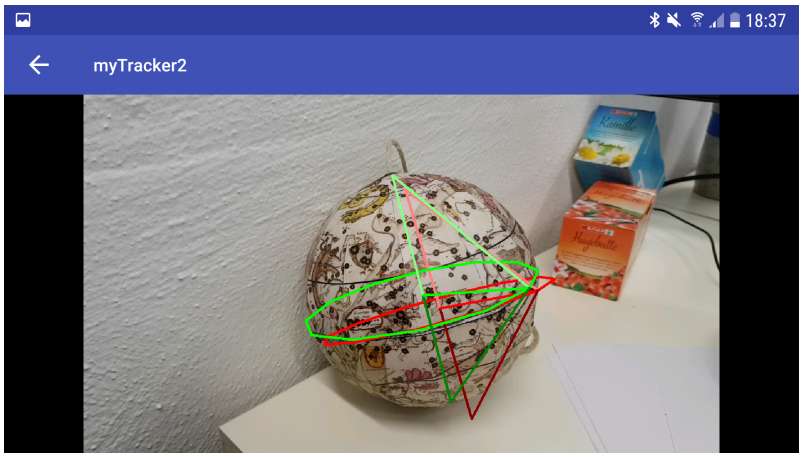



**Figure 5.7:** Android application: the tracking screen. The red lines indicate the pose before the optimization step (we draw the equatorial line, the axis and a triangle for orientation), the green lines indicate the pose including the optimization step.

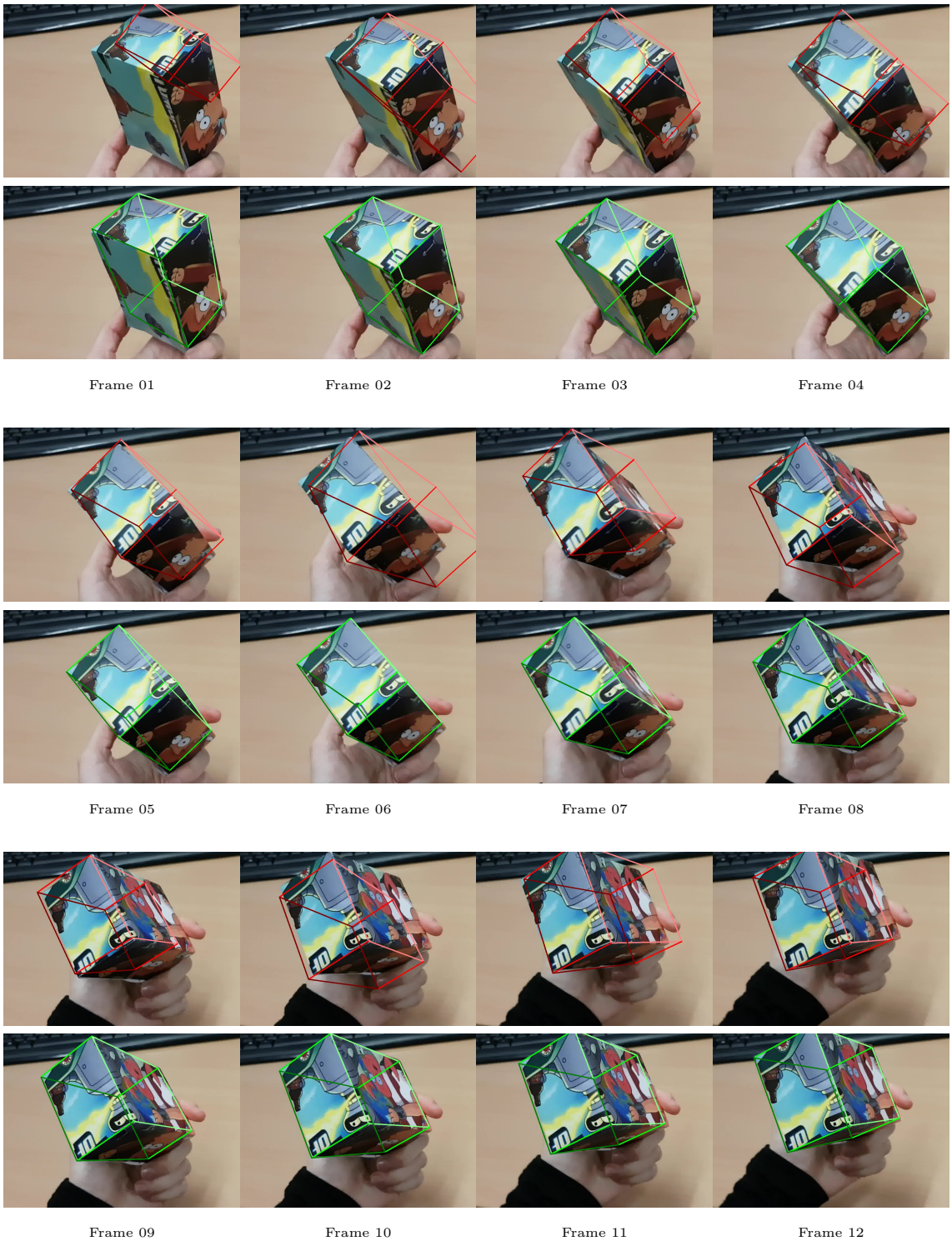
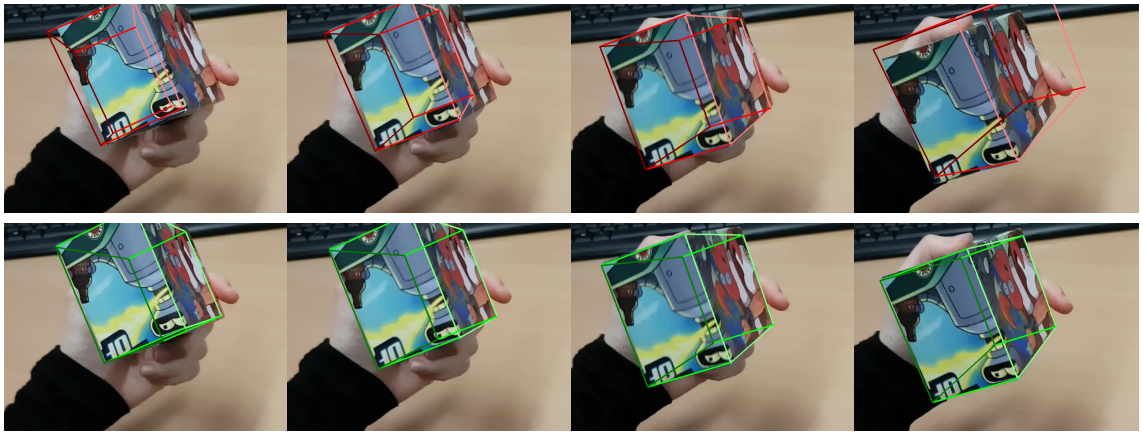
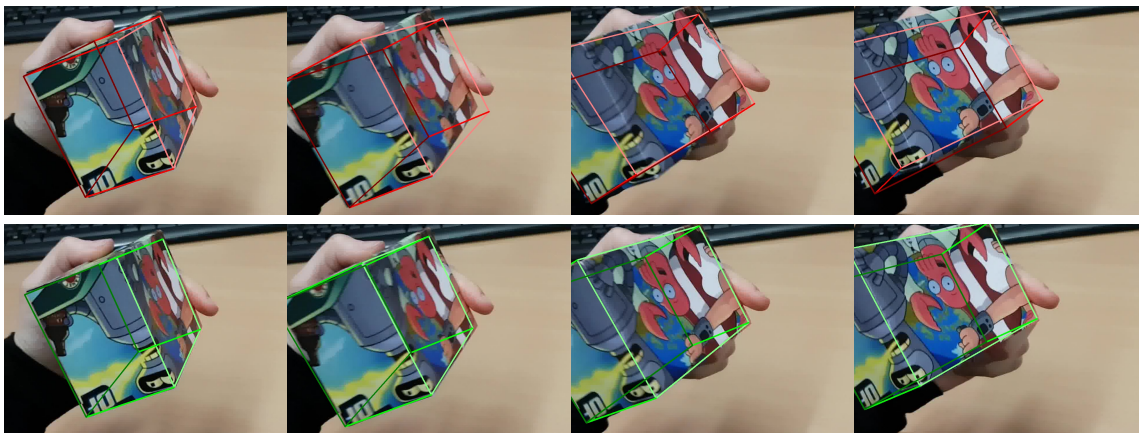


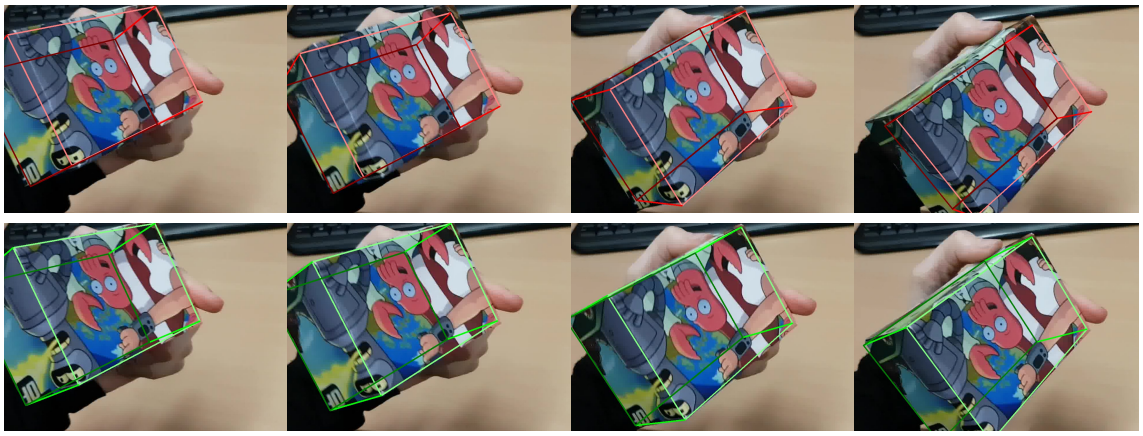

**Figure 5.8:** A series of frames taken from a video: the red bounding box indicates the pose before the optimization step, the green bounding box after optimization step. (The images have been trimmed for better visibility.)



Frame 13 Frame 14 Frame 15 Frame 16



Frame 17 Frame 18 Frame 19 Frame 20



Frame 21 Frame 22 Frame 23 Frame 24

**Figure 5.9:** Continuation of the series of frames in Figure 5.8.

# 6 Conclusion and Future Work

## 6.1 Conclusion

Comparing the homography method with P3P, we saw that for coplanar 3D points, homographies will be more accurate and much faster than P3P. P3P on the other hand has its major strength of being able to use all point correspondences on an object at once. P3P will hardly suffer from not having enough points to work with and therefore hardly fail completely. Which one to pick highly depends on the model we want to track: if we expect one side to be much more prominent than others or more uniquely structured than the other sides (e.g. a book cover), homographies clearly are the way to go. If we expect to only get a few good feature correspondences per side of an object, using P3P makes the most sense. It is needless to say, that P3P is the best way for all non-polyhedral objects.

For the optimization our major interest was to determine if using a ready-made library function is the best choice. While one might expect them to have highly optimized code and therefore much better performance, it turned out that the necessary data conversion to feed our problem into its functions is much more expensive than the optimization step itself. Their run time was approximately 10 times higher than that of our own implementations. Even though library functions definitely are convenient for a prototype, we conclude that in order to reduce the run time it is highly recommendable to use own implementations. Not to mention, that this makes porting the project to a mobile system a lot easier. While the Levenberg-Marquardt method is usually considered a successor to the Gauss-Newton method, we could not see any benefit in our case, as the Gauss-Newton method had excellent results as well, converged after fewer iteration steps and had faster iteration steps. We therefore do not see a need for more effort than Gauss-Newton optimization.

Looking at the distribution of run time in general however, we saw that the biggest issues of pose estimation still are feature detection and feature matching. While a good

choice for a first pose estimate and for optimization improves their step significantly, they are still hardly changing the overall run time in general.

## 6.2 Future Work

The probably most important task for the future would be to find more efficient feature detection and feature matching functions than those of OpenCV, as they clearly showed to be the bottleneck in our implementation.

Another obvious step for the future would be to extend the tracking to incremental tracking: while for now we only looked into initial pose estimation without any prior knowledge of the scene, we could use a given pose of the last frame to look for features only in the vicinity of their last position, which would reduce the workload on it significantly.

Furthermore, it could be interesting to try optimization on the parameters of our implementation. These parameters include Random Sample Consensus (RANSAC) parameters (iteration count, reprojection error, confidence value), orthogonality threshold, various cost functions and their tuning constants, the number of optimization iterations or convergence thresholds. An easy way to do that would probably be evolution strategies, where the parameter vector gets mutated by adding normally distributed vectors to it and we check whether or not that improved our results.

# A

# Derivatives of Rotation Matrix R

Here we show the Jacobian matrix of the error function

$$E = \sum_{i=1}^{n} ||K^{-1}x - [R\,|\,t] \begin{bmatrix} X \\ 1 \end{bmatrix} ||^2.$$

$$\theta = \sqrt{r_1^2 + r_2^2 + r_3^2}$$

$$\begin{aligned}
J(0,0) = & \frac{\partial R_{11}}{\partial r_1} X + \frac{\partial R_{12}}{\partial r_1} Y + \frac{\partial R_{13}}{\partial r_1} Z \\
= & [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1^3}{\theta^4} + \frac{2r_1}{\theta^2}) + \frac{r_1^3 \sin\theta}{\theta^3} - \frac{r_1 \sin\theta}{+} \cos\theta] X \\
& + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1^2 r_2}{\theta^4} + \frac{r_2}{\theta^2}) + \cos\theta \\
& + \frac{r_1^2 r_2 \sin\theta}{\theta^3} - \frac{r_1 r_3 \cos\theta}{\theta^2} + \frac{r_1 r_3 \sin\theta}{\theta^3}] Y \\
& + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1^2 r_3}{\theta^4} + \frac{r_3}{\theta^2}) + \cos\theta \\
& + \frac{r_1^2 r_3 \sin\theta}{\theta^3} + \frac{r_1 r_2 \cos\theta}{\theta^2} - \frac{r_1 r_2 \sin\theta}{\theta^3}] Z
\end{aligned}$$

$$
\begin{aligned}
J(0,1) &= \frac{\partial R_{11}}{\partial r_2} + \frac{\partial R_{12}}{\partial r_2} + \frac{\partial R_{13}}{\partial r_2} \\
&= [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1^2 r_2}{\theta^4}) + \cos\theta + \frac{r_1^2 r_2 \sin\theta}{\theta^3} - \frac{r_2 \sin\theta}{\theta}]X \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1 r_2^2}{\theta^4} + \frac{r_1}{\theta^2}) + \cos\theta \\
&\quad + \frac{r_1 r_2^2 \sin\theta}{\theta^3} - \frac{r_2 r_3 \cos\theta}{\theta^2} + \frac{r_2 r_3 \sin\theta}{\theta^3}]Y \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1 r_2 r_3}{\theta^4}) + \cos\theta \\
&\quad + \frac{r_1 r_2 r_3}{\theta^3}\sin\theta + \frac{r_2^2 \cos\theta}{\theta^2} - \frac{r_2^2 \sin\theta}{\theta^3} + \frac{\sin\theta}{\theta}]Z
\end{aligned}
$$

$$
\begin{aligned}
J(0,2) &= \frac{\partial R_{11}}{\partial r_3} + \frac{\partial R_{12}}{\partial r_3} + \frac{\partial R_{13}}{\partial r_3} \\
&= [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1^2 r_3}{\theta^4}) + \cos\theta + \frac{r_1^2 r_3 \sin\theta}{\theta^3} - \frac{r_3 \sin\theta}{\theta}]X \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1 r_2 r_3}{\theta^4}) + \cos\theta \\
&\quad + \frac{r_1 r_2 r_3}{\theta^3}\sin\theta - \frac{r_3^2 \cos\theta}{\theta^2} + \frac{r_3^2 \sin\theta}{\theta^3} - \frac{\sin\theta}{\theta}]Y \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1 r_3^2}{\theta^4} + \frac{r_1}{\theta^2}) + \cos\theta \\
&\quad + \frac{r_1 r_3^2 \sin\theta}{\theta^3} + \frac{r_2 r_3 \cos\theta}{\theta^2} - \frac{r_2 r_3 \sin\theta}{\theta^3}]Z
\end{aligned}
$$

$$
\begin{aligned}
J(1,0) &= \frac{\partial R_{21}}{\partial r_1} + \frac{\partial R_{22}}{\partial r_1} + \frac{\partial R_{23}}{\partial r_1} \\
&= [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1^2 r_2}{\theta^4} + \frac{r_2}{\theta^2}) + \cos\theta \\
&\quad + \frac{r_1^2 r_2 \sin\theta}{\theta^3} + \frac{r_1 r_3 \cos\theta}{\theta^2} - \frac{r_1 r_3 \sin\theta}{\theta^3}]X \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1 r_2^2}{\theta^4}) + \cos\theta + \frac{r_1 r_2^2 \sin\theta}{\theta^3} - \frac{r_1 \sin\theta}{\theta}]Y \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1 r_2 r_3}{\theta^4}) + \cos\theta \\
&\quad - \frac{r_1^2 \cos\theta}{\theta^2} + \frac{r_1^2 \sin\theta}{\theta^3} + \frac{r_1 r_2 r_3}{\theta^3}\sin\theta - \frac{\sin\theta}{\theta}]Z
\end{aligned}
$$

$$
\begin{aligned}
J(1,1) =& \frac{\partial R_{21}}{\partial r_2} + \frac{\partial R_{22}}{\partial r_2} + \frac{\partial R_{23}}{\partial r_2} \\
=& [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1r_2^2}{\theta^4} + \frac{r_1}{\theta^2}) + \cos\theta \\
&+ \frac{r_1r_2^2\sin\theta}{\theta^3} + \frac{r_2r_3\cos\theta}{\theta^2} - \frac{r_2r_3\sin\theta}{\theta^3}]X \\
&+ [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_2^3}{\theta^4} + \frac{2r_2}{\theta^2}) + \cos\theta \\
&+ \frac{r_2^3\sin\theta}{\theta^3} - \frac{r_2\sin\theta}{\theta}]Y \\
&+ [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_2^2r_3}{\theta^4} + \frac{r_3}{\theta^2}) + \cos\theta \\
&- \frac{r_1r_2\cos\theta}{\theta^2} + \frac{r_1r_2\sin\theta}{\theta^3} + \frac{r_2^2r_3\sin\theta}{\theta^3}]Z
\end{aligned}
$$

$$
\begin{aligned}
J(1,2) =& \frac{\partial R_{21}}{\partial r_3} + \frac{\partial R_{22}}{\partial r_3} + \frac{\partial R_{23}}{\partial r_3} \\
=& [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1r_2r_3}{\theta^4}) + \cos\theta \\
&+ \frac{r_1r_2r_3}{\theta^3}\sin\theta + \frac{r_3^2\cos\theta}{\theta^2} - \frac{r_3^2\sin\theta}{\theta^3} + \frac{\sin\theta}{\theta}]X \\
&+ [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_2^2r_3}{\theta^4}) + \cos\theta + \frac{r_2^2r_3\sin\theta}{\theta^3} - \frac{r_3\sin\theta}{\theta}]Y \\
&+ [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_2r_3^2}{\theta^4} + \frac{r_2}{\theta^2}) + \cos\theta \\
&- \frac{r_1r_3\cos\theta}{\theta^2} + \frac{r_1r_3\sin\theta}{\theta^3} + \frac{r_2r_3^2\sin\theta}{\theta^3}]Z
\end{aligned}
$$

$$
\begin{aligned}
J(2,0) =& \frac{\partial R_{31}}{\partial r_1} + \frac{\partial R_{32}}{\partial r_1} + \frac{\partial R_{33}}{\partial r_1} \\
=& [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1^2r_3}{\theta^4} + \frac{r_3}{\theta^2}) + \cos\theta \\
&+ \frac{r_1^2r_3\sin\theta}{\theta^3} - \frac{r_1r_2\cos\theta}{\theta^2} + \frac{r_1r_2\sin\theta}{\theta^3}]X \\
&+ [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1r_2r_3}{\theta^4}) + \cos\theta \\
&\frac{r_1^2\cos\theta}{\theta^2} - \frac{r_1^2\sin\theta}{\theta^3} + \frac{r_1r_2r_3}{\theta^3}\sin\theta + \frac{\sin\theta}{\theta}]Y \\
&+ [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1r_3^2}{\theta^4}) + \cos\theta + \frac{r_1r_3^2\sin\theta}{\theta^3} - \frac{r_1\sin\theta}{\theta}]Z
\end{aligned}
$$

$$
\begin{aligned}
J(2,1) &= \frac{\partial R_{31}}{\partial r_2} + \frac{\partial R_{32}}{\partial r_2} + \frac{\partial R_{33}}{\partial r_2} \\
&= [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1r_2r_3}{\theta^4}) + \cos\theta \\
&\quad + \frac{r_1r_2r_3}{\theta^3}\sin\theta - \frac{r_2^2\cos\theta}{\theta^2} + \frac{r_2^2\sin\theta}{\theta^3} - \frac{\sin\theta}{\theta}]X \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_2^2r_3}{\theta^4} + \frac{r_3}{\theta^2}) + \cos\theta \\
&\quad \frac{r_1r_2\cos\theta}{\theta^2} - \frac{r_1r_2\sin\theta}{\theta^3} + \frac{r_2^2r_3\sin\theta}{\theta^3}]Y \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_2r_3^2}{\theta^4}) + \cos\theta + \frac{r_2r_3^2\sin\theta}{\theta^3} - \frac{r_2\sin\theta}{\theta}]Z
\end{aligned}
$$

$$
\begin{aligned}
J(2,2) &= \frac{\partial R_{31}}{\partial r_3} + \frac{\partial R_{32}}{\partial r_3} + \frac{\partial R_{33}}{\partial r_3} \\
&= [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_1r_3^2}{\theta^4} + \frac{r_1}{\theta^2}) + \cos\theta \\
&\quad + \frac{r_1r_3^2\sin\theta}{\theta^3} - \frac{r_2r_3\cos\theta}{\theta^2} + \frac{r_2r_3\sin\theta}{\theta^3}]X \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_2r_3^2}{\theta^4} + \frac{r_2}{\theta^2}) + \cos\theta \\
&\quad \frac{r_1r_3\cos\theta}{\theta^2} - \frac{r_1r_3\sin\theta}{\theta^3} + \frac{r_2r_3^2\sin\theta}{\theta^3}]Y \\
&\quad + [(1-\cos\theta)(\frac{r_1^2}{\theta^2} - \frac{2r_3^3}{\theta^4} + \frac{2r_3}{\theta^2}) + \cos\theta + \frac{r_3^3\sin\theta}{\theta^3} - \frac{r_3\sin\theta}{\theta}]Z
\end{aligned}
$$

$$
J(0:2,3:5) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}
$$

# A

# List of Acronyms

| | |
|---|---|
| AR | Augmented Reality |
| DLT | Direct Linear Transformation |
| DOF | Degrees of Freedom |
| fps | frames per second |
| IMU | Inertial Measurement Unit |
| KNN | k-nearest neighbor |
| ORB | Oriented FAST and Rotated BRIEF |
| PnP | Perspective-$n$-Point |
| RANSAC | Random Sample Consensus |
| SIFT | Scale-Invariant Feature Transform |
| SURF | Speeded Up Robust Features |
| VR | Virtual Reality |

# B

# List of Definitions

**Cost Function** A function that gives some sort of penalty to bigger error values. A common example is least squares, where the error is squared. Some sources refer to it as loss function.

**Error Function** An error function (or objective function) is the sum over all residuals and cost functions applied to them:

$$F(x) = \frac{1}{2} \sum_i \rho(f_i(x)),$$

with $\rho$ being a cost function and $f_i(x)$ a residual.

**Developable Surface** Surfaces that can be mapped to 2D without any distortion are called "developable". Examples would be cubes, cylinders or cones. A counterexample would be a sphere.

**Homography** A projective transformation H from projective space $P^2$ to $P^2$.

**Jacobian** The partial derivatives of a multi-variate function. Needed to find minima for our error function.

**PnP** A variety of methods to retrieve a pose from n point correspondences.

**P3P** Generally speaking, all methods to retrieve a pose from three points. However, we specifically refer to the P3P method by Gao et al. [23].

**Point Correspondence** A set of a 2D point in an image and a 3D point on the model.

**Pose** A pose refers to the combination of calibration matrix K, a camera's position in world space and its orientation.

$$P = K[R \,|\, t]$$

Since we assume the camera matrix K to be known, that leaves only R and t do determine, which have 6 dof.

**RANSAC** A popular method to fit a data model to data points that contain a significant amount of outliers.

**Residual** The image space error per point correspondence, i.e. the difference between a predicted point and an observe point.

# Bibliography

[1] Agarwal, S., Mierle, K., and et al. (2012). Ceres Solver. (page 5, 32)

[2] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer. (page 9)

[3] Bern, M., Demaine, E. D., Eppstein, D., Kuo, E., Mantler, A., and Snoeyink, J. (2003). Ununfoldable polyhedra with convex faces. *Computational Geometry*, 24(2):51–62. (page 23)

[4] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. (page 29)

[5] Briales, J. and Gonzalez-Jimenez, J. (2017). Convex global 3d registration with lagrangian duality. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5612–5621. IEEE. (page 16)

[6] Chum, O. and Matas, J. (2005). Matching with prosac-progressive sample consensus. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 220–226. IEEE. (page 14)

[7] Crivellaro, A., Rad, M., Verdie, Y., Yi, K. M., Fua, P., and Lepetit, V. (2015). A Novel Representation of Parts for Accurate 3d Object Detection and Tracking in Monocular Images. pages 4391–4399. IEEE. (page 9)

[8] DeMenthon, D. F. and Davis, L. S. (1992). Model-based object pose in 25 lines of code. In *European conference on computer vision*, pages 335–343. Springer. (page 12)

[9] Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15–16):1–35. (page 17)

[10] Drummond, T. and Cipolla, R. (2000). Application of lie algebras to visual servoing. *International Journal of Computer Vision*, 37(1):21–41. (page 20)

[11] Drummond, T. and Cipolla, R. (2002). Real-time visual tracking of complex structures. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):932–946. (page 20)

[12] Eade, E. (2013). Gauss-Newton / Levenberg-Marquardt Optimization. page 9. (page 34)

[13] Eade, E. (2017). Lie groups for 2d and 3d transformations. page 25. (page 20)

[14] Engel, J., Schöps, T., and Cremers, D. (2014). Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, page 834–849. Springer. (page 10)

[15] Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A. (1997). A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. *Personal Technologies*, 1(4):208–217. (page 7)

[16] Ferraz, L., Binefa, X., and Moreno-Noguer, F. (2014). Leveraging Feature Uncertainty in the PnP Problem. In *BMVC*. (page 39)

[17] Fiala, M. (2010). Designing highly reliable fiducial markers. *IEEE Transactions on Pattern analysis and machine intelligence*, 32(7):1317–1324. (page 21)

[18] Fischler, M. A. and Bolles, R. C. (1987). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pages 726–740. Elsevier. (page 4, 5, 11, 13, 14)

[19] Fletcher, R. (1971). Modified marquaredt subroutine for nonlinear least square. *Harwell Report AERE-R*, 6799. (page 15)

[20] Fox, J. (2002). Robust regression. *An R and S-Plus companion to applied regression*. (page 16)

[21] Fraundorfer, F. and Scaramuzza, D. (2012). Visual odometry: Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90. (page 14, 51)

[22] Gallego, G. and Yezzi, A. (2015). A compact formula for the derivative of a 3-D rotation in exponential coordinates. *Journal of Mathematical Imaging and Vision*, 51(3):378–384. (page 18)

[23] Gao, X.-S., Hou, X.-R., Tang, J., and Cheng, H.-F. (2003). Complete solution classification for the perspective-three-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 25(8):930–943. (page 13, 30, 67)

[24] Gokturk, S. B., Yalcin, H., and Bamji, C. (2004). A time-of-flight depth sensor-system description, issues and solutions. In *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW'04. Conference on*, pages 35–35. IEEE. (page 9)

[25] Hallaway, D., Feiner, S., and Höllerer, T. (2004). Bridging the gaps: Hybrid tracking for adaptive mobile augmented reality. *Applied Artificial Intelligence*, 18(6):477–500. (page 22)

[26] Haralick, R. M. and Shapiro, L. G. (1992). *Computer and robot vision*. Addison-wesley. (page 15)

[27] Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press. (page 13, 31)

[28] Hartley, R. I. (1998). Minimizing algebraic error. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 356(1740):1175–1192. (page 13)

[29] Hoff, W. A., Nguyen, K., and Lyon, T. (1996). Computer Vision-Based Registration Techniques for Augmented Reality. In *Intelligent Robots and Computer Vision XV*, pages 538–548. (page 9)

[30] Kato, H. and Billinghurst, M. (1999). Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999.(IWAR'99) Proceedings. 2nd IEEE and ACM International Workshop on*, pages 85–94. IEEE. (page 9)

[31] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, page 225–234. IEEE. (page 10, 20)

[32] Klein, G. and Murray, D. (2008). Improving the agility of keyframe-based slam. In *European Conference on Computer Vision*, page 802–815. Springer. (page 10)

[33] Kneip, L., Scaramuzza, D., and Siegwart, R. (2011). A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. (page 13)

[34] Koller, D., Klinker, G., Rose, E., Breen, D., Whitaker, R., and Tuceryan, M. (1997). Real-time vision-based camera tracking for augmented reality applications. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 87–94. ACM. (page 9)

[35] Lee, G. A., Dünser, A., Kim, S., and Billinghurst, M. (2012). CityViewAR: A mobile outdoor AR application for city visualization. In *Mixed and Augmented Reality (ISMAR-AMH), 2012 IEEE International Symposium on*, pages 57–64. IEEE. (page 7)

[36] Lepetit, V. and Fua, P. (2005). Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends® in Computer Graphics and Vision*, 1(1):1–89. (page 17)

[37] Lepetit, V. and Fua, P. (2006). Keypoint recognition using randomized trees. *IEEE transactions on pattern analysis and machine intelligence*, 28(9):1465–1479. (page 22)

[38] Lepetit, V., Moreno-Noguer, F., and Fua, P. (2009). Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155. (page 13)

[39] Li, S., Xu, C., and Xie, M. (2012). A Robust O(n) Solution to the Perspective-n-Point Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1444–1450. (page 13)

[40] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee. (page 2)

[41] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110. (page 9, 21, 52)

[42] Lu, C.-P., Hager, G. D., and Mjolsness, E. (2000). Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):610–622. (page 12, 13, 15)

[43] Lucier, B. (2006). Unfolding and reconstructing polyhedra. Master's thesis, University of Waterloo. (page 23)

[44] Madsen, K., Nielsen, H. B., and Tingleff, O. (1999). Methods for non-linear least squares problems. (page 15)

[45] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441. (page 15)

[46] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163. (page 10)

[47] Naimark, L. and Foxlin, E. (2002). Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, page 27. IEEE Computer Society. (page 9)

[48] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE. (page 9)

[49] Nistér, D. (2005). Preemptive ransac for live structure and motion estimation. *Machine Vision and Applications*, 16(5):321–329. (page 14)

[50] Obeidy, W. K., Arshad, H., Chowdhury, S. A., Parhizkar, B., and Huang, J. (2013). Increasing the tracking efficiency of mobile augmented reality using a hybrid tracking technique. In *International Visual Informatics Conference*, pages 447–457. Springer. (page 8)

[51] Oberkampf, D., DeMenthon, D. F., and Davis, L. S. (1996). Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, 63(3):495–511. (page 12)

[52] Olsson, C. and Eriksson, A. (2008). Solving quadratically constrained geometrical problems using lagrangian duality. In *2008 19th International Conference on Pattern Recognition*, page 1–5. IEEE. (page 16)

[53] Onyesolu, M. O., Ezeani, I., and Okonkwo, O. R. (2012). A Survey of Some Virtual Reality Tools and Resources. In *Virtual Reality and Environments*. InTech. (page 7, 8)

[54] Ozuysal, M., Calonder, M., Lepetit, V., and Fua, P. (2010). Fast keypoint recognition using random ferns. *IEEE transactions on pattern analysis and machine intelligence*, 32(3):448–461. (page 22)

[55] Ozuysal, M., Fua, P., and Lepetit, V. (2007). Fast keypoint recognition in ten lines of code. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. Ieee. (page 2)

[56] Pauwels, K., Rubio, L., Diaz, J., and Ros, E. (2013). Real-Time Model-Based Rigid Object Pose Estimation and Tracking Combining Dense and Sparse Visual Cues. pages 2347–2354. IEEE. (page 9)

[57] Přibyl, B., Zemčík, P., and Čadík, M. (2016). Camera Pose Estimation from Lines using Pl$\backslash$" ucker Coordinates. *arXiv preprint arXiv:1608.02824*. (page 13)

[58] Raab, F. H., Blood, E. B., Steiner, T. O., and Jones, H. R. (1979). Magnetic position and orientation tracking system. *IEEE Transactions on Aerospace and Electronic systems*, (5):709–718. (page 8)

[59] Ramalingam, S. and Taguchi, Y. (2013). A theory of minimal 3d point to 3d plane registration and its generalization. *International journal of computer vision*, 102(1-3):73–90. (page 3)

[60] Reitmayr, G. and Drummond, T. (2006). Going out: robust model-based tracking for outdoor augmented reality. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 109–118. IEEE Computer Society. (page 10)

[61] Reitmayr, G. and Drummond, T. W. (2007). Initialisation for visual tracking in urban environments. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–9. IEEE Computer Society. (page 8)

[62] Rosten, E. and Drummond, T. (2006a). Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer. (page 9)

[63] Rosten, E. and Drummond, T. (2006b). Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer. (page 22)

[64] Rosten, E., Reitmayr, G., and Drummond, T. (2010). Improved ransac performance using simple, iterative minimal-set solvers. *arXiv preprint arXiv:1007.1432*. (page 14)

[65] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011a). ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE. (page 9)

[66] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011b). Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE. (page 22)

[67] Scharstein, D. and Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE. (page 9)

[68] Schulz, A., Jung, F., Hartte, S., Trick, D., Wojek, C., Schindler, K., Ackermann, J., and Goesele, M. (2011). Cuda surf-a real-time implementation for surf. (page 22)

[69] Schweighofer, G. and Pinz, A. (2006). Robust pose estimation from a planar target. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2024–2030. (page 12)

[70] Strasdat, H., Lovegrove, S., et al. (2011). Sophus: C++ implementation of lie groups using eigen. (page 54)

[71] Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pages 6–329. ACM. (page 14)

[72] Taylor, C. J. and Kriegman, D. J. (1994). Minimization on the lie group so (3) and related manifolds. (page 20)

[73] Torr, P. H. and Zisserman, A. (2000). Mlesac: A new robust estimator with application to estimating image geometry. *Computer vision and image understanding*, 78(1):138–156. (page 14)

[74] Vacchetti, L., Lepetit, V., and Fua, P. (2003). Fusing online and offline information for stable 3d tracking in real-time. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–241. IEEE. (page 10)

[75] Wagner, D. (2009). `https://www.youtube.com/watch?v=eZ2-Z7nI6SM`. [Online; accessed 25-February-2019]. (page 2)

[76] Wagner, D., Langlotz, T., and Schmalstieg, D. (2008a). Robust and unobtrusive marker tracking on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 121–124. IEEE Computer Society. (page 21)

[77] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2008b). Pose tracking from natural features on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 125–134. IEEE Computer Society. (page 1, 21)

[78] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2010). Real-time detection and tracking for augmented reality on mobile phones. *IEEE transactions on visualization and computer graphics*, 16(3):355–368. (page 22)

[79] Wagner, D. and Schmalstieg, D. (2007). *Artoolkitplus for pose tracking on mobile devices*. na. (page 9)

[80] Wagner, D., Schmalstieg, D., and Bischof, H. (2009). Multiple target detection and tracking with guaranteed framerates on mobile phones. In *Mixed and augmented reality, 2009. ISMAR 2009. 8th IEEE international symposium on*, pages 57–64. IEEE. (page 4)

[81] White, S., Feiner, S., and Kopylec, J. (2006). Virtual vouchers: Prototyping a mobile augmented reality user interface for botanical species identification. In *3D User Interfaces, 2006. 3DUI 2006. IEEE Symposium on*, pages 119–126. IEEE. (page 22)

[82] Wohlhart, P. (2008). *Tracking on-line learned natural features for mobile augmented reality*. na. (page 12)

[83] Wuest, H., Vial, F., and Stricker, D. (2005). Adaptive line tracking with multiple hypotheses for augmented reality. In *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 62–69. IEEE Computer Society. (page 10)

[84] Yang, X., Guo, J., Xue, T., and Cheng, K.-T. T. (2018). Robust and real-time pose tracking for augmented reality on mobile devices. *Multimedia Tools and Applications*, 77(6):6607–6628. (page 8)

[85] Zhang, L., Xu, C., Lee, K.-M., and Koch, R. (2012). Robust and efficient pose estimation from line correspondences. In *Asian Conference on Computer Vision*, pages 217–230. Springer. (page 13)

[86] Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10. (page 9)

[87] Zheng, Y., Kuang, Y., Sugimoto, S., Astrom, K., and Okutomi, M. (2013). Revisiting the pnp problem: A fast, general and optimal solution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2344–2351. (page 13)