DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Efficient and Robust Circle Grids for Fiducial Detection

Michael Loipführer

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Efficient and Robust Circle Grids for Fiducial Detection

Kreisgitter zu effizienten und robusten Referenzmarkenerkennung

Author:	Michael Loipführer
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	Nikolaus Demmel
Submission Date:	16.05.2022

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 13.05.2022 Michael Loipführer

Abstract

Many augmented reality and robotic applications rely on fiducial markers for reliable and accurate estimation of the relative 3D pose between camera and an object. This thesis introduces a new fiducial system design based on 3×3 grids of individual circles. Our method is specifically designed to work well with fisheye cameras, which produce significant amounts of image distortion. By combining individual ellipse detections with geometric heuristics our method is able to reliably detect markers even at the edges of distorted camera field-of-views. Using all circle locations for pose estimation leads to more robust results compared to previous state of the art methods. We also describe a novel way of composing individual fiducial tags into larger grids. This enables the use of larger markers without adapting the detection pipeline, resulting in additional occlusion resistance and error detection. Our marker coding system facilitates grids up to sizes of 19×19 circles, which is especially useful for the calibration of large field-of-view cameras. Experiments on both simulated datasets as well as real world examples show the robustness of our marker system in comparison to existing state of the art methods.

Contents

1 Introduction

With the still ongoing rise in popularity of augmented reality applications as well as an increasing number of robots in use in industry and personal areas, there has been more and more research focused into methods which solve the problem of 3D tracking and environmental localization. In order to accurately project an artificual overlay onto the environment for immersive augmented reality, one first has to determine the 3D position of the camera which is capturing a given scene. A similar challenge is presented by many robotic applications in which a robot has to be localized reliably in predetermined environments. While there exist a multitude of possible solutions to these problems such as performing simultaneous localization and mapping (SLAM) to localize a camera in an unknown environment while simultaneously generating a map, the most common one has been the use of specifically designed fiducial markers. Fiducial markers can be seen as artificially constructed patterns designed to be easily recognizable and distinguishable. A key aspect which sets them apart from many SLAM algorithms is that the 3D position of a marker can even be recovered with only one image captured with a single camera. In SLAM one generally either needs a stereo camera setup or additional depth information to recover 3D structure from images. While fiducial markers, also called (fiducial) tags, often look similar to 2D barcodes such as QR codes, they aim towards solving very different problems. QR codes try to maximize the amount of general purpose information present in one marker to transmit payloads of multiple hundred bytes such as URLs. They also do not have to be detected correctly in challenging conditions such as unusual viewing angles or difficult lighting, as the person holding the camera is in charge of roughly aligning the camera with the code. Nor do they allow to recover the relative 3D position of a QR code relative to the capturing camera. Visual fiducial markers on the other hand are specifically designed to be recognizable in challenging environments and focus more on accurate detections of a tag than actual information content. They still encode some information to be able to detect and distinguish multiple unique markers present in the same image. The goal is to accurately recover the 3D position of a tag relative to the camera even when the marker is far away, viewed at a low angle or slightly obstructed.

1.1 Problem Statement

The design of visual fiducial tags is quite challenging, as a number of key characteristics factor into their performance. In order to be used in real world applications tags should be detected reliably even at large distances and unusual viewing angles. Ideally, the tag should still be detected correctly if parts are occluded by other objects in a scene. Determining whether a

tag is present in an image is usually done in a two step approach. First, one needs to actually find the pixels making up a marker in an image and second, the information encoded in these pixels has to be recovered. Selecting a recognizable geometric shape for the markers is a prerequisite for the first step while using an appropriate coding scheme for the tag payload provides some error tolerance to accidentally detecting random objects and image artifacts as valid fiducials. Many applications also require the detection algorithm to run in real time to process a camera feed at 30 frames per second. Finally, the method should be able to provide very stable estimates of the 3D marker position, as e.g. overlays in augmented reality applications would otherwise jitter, leading to an unpleasant user experience. Since more and more robotic and augmented reality applications make use of large field-of-view cameras to observe a larger portion of the surroundings for tracking and object detection, fiducial markers should also work well with the quite significant image distortion produced by such cameras. Many current methods, however, are not designed for high distortion camera lenses, as they often make very strict assumptions about the nature of the camera projection.

In this thesis, we introduce a new fiducial marker system based on 3×3 grids of circles that specifically provides accurate detections for high distortion cameras such as fisheye cameras. The main contributions of this work are:

- We describe a method of achieving sub-pixel detection precision of fiducial markers based on aggregating individual detections of circles in input images into circle grids using local geometric heuristics which are not affected by image distortion.
- We design our markers to be detected reliably in all regions of highly distorted images such as produced by large field-of-view cameras with e.g. fisheye lenses, without the need for costly preprocessing steps such as undistorting the input image. This is a use case currently not covered by many existing state of the art methods and is especially useful for many robotic applications using these kinds of cameras.
- We describe an easy to implement coding system based on Lyndon words and optional error-detection and correction schemes to provide rotationally unique tag identifiers.
- We introduce a novel way of composing individual fiducials into larger marker grids to increase detection robustness and error-correction capabilities.
- We provide experimental results on a set of synthetic and real world benchmarks comparing our method to state of the art fiducial systems.

1.2 Outline

We will start by introducing and reviewing related fiducial systems in the next chapter. Afterwards, Chapter 3 covers the theoretical foundations used in later chapters such as the mathematical representations of ellipses, different camera models and the methods for 3D pose estimation used in our implementation. Subsequently we will give a high level overview of our marker system design and explain the underlying design decisions in Chapter 4

before diving into the actual marker detection pipeline in Chapter 5. In Chapter 6, we will introduce our novel way of composing individual fiducial markers into larger grids. Chapter 7 then gives an overview of the experimental results of our marker system on both synthetic datasets as well as real world examples. Finally, Chapter 8 will close of this thesis by shortly summarizing our contributions and presenting avenues for future work.

2 Related Work

This chapter aims to give a short overview over the many different designs of existing fiducial marker systems. While the individual markers often differ quite strongly, their basic design can mostly be attributed to one of three categories. Markers with a square outer border, ones based around circular features and ones that use a specific topological, geometric structure.

2.1 Square Fiducial Markers

Most practically used fiducial marker systems are based around square features. In general, these markers are made up of a black, square border containing some black and white features to encode a tag identifier. One of the older while still quite frequently used implementations is ARToolkit [1]. While ARToolkit uses simple symbols like latin letters to encode the tag identifier most newer marker designs that build upon it have adopted a QRCode-like black and white binary pixel grid to encode their payload. Some prime examples are the ARTag [2] and ARToolkitPlus [3] fiducial systems. Other widely used fiducial system are AprilTag [4], [5] and ArUco [6] which improved upon ARTag and ARToolkitPlus in terms of identifier encoding and detection accuracy. Generally all of the previously mentioned marker systems follow the same basic detection pipeline. First, all approaches try to find the square border that encompasses every marker. In some fiducial systems like AprilTag this is done by fitting line segments to the image gradient, then building quads from these line segments. In others like ArUco the image is first binarized using adaptive thresholding. Then a contour following algorithm is employed to find all convex four-corner polygons. After generating these marker candidates, a homography is computed to remove the perspective distortion of the tag. Recovering the marker payload is done by overlaying a grid on top of the undistorted binary image. ARToolkit correlates the sampled grid points against the known tag symbols (e.g. latin characters). In most other fiducial systems the sampled grid points directly represent the decoded tag identifier. At this point most square fiducial markers use specific coding systems to generate tag identifiers resulting in different error detection and correction schemes. In addition to differentiating between multiple tags in a given scene these marker identifiers are also used to reliably recover the orientation of a detected fiducial. AprilTag e.g. uses lexicodes to generate rotationally unique code words. A slightly more recent addition to the family of square markers was introduced by Benligiray et al. in 2019 as STag [7]. Their basic detection process is similar to that of AprilTag or ArUco as they first detect the quad representing the outer marker border. In a second step before payload decoding they exploit the more accurate localization of circles by fitting an ellipse to the inner circle of the tag to refine the corner positions as well as the homography later used

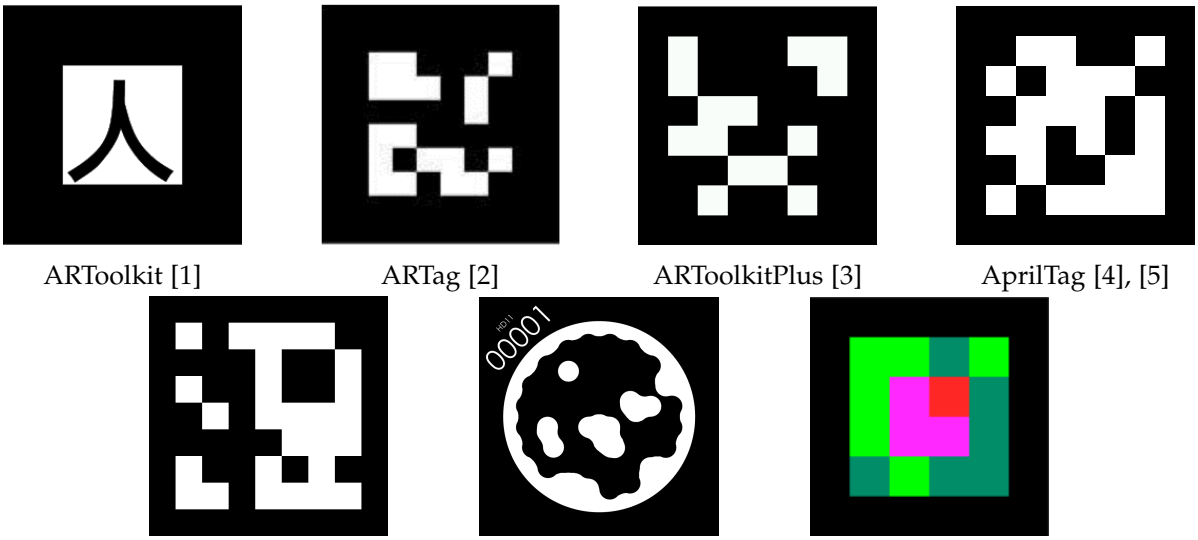


Figure 2.1: Existing square fiducial marker systems.

for payload detection. While the basic principle behind STag's payload encoding follows the idea of using a binary pixel grid, they propose to encode bits using circles instead of squares and merge them using morphological operations which they claim is more robust to slight localization errors.

A mentionable exception to the described detection pipeline is presented by ChromaTag [8]. While the method still uses a square border around the tag its payload is encoded using distinctive color patterns. Instead of initially trying to detect the square border itself the color patterns are used to discard false positives in the early detection stages thereby achieving faster detections. Similar to the other square fiducial markers ChromaTag also uses a homography to remove the perspective distortion before decoding the marker payload. Examples for all of the marker systems discussed in this section can be seen in Figure 2.1.

The main disadvantage of fiducial systems which rely on a square border for marker detection is that they do not work well in heavily distorted images as produced by fisheye cameras. Straight lines in a marker, which make up the square border, can be distorted and bent significantly if markers are captured at the edges of a fisheye camera's field-of-view. This renders the basic detection scheme of many square markers unusable as they generally rely on detecting a four sided quad for initial marker detection, often based on detecting straight lines as a first step. The solution of first undistorting the whole input image is also not quite ideal as it requires additional processing steps and leads to some loss in image detail during undistortion.

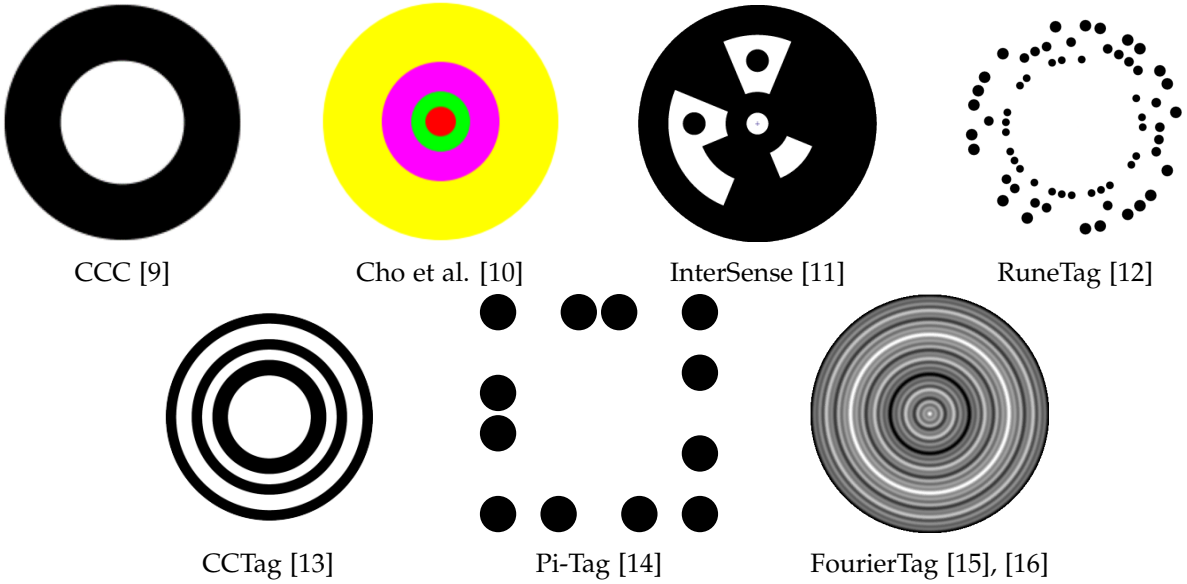


Figure 2.2: Existing circular fiducial marker systems.

2.2 Circular Fiducial Markers

Very early work towards using circular features for fiducial markers was done by Gatrell et al. [9] who introduced the concept of using concentric contrasting circles (CCC). This was later extended by Cho et al. [10] by using multiple colored concentric rings to achieve a larger detection range. One of the downsides of these early methods is the limited amount of payload (or lack thereof) that can be encoded in a marker. A later work by Naimark et al. [11] extended the original concept by reserving two inner rings subdivided into 45° radial sectors to encode up to 15 bits of payload. When introducing CCTag, Calvet et al. [13] extended the original work of Gatrell et al. by using multiple concentric black circles and employing theoretical gradient properties of concentric circles to get a robust detection pipeline. By altering the width of its concentric rings CCTag is able to encode different tag identifiers. A similar looking, but fundamentally different marker system was proposed by Satter et al. [15] and Xu et al. [16], named FourierTag which encodes payload information as a circular grayscale frequency image. Another fiducial system similar to that of Naimark et al. was introduced with RuneTag [12]. By placing circles in concentric rings RuneTag achieves high occlusion resilience while being able to encode a large number of unique marker identifiers.

A method which is quite similar to the fiducial system proposed in this work is Pi-Tag [14]. Here circles are laid out in a square with the distance ratios between circular features encoding the marker identifier. It is important to note that while not using homographic projections to remove perspective projection, most fiducial systems discussed in this section still assume a regular projective geometry to decode their marker payload. The exception here being the very basic systems by Gatrell et al. and Cho at al., CCTag as well as Fourier Tag, as these only use properties of concentric rings to encode marker payloads.

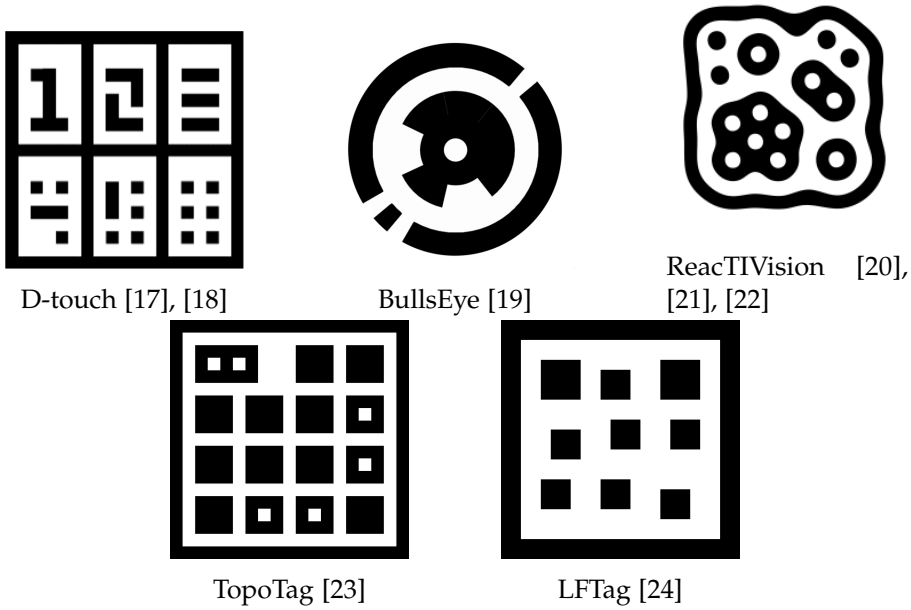


Figure 2.3: Existing topological fiducial marker systems.

A crucial downside of many of the marker systems describe above is that they cannot be used for camera or marker pose estimation as they provide too few point correspondences. The exception being Pi-Tag and RuneTag as they each use multiple circles to build one marker. Both of these two systems, however, also do not work well for distorted fisheye images without an initial undistortion step, as they incorporate the assumption of an ideal projective geometry into their marker detection scheme.

Figure 2.2 shows an example for every marker designs discussed in this section.

2.3 Topological Fiducial Markers

The third major category of fiducial marker systems is based around the idea of exploiting the topological information present in certain marker geometries for easier detection and localization. A very early such design is D-touch [17], [18]. To detect its fiducials D-touch proposes to build adjacency graphs of connected components in an input image. By comparing the adjacency information in the captured image with the known topology of markers the method is then able to recover the position of fiducials. Due to its strict topological requirements D-touch is only able to provide one distinct marker design. This problem was later solved by ReacTIVision [20], [21], [22] which introduced a left heavy depth sequency of the adjacency graph to encode a larger number of unique tag identifiers. Another method called BullsEye [19] uses a black ring border with three white notches encoding its orientation around a segmented payload ring which encodes the tag identifier (similar to InterSense). All three methods, however, do not provide a way to perform 3D pose estimation

using the detected fiducials as the markers contain too few feature points. Two later methods called TopoTag [23] and LFTag [24] solve this problem by using more feature points while still being able to encode a large number of unique tag identifiers. In the case of TopoTag the topological structure itself, similar to ReacTIVision, is used to encode the marker identifier while LFTag uses relative position differences in its feature points to distinguish different markers. Similar to the previous sections, an example for each fiducial design can be found in Figure 2.3.

A major difference between the class of circular or square markers designs and markers based around topological information is that the latter is generally more robust to larger distortions in the input images as they do not assume a standard projective geometry during their detection process. This means, markers like TopoTag or LFTag are in principle more robust in applications using large field of view cameras such as fisheye lenses. There are some notable exceptions such as CCTag but this general rule holds for most approaches introduced in Section 2.1 and Section 2.2.

3 Preliminaries

This chapter covers the mathematical basics and preliminaries needed to understand the fiducial marker design and detection process presented in this work. At first, the mathematical foundations behind ellipse representations will be introduced, followed by a deeper look at different camera models. Finally, the two approaches this work uses for 3D pose estimation, linear methods and nonlinear optimization will be explained in more detail. To keep a consistent notation throughout the thesis, we will use bold faced lowercase symbols for vectors (e.g. $\boldsymbol{x} \in \mathbb{R}^3$) and bold faced uppercase symbols for matrices (e.g. $\boldsymbol{R} \in \mathbb{R}^{3\times3}$).

3.1 Ellipse Representations

Since one of the major steps in our fiducial detection pipeline consists of the detection and estimation of ellipses in input images it will be helpful to take a closer look at the different mathematical representations as they will be used in later chapters.

Ellipses are a special case of conic sections of which the most general representation, which includes all degenerate cases, can be represented by the equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0. \tag{3.1}$$

This representation also clearly shows that a conic section is a generalization of a circle, since we can obtain the equation for a unit circle $x^2 + y^2 = 1$ by substituting $A = 1, B = 0, C = 1, D = 0, E = 0, F = -1$. Rearranging Equation 3.1 results in the widely used matrix representation

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} D & E \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + F = 0, \tag{3.2}$$

which can be written more concisely by using homogenous coordinates $\boldsymbol{x} = [x, y, 1]^T$ as

$$\boldsymbol{x}^T \boldsymbol{C} \boldsymbol{x} = 0, \tag{3.3}$$

$$\boldsymbol{C} = \begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix}. \tag{3.4}$$

Here, $\boldsymbol{C}$ is often called the matrix representation of the conic where the determinant $\det(\boldsymbol{C})$ determines the type of conic. If and only if $\det(\boldsymbol{C}) > 0$ then the equation describes a true ellipse. If $\det(\boldsymbol{C}) < 0$ the equation represents a hyperbola and if $\det(\boldsymbol{C}) = 0$ a parabola. A more concise way of formulating the constraint of a conic representing an actual ellipse is to

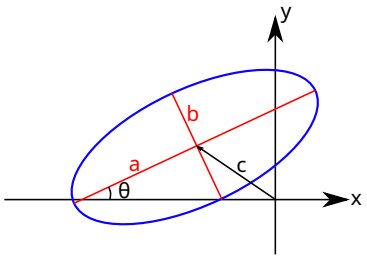


Figure 3.1: Standard representation of an ellipse with major and minor axis lengths a and b, offset to the coordinate origin $\boldsymbol{c}$ and angle θ.

use the discriminant of Equation 3.1. As stated in [25], a conic is an ellipse if this discriminant $B^2 - 4AC$ is negative. In case the conic parameters can be arbitrarily scaled this inequality constraint can be changed to the equality constraint $4AC - B^2 = 1$ by incorporating the scaling.

Another property of the conic matrix is that given its inverse $\boldsymbol{C}^{-1}$, also called the dual conic matrix, we can formulate a constraint for a 2D line $ax + by + c = 0$, described in homogenous coordinates as $\boldsymbol{l} = [a, b, c]^T$, which is tangent to the conic if it satisfies

$$\boldsymbol{l}^T \boldsymbol{C}^{-1} \boldsymbol{l} = 0. \tag{3.5}$$

Similar to the conic matrix, its dual can also be represented using six parameters as

$$\boldsymbol{C}^{-1} = \begin{bmatrix} A^* & B^*/2 & D^*/2 \\ B^*/2 & C^* & E^*/2 \\ D^*/2 & E^*/2 & F^* \end{bmatrix}. \tag{3.6}$$

We can also have a look at deriving the ellipse representation from a unit circle. Any ellipse can be regarded as an affine image of the generic unit circle described by $x^2 + y^2 = 1$. This means there exists an affine matrix $\boldsymbol{A}$ transforming points on the unit circle $\boldsymbol{x} = [x, y, 1]^T$ to points on the ellipse $\hat{\boldsymbol{x}} = [\hat{x}, \hat{y}, 1]^T$ as $\hat{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{A}$ is defined as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{L} & \boldsymbol{c} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{3.7}$$

with $\boldsymbol{L}$ representing the linear part of the affine transformation and $\boldsymbol{c}$ being the center of the ellipse. If the standard representation of the ellipse with major and minor axis lengths a and b and the rotation angle θ are known the linear transformation can be written as the product of an orthonormal rotation matrix $\boldsymbol{Q}$ and a diagonal scaling matrix $\boldsymbol{S}$ as

$$\boldsymbol{L} = \boldsymbol{Q}\boldsymbol{S} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}. \tag{3.8}$$

For non-zero a and b this transformation is always invertible which means we can transform points on an ellipse back to the unit circle via $\boldsymbol{x} = \boldsymbol{A}^{-1}\hat{\boldsymbol{x}}$. A graphical representation of these

ellipse parameters can be found in Figure 3.1. To derive the conic matrix $\boldsymbol{C}$ from this affine representation we can start with the matrix representation of a unit circle

$$\boldsymbol{x}^T \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \boldsymbol{x} = \boldsymbol{x}^T \boldsymbol{D} \boldsymbol{x} = 0 \tag{3.9}$$

Using the inverse affine ellipse transformation $\boldsymbol{A}^{-1}$ we can now formulate a matrix equation for the ellipse as

$$(\boldsymbol{A}^{-1}\hat{\boldsymbol{x}})^T \boldsymbol{D} \boldsymbol{A}^{-1} \hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}^T \boldsymbol{A}^{-T} \boldsymbol{D} \boldsymbol{A}^{-1} \hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}^T \boldsymbol{C} \hat{\boldsymbol{x}}, \tag{3.10}$$

where $\boldsymbol{C} = \boldsymbol{A}^{-T} \boldsymbol{D} \boldsymbol{A}^{-1}$ denotes the matrix representation of the ellipse.

Since we will mostly use the inverse conic matrix for our ellipse estimation and detection in later chapters, it is also helpful to represent the inverse in terms of affine parameters:

$$\begin{aligned} \boldsymbol{C}^{-1} &= \boldsymbol{A} \boldsymbol{D} \boldsymbol{A}^T && (3.11) \\ &= \begin{bmatrix} \boldsymbol{L} & -\boldsymbol{c} \\ \boldsymbol{0} & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}^T & \boldsymbol{0} \\ \boldsymbol{c}^T & 1 \end{bmatrix} && (3.12) \\ &= \begin{bmatrix} \boldsymbol{L}\boldsymbol{L}^T - \boldsymbol{c}\boldsymbol{c}^T & -\boldsymbol{c} \\ -\boldsymbol{c}^T & -1 \end{bmatrix}. && (3.13) \end{aligned}$$

As the matrix representation is only defined up to a scalar, we can normalize it for a nicer representation with 1 in the bottom right corner:

$$\begin{aligned} \boldsymbol{C}^{-1} &= \begin{bmatrix} \boldsymbol{c}\boldsymbol{c}^T - \boldsymbol{L}\boldsymbol{L}^T & \boldsymbol{c} \\ \boldsymbol{c}^T & 1 \end{bmatrix} && (3.14) \\ &= \begin{bmatrix} \boldsymbol{M} & \boldsymbol{c} \\ \boldsymbol{c}^T & 1 \end{bmatrix}. && (3.15) \end{aligned}$$

This formulation also shows a useful property of the inverse conic matrix as we can directly observe the ellipse center $\boldsymbol{c}$ from the matrix. To recover the axis lengths a and b and rotation θ from a given inverse conic matrix $\boldsymbol{C}^{-1}$ one can simply perform eigendecomposition on the squared linear part as

$$\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{Q}\boldsymbol{S}^2\boldsymbol{Q}^T \tag{3.16}$$

where the squared linear part is easily computed from the inverse conic matrix as $\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{c}\boldsymbol{c}^T - \boldsymbol{M}$. The axis lengths can be computed by taking the squareroot of $\boldsymbol{S}^2$, and the angle can be computed as $\theta = \text{atan2}(Q_{21}, Q_{11})$.

3.2 Camera Models

Almost all computer vision applications that work with a combination of 2D images and 3D data such as poses or 3D points need a proper theoretical model representing the camera used to capture images. This is especially true for fiducial markers as many designs incorporate

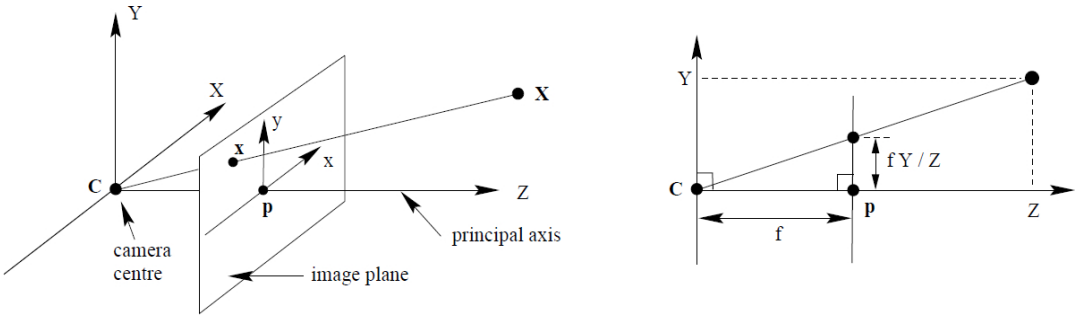


Figure 3.2: A schematic depiction of the pinhole camera model marking the optical center C, principal point p and focal length f. A point $\boldsymbol{X}$ in the three dimensional scene is projected onto the image plane as point $\boldsymbol{x}$ by casting a ray from $\boldsymbol{X}$ through the camera center $\boldsymbol{C}$. Illustration taken from [26].

specific assumptions about the intrinsics of cameras they can be used with. In order to better reason about the differences between marker designs and better understand some of the design decisions behind this work's fiducial design, this section will introduce two of the most widely used camera models, the pinhole camera and the Kannala-Brandt model.

The main purpose of a camera model is to provide a mathematical representation for the projection of a three dimensional scene onto a two dimensional canvas. This is usually formulated using two functions, the camera projection function $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ and its inverse, the reprojection function $\pi^{-1} : \mathbb{R}^2 \to \mathbb{R}^3$. These describe how a point in Euclidean coordinates in three dimensional space is mapped onto the two dimensional image plane and vice versa. Interested readers may refer to [26], [27] or the short summary in [28] which represent the basis of this section.

3.2.1 Pinhole Camera

The pinhole camera model can be seen as the simplest description of a physical camera and incidentally also directly represents the principle behind the first cameras of the 19th century. It makes the theoretical assumption that all light rays emitted from objects in a given scene are projected onto the image plane through a straight line passing through the optical center C. Unfortunately, when projecting a real scene through a pinhole onto a flat surface, i.e. when the image plane lies behind the camera center, the projection will appear upside down. This is the reason why a second equivalent, but only theoretical representation of the pinhole model is more widely used in practice in which the image plane lies in front of the camera center. An illustration of this can be seen in Figure 3.2.

This basic idea now leads to a simple mathematical representation for the pinhole model, mapping a point $\boldsymbol{x} = [x, y, z]^T \in \mathbb{R}^3$ in a scene onto the point $\boldsymbol{u} = [u, v]^T \in \mathbb{R}^2$ in the image plane using the projection function

$$\pi(\boldsymbol{x}) : [x, y, z]^T \mapsto \left[f_x \frac{x}{z} + p_x, f_y \frac{y}{z} + p_y \right]^T . \tag{3.17}$$

In this formulation the principal point $\boldsymbol{p} = [p_x, p_y]^T$ generally does not lie at the coordinate origin of the image plane, hence the additional offset. Additionally, we do allow differing focal lengths f_x and f_y for both axes of the image plane, as in addition to the actual focal length, f_x and f_y describe the scaling factor to convert world coordinates to pixels which might be different in x- and z-direction. Overall, we can nicely summarize all parameters of the pinhole model in a vector $\boldsymbol{i} = [f_x, f_y, p_x, p_y]^T$ which describes the intrinsic characteristics of a specific physical camera.

For many three dimensional applications we also need a closed-form formulation of the camera reprojection function π^{-1}, e.g. when triangulating a point which was observed from different viewpoints in multiple images. As all 3D points that lie on a straight line through the camera center are projected onto the same 2D point in the image plane, distance information is lost when applying the inverse camera projection. We therefore only obtain a vector of unit length that describes the ray the original 3D point was situated on. This can be quite easily seen in Figure 3.2 as all points on the line between $\boldsymbol{X}$ and the camera center $\boldsymbol{C}$ are projected onto the same point $\boldsymbol{x}$ in the image plane. The reprojection function π^{-1} can simply be described in a closed form as

$$\pi^{-1}(\boldsymbol{u}) : \begin{bmatrix} u \\ v \end{bmatrix} \mapsto \frac{1}{\sqrt{m_x^2 + m_y^2 + 1}} \begin{bmatrix} m_x \\ m_y \\ 1 \end{bmatrix}, \tag{3.18}$$

$$m_x = \frac{u - p_x}{f_x}, \tag{3.19}$$

$$m_y = \frac{v - p_y}{f_y}. \tag{3.20}$$

The standalone pinhole camera model generally does not work well with general application cameras as it does not incorporate any concept of camera lens distortion. In practice it is therefore often either combined with a distortion function or only applied to already undistorted images. A more suitable approach for cameras that produce a large amount of image distortion, such as fisheye lenses, is to directly use camera models that incorporate an adequate distortion model.

3.2.2 Kannala-Brandt Camera

A more general camera model that is also able to properly capture the intrinsics of cameras with larger lens distortions was introduced by Kannala and Brandt in 2006 [29] and is mostly referred to as the Kannala-Brandt camera model. The basic idea behind the model is to use a higher order polynomial of the angle between the principal axis and the incoming light ray to describe the distortion produced by different lenses. This polynomial is then combined with a pinhole projection to produce the final camera projection function. The original paper by Kannala and Brandt introduced their model with up to 23 parameters, however, only a smaller number of parameters is used in most practical use cases. We will only use eight parameters in this work as this is the most widely used convention.

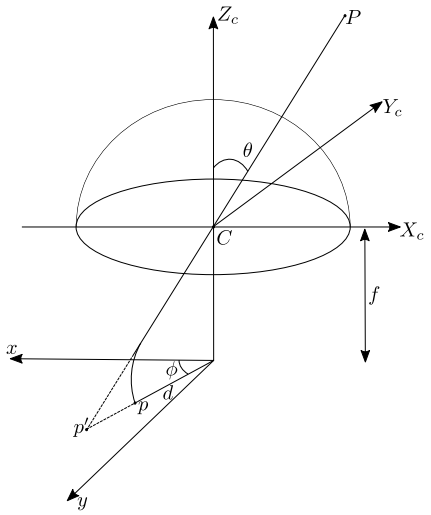


Figure 3.3: Illustration of the Kannala-Brandt camera model. The distortion model $d(\theta)$ is first applied to a point $\boldsymbol{P}$ in the scene which is then projected onto the point $\boldsymbol{p}$ in the image plane via a traditional pinhole model. Here $d(\theta)$ is described by a higher order polynomial which is only dependent on the angle θ between the original projection ray and the principal axis. Point $\boldsymbol{p}'$ shows the projection using an ideal pinhole model without any distortion. Figure taken from [29].

To explain the mathematical formulation of the Kannala-Brandt model, we first need to introduce the polynomial distortion coefficient. The simplified model introduces a slight constraint as it assumes all camera lenses to be symmetric, i.e. any distortion only depends on the angle θ between the principal camera axis and the incoming projection ray. For a more intuitive depiction see Figure 3.3. The distortion of a point $\boldsymbol{P}$ projected onto the image plane as $\boldsymbol{p}$ using an ideal pinhole model is now described by relating the distance d of the projected point to the principal point with the angle θ between the projection ray and the principal axis as

$$d(\theta) = \theta + k_1\theta^3 + k_2\theta^5 + k_3\theta^7 + k_4\theta^9. \tag{3.21}$$

This distortion coefficient is then incorporated into the ideal pinhole model, resulting in the camera projection function

$$\pi(\boldsymbol{x}) = \begin{bmatrix} u \\ v \end{bmatrix} = d(\theta) \begin{bmatrix} f_x \frac{x}{r} \\ f_y \frac{y}{r} \end{bmatrix} + \begin{bmatrix} p_x \\ p_y \end{bmatrix}, \tag{3.22}$$

$$r = \sqrt{x^2 + y^2}, \tag{3.23}$$

$$\theta = \text{atan2}(r, z), \tag{3.24}$$

enabling us to represent a Kannala-Brandt camera with an intrinsic vector

$$\boldsymbol{i} = [f_x, f_y, p_x, p_y, k_1, k_2, k_3, k_4]^T. \tag{3.25}$$

Unfortunately, as polynomials of degree nine in general do not have analytical inverses, the reprojection function does not have a closed form formulation. Given a point $\boldsymbol{u} = [u, v]^T$ in the image plane the corresponding angle θ^* has to be recovered by solving $d(\theta) = r_u$ using

$$m_x = \frac{u - p_x}{f_x}, \tag{3.26}$$

$$m_y = \frac{v - p_y}{f_y}, \tag{3.27}$$

$$r_u = \sqrt{m_x^2 + m_y^2}. \tag{3.28}$$

Afterwards, the inverse projection function can be written as

$$\pi^{-1}(\boldsymbol{u}) = \begin{bmatrix} \sin(\theta^*)\frac{m_x}{r_u} \\ \cos(\theta^*)\frac{m_y}{r_u} \\ \cos(\theta^*) \end{bmatrix}. \tag{3.29}$$

Since the Kannala-Brandt model probably is the most widely used camera model for large field-of-view lenses, we will be using it in this work, as our marker design specifically targets fisheye cameras. The main drawbacks of the model are its need to numerically solve a ninth degree polynomial and its use of expensive trigonometric operations. These performance limitations will not be too essential, however, as we will only use the reprojection function for a relatively small number of points in our input images.

3.3 Pose Estimation

One of the main applications of fiducial marker systems is to track objects in 3D space. This is usually done by attaching one or more markers to a tracked object such as a robot. The inverse use case of placing markers in a static environment and tracking the camera position is also quite common. It is important to note that there is no essential algorithmical difference between tracking the camera motion based on static markers or recovering object positions in static camera frames. To recover the 3D position of a marker one needs to compute its six degrees of freedom (6DoF) pose with respect to the camera pose from 2D and 3D point correspondences. These correspondences can be easily found by matching the detected 2D marker features in input images with the known positions of marker features in a 3D reference coordinate frame relative to the marker itself. The problem of pose estimation from 2D - 3D point correspondences, also often called the Perspective-n-Point (PnP) problem, has been tackled in numerous ways and this section aims at giving a short introduction into the general problem, its formulation and the algorithmic approaches we use in the implementation of our fiducial markers.

To keep a consistent notation throughout this thesis we will use the matrix representation of 3D positions in which a pose consisting of a rotation $\boldsymbol{R} \in SO(3)$ and a translation $\boldsymbol{t} \in \mathbb{R}^3$ is represented as a tuple $g = (\boldsymbol{R}, \boldsymbol{t})$ or when using homogenous coordinates as a full 4×4

matrix

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \in SE(3). \tag{3.30}$$

The traditional formulation of 6DoF pose estimation now takes a number of points in 3D space $\boldsymbol{p}_i \in \mathbb{R}^3$ in a known reference coordinate frame, e.g. the world coordinate frame, and their 2D projections onto a camera image plane $\boldsymbol{p}'_i \in \mathbb{R}^2$. Using this information the goal is to recover the pose of the camera $g = (\boldsymbol{R}, \boldsymbol{t})$ in the reference coordinate frame. Provided the camera is calibrated, i.e. we have all necessary intrinsics for a selected camera model, the relation between 3D points in world coordinates and 2D points in image coordinates can be expressed as

$$\alpha_i \pi^{-1}(\boldsymbol{p}'_i) = \boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t}. \tag{3.31}$$

Here α_i denotes the distance of the 3D point to the camera center as a camera reprojection does not provide the depth information of reprojected points. Instead, it only provides so-called bearing vectors describing a ray from the corresponding 3D point through the camera center. Using the relation described by Equation 3.31 the goal of pose estimation is to recover $\boldsymbol{R}$ and $\boldsymbol{t}$ from a given set of point correspondences. This formulation is often called the absolute pose problem or central PnP problem in computer vision literature.

3.3.1 Linear solutions

The most popular approach to solving the PnP problem is to use algebraically linear solutions whose computational complexity lies in $\mathcal{O}(n)$, scaling linearly with the number of point correspondences n. They are generally based on solving linear equations which lead to unique solutions irrespective of initialization. Most earlier approaches to solving the PnP problem with linear algorithms constrained themselves to the special case of the P3P problem in which a fixed number of 3 point correspondences is given. An example for such an approach is the one introduced by Kneip et al. [30]. Probably one of the most used algorithms, called EPnP, was introduced by Lepetit et al. [31] which was the first solution able to deal with an arbitrary number of points $n \geq 4$ while still maintaining a computational complexity in $\mathcal{O}(n)$. Another relatively recent approach was introduced by Kneip et al. in 2014 [32], called UPnP. The main advantages of UPnP compared to other methods is that it provides all possible solutions in ambiguous cases instead of just a single one. As the mathematical details behind the algorithms are quite involved, interested readers may refer to the original papers for more details.

3.3.2 Non-linear Optimization

Another quite common approach to the PnP problem is to simply solve it using nonlinear optimization. Unfortunately, this entails a more expensive computational approach than linear methods, such as UPnP, due to the need to use iterative solvers, and almost always requires a decent initialization to converge. Due to these limitations it is often used in combination with a linear time method where e.g. UPnP is used to get an initial estimate of the optimal camera

pose which is then refined in a nonlinear optimization. As this approach results in good pose estimation accuracy we will be using it in our implementation as described in later chapters.

To pose the original PnP correspondence problem described in Equation 3.31 in terms of a nonlinear optimization problem, we first have to derive an error residual for each point correspondence. Since the reprojection $\pi^{-1}(\boldsymbol{p}'_i)$ is only defined up to a scalar α_i and otherwise returns a unit direction vector (also called a bearing vector) we can rewrite Equation 3.31 as

$$\pi^{-1}(\boldsymbol{p}'_i) = \frac{\boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t}}{\|\boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t}\|}. \tag{3.32}$$

Now that we are only dealing with unit vectors we can employ the vector dot product to obtain a scalar residual function as

$$r_i(\boldsymbol{R}, \boldsymbol{t}) = 1 - \langle \pi^{-1}(\boldsymbol{p}'_i), \frac{\boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t}}{\|\boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t}\|} \rangle. \tag{3.33}$$

While this is not equivalent to Equation 3.32 it still results in a smooth error function that is 0 when both vectors are parallel, which in our case means they are equal as both are unit vectors, and increases when the angle between them increases. This can be easily seen as $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \|\boldsymbol{a}\|\|\boldsymbol{b}\| \cos\theta = \cos\theta$ for arbitrary vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ with $\|\boldsymbol{a}\| = \|\boldsymbol{b}\| = 1$. This means the dot product in Equation 3.32 is equal to the cosine of the angle between the reprojected ray and the unit bearing vector. Employing this error residual in e.g. a non-linear least squares formulation as

$$f(\boldsymbol{R}, \boldsymbol{t}) = \frac{1}{2} \sum_{i=1}^{n} r_i^2(\boldsymbol{R}, \boldsymbol{t}) \tag{3.34}$$

enables us to solve for the camera pose $g = (\boldsymbol{R}, \boldsymbol{t})$ using an appropriate solver such as the Levenberg-Marquardt method. Another way of formulating a non-linear optimization problem for pose estimation is to optimize the pixel reprojection error

$$\frac{1}{2} \sum_{i=1}^{n} |\pi(\boldsymbol{R}_j \boldsymbol{p}_i + \boldsymbol{t}) - \boldsymbol{p}'_i|^2. \tag{3.35}$$

While the two options are roughly interchangeable we decided on using the bearing vector approach as this is the one implemented in the OpenGV library [33], which we use in our implementation.

4 Marker Design

Before diving into the finer details of our fiducial system in Chapter 5, we aim to give a high level overview over some of the more general design decisions behind our markers. First, we introduce the kind of fiducial features that make up our markers in Section 4.1. Second, we will explain the geometric layout of our markers in Section 4.2. Finally, Section 4.3 will describe the method we use to encode different marker identifiers as well as enforce rotational uniqueness in our tags. As introduced in Chapter 2, there are basically three kinds of fiducial marker designs: Square, circular and topological markers, each with its particular advantages and drawbacks. In terms of geometric layout and fiducial features our method can be seen as a hybrid, combining aspects from both circular and topological markers.

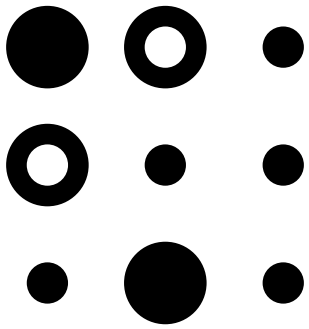
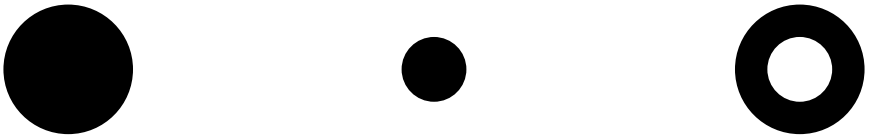


Figure 4.1: Example marker.

4.1 Fiducial Features

As shown in the exemplary marker in Figure 4.1, each one of our markers is made up of nine individual circles. This basic design is similar to Pi-Tag [14] and RuneTag [12] which also use circles as the basic fiducial features and arrange them into a square (Pi-Tag) or concentric circles (RuneTag). The main reason behind using circular features is, that when observed with an ideal pinhole camera they are always projected onto ellipses in the resulting image. This makes them quite easy and reliable to detect even under difficult lighting conditions or motion blur. For example by using the method introduced by Ouellet et al. [34]. Even though we are not using an ideal pinhole camera in practice but rather fisheye lenses with large distortions, this basic assumption still holds reasonably well as the relatively small circles aren't distorted too heavily and therefore still project onto ellipses. An additional advantage of circular features over square features is that the position of an ellipse can be detected more accurately, especially in the presence of noise, compared to detecting corners



Large circle, $b = 0$. Small circle, $b = 1$. Hollow circle, $b = 2$.

Figure 4.2: Different circular fiducial features providing a ternary digit of information per circle.

of a quad based on intersecting lines. Another aspect of using individual circles instead of a single connected topology such as a square border is, that each circle can be detected individually by only looking at a small, local image area. The individual circle detections can then be aggregated into markers by regarding their respective relative positions to each other. In contrast many square markers such as AprilTag perform their initial marker candidate detection by searching an image for lines forming a quad which represents the square border around the tag. As lines are bent significantly in very distorted images, a line based detection scheme breaks for large field-of-view cameras when a marker takes up a lot of image area and therefore is very distorted. Since we specifically tailored our method to also work well with fisheye lenses with significant distortion, this was an important consideration.

Our marker design does not alter the geometric layout to encode marker payloads as done by Pi-Tag or RuneTag but directly encodes information into our circular features. Figure 4.2 shows the three different kinds of circles used to provide a sufficient alphabet to encode up to $3^9 = 19683$ unique identifiers in a marker made up of 9 circles. As we will see later in Section 4.3, the real information content of a marker is a lot smaller, as aspects such as error correction and rotational uniqueness need to be considered as well. We also looked at using a subset of the ternary alphabet for markers, as in theory markers only made up of small and hollow circles could be detected better at larger ranges but found no significant difference in detection precision and recall during testing.

4.2 Geometric Layout

The basic geometrical structure of our fiducial markers is that of a 3×3 grid of individual circles, as seen in Figure 4.1. Instead of detecting a square border (e.g. AprilTag) or topologically structured connected components (TopoTag), our method finds markers from individually detected ellipses in given input images. The first step of our detection pipeline consists of scanning an image for all circular, or rather elliptical shapes. In a second step we then determine which set of 9 of these individual ellipses together are laid out in the basic geometry of a 3×3 grid. The main reason behind placing individual circles in a grid instead of a circular arrangement like in RuneTag is that it reduces the number of possible rotations of a tag. An important aspect of the coding system of every fiducial marker design is to enable the efficient recovery of a tag's rotation. Choosing a circular arrangement for 8 circles would lead to 8 possible rotations, meaning more bits in each code word have to

be invested to encode the marker rotation. Due to the specific geometry of a grid we can easily distinguish corner points from edge points in the grid's border and therefore reduce the number of marker rotations which have to be covered by the coding system to 4. The rationale for using a small 3×3 grid instead of a larger one is closely tied to our approach to marker detection. After finding all elliptical shapes in an input image we perform a *k*-nearest-neighbor search to find the 8 closest neighbors for each detected ellipse. By employing some geometric heuristics we are able to determine whether an ellipse, together with its neighbors fit the quite specific geometry of a 3×3 grid. The main advantage of this approach is that it operates only locally on small image areas which are not as prone to distortions, making it robust even for fisheye cameras. Larger grids would cover larger image areas and thus, as a whole, be distorted more significantly making simple geometric heuristics more difficult to use. Using a *k*-nearest-neighbor search with larger grids than 3×3 would also lead to a significantly increased computational complexity as this search is done for each individual ellipse detected in an image. The finer details of the detection process will be described later on in Chapter 5. The need for larger markers can, however, be simply covered by composing multiple individual markers into a larger grid which will be explained in more detail in Chapter 6. The ability to compose individual markers into a larger grid by overlapping multiple markers also factored into our design decisions, as it rules out the classical square or topological marker approach of including a border around the marker for easier detection.

4.3 Code Words

The final aspect of our marker design is the approach we take to encode marker identifiers. For this we use a ternary alphabet $\Sigma = \{0, 1, 2\}$ as illustrated in Figure 4.2. As already sketched in the previous section only the four corners of the marker are used to encode its rotation while the remaining five circles can be freely filled with arbitrary payload or have bits reserved for error correction. To easily generate rotationally unique code words for the four corners we use Lyndon words. These describe the set of non-empty words which are strictly smaller in lexicographic order than all of their rotations. A linear time algorithm to generate all Lyndon words of a given length was introduced by Duval in 1983 [35]. For a ternary alphabet there exist 18 Lyndon words of length 4:

$$L_4 \subset \Sigma^4 = \{0001, 0002, 0011, 0012, 0021, 0022, 0102, 0111, 0112, \tag{4.1}$$

$$0121, 0122, 0211, 0212, 0221, 0222, 1112, 1122, 1222\}. \tag{4.2}$$

This results in a theoretical number of $18 \cdot 3^5 = 4374$ unique marker identifiers we are able to encode while enforcing rotational uniqueness. In practice we decided to increase our error tolerance by additionally enforcing a minimum Hamming distance of 2 between Lyndon words to account for potential ellipse detection errors. The maximum set of Lyndon words satisfying this condition can be easily found by performing an exhaustive search over the power set of all 18 words. This leaves us with 8 words to encode our rotation:

$$L_4^* \subset \Sigma^4 = \{0001, 0022, 0102, 0111, 0212, 0221, 1112, 1222\} \tag{4.3}$$

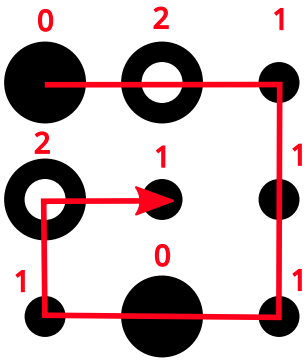


Figure 4.3: Payload encoding scheme for our markers. A code word, in this example 021110121, is encoded into a marker starting at the top left ellipse, progressing clockwise around the center with the center ellipse representing the last digit.

Before applying any error correction to the remaining five ternary digits we therefore obtain a library size of $8 \cdot 3^5 = 1944$. To describe full 9 digit code words we follow the convention highlighted in Figure 4.3. A code word, e.g. 021110121, is denoted following its encoding in a real tag. The top left ellipse in a marker represents the first digit. We then progress clockwise around the border of the 3×3 grid with the center ellipse encoding the last digit. Incorporating our rotational uniqueness constraints from Equation 4.3 we can hence formally describe the set of valid marker identifiers as

$$L \subset \Sigma^9 = \{a_1a_2a_3a_4a_5a_6a_7a_8a_9 \mid a_i \in \Sigma, (a_1a_3a_5a_7) \in L_4^*, (a_2a_4a_6a_8a_9) \in L_{ecc}\}, \tag{4.4}$$

with $L_{ecc} \subset \Sigma^5$ describing arbitrary error correction properties for the remaining payload which does not have to fulfil the rotational uniqueness constraints. To compare different rotations of a given code word, one simply needs to shift the first 8 digits leaving the last digit unchanged, as the location of the center ellipse is not affected by rotation. Rotating the code word 021110121 by 90° means we have to shift the first 8 digits by 2 positions resulting in the rotated word 111012021.

Error detection and correction

In the definition shown in Equation 4.4 our marker codes only contain a small degree of error detection which is given by using Lyndon words of length 4 with a minimum Hamming distance of 2 to encode the corner ellipses. To achieve better error detection or even some error correction, we can use the payload encoded by the remaining 5 non-corner ellipses, described by $(a_2a_4a_6a_8a_9) \in L_{ecc}$. As these do not have to be rotationally unique we can use arbitrary coding schemes to increase error detection and correction capabilities. One could, for example, use the last digit (representing the center ellipse) to encode some sort of checksum. A possibility would be to enforce a last digit a_9 such that the sum of the code word is divisible by 3. Such a simple checksum would enable the detection of a bit flip on any

of the 9 digits by reducing the library size to $8 \cdot 3^4 = 648$. Depending on the requirements for practical applications in terms of library size and robustness to erroneous detections, this is a parameter that can be easily tweaked in concrete implementations.

5 Detection Process

After introducing the required preliminaries in Chapter 3 and going over the basic ideas and considerations behind this work's fiducial marker design in Chapter 4, this chapter will introduce the actual marker detection process. A schematic representation of the detection pipeline is provided in Figure 5.1. We start by thresholding the input image to obtain a binarized version for easier processing. A set of candidate ellipse locations is generated through a connected component analysis using the binarized image. Ellipses are subsequently fitted to each candidate. We then employ some local geometric heuristics based on a k-nearest-neighbor search to find possible markers from the set of detected ellipses. Afterwards, the payload of each marker candidate is decoded and checked for validity. Finally, the pose of each detected marker relative to the camera is estimated.

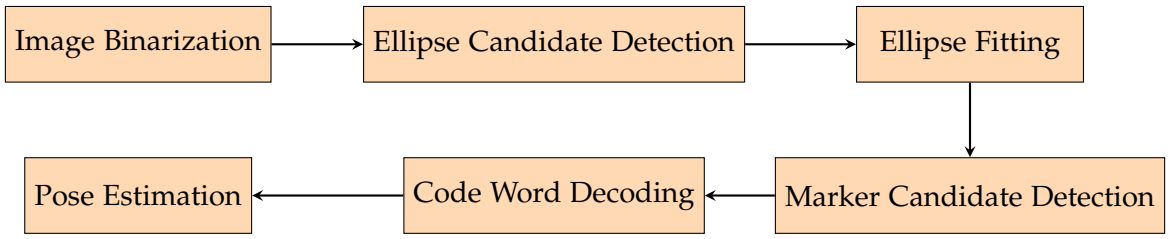


Figure 5.1: Schematic overview over the detection pipeline

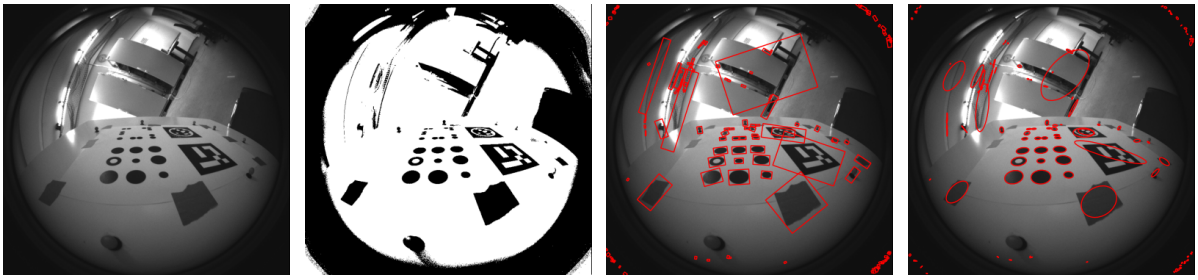
5.1 Adaptive Thresholding and Binarization

To binarize the input image we use a technique called adaptive thresholding as described in [36]. The basic idea behind adaptive thresholding is to compare the intensity of each pixel to the average intensity in a local neighborhood around it. This enables the method to deal with changing lighting conditions within an image. A key aspect of adaptive thresholding is the usage of the integral image to speed up the thresholding process, as it enables the efficient computation of sums over large image areas. The integral image $I(x, y)$ can be computed iteratively in one pass over an input image $f(x, y)$ as

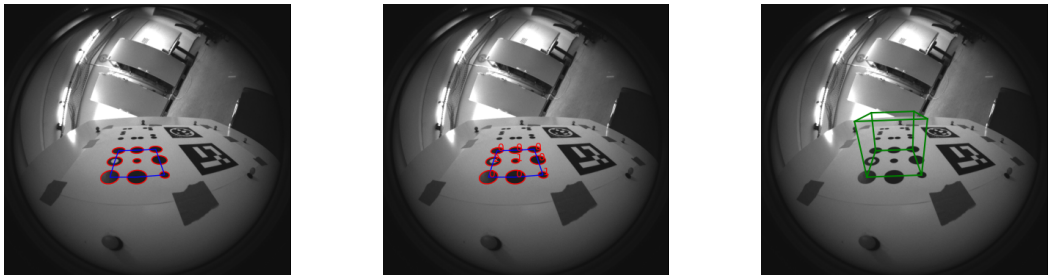
$$I(x, y) = f(x, y) + I(x - 1, y) + I(x, y - 1) - I(x - 1, y - 1), \tag{5.1}$$

enabling the efficient computation of a sum over any rectangular image area with upper left corner (x_1, y_1) and lower right corner (x_2, y_2) as

$$\sum_{x=x_1}^{x_2} \sum_{y=y_1}^{y_2} f(x, y) = I(x_2, y_2) - I(x_2, y_1 - 1) - I(x_1 - 1, y_2) + I(x_1 - 1, y_1 - 1). \tag{5.2}$$



(a) Original image (b) Thresholded image (c) Ellipse candidates (d) Detected ellipses



(e) Marker candidates (f) Valid markers (g) Pose Estimation

Figure 5.2: Detection pipeline run for an example image.

After computing the integral image, the adaptive thresholding process performs one more pass through the input image. The intensity of each pixel is compared to the average intensity in a $s \times s$ window centered around the pixel. The resulting binarized image $f'(x,y)$ can therefore be mathematically described as

$$f'(x,y) = \begin{cases} 0 & \text{if } s^2 f(x,y) \leq t \cdot \sum_{(x',y') \in S} f(x',y') \\ 1 & \text{otherwise} \end{cases} \tag{5.3}$$

where $(x',y') \in S$ describes all pixels indices within the $s \times s$ window around a given pixel at position (x,y) and $t \in (0,1)$ represents the threshold value. For simplicity this formulation does not take the borders of the image into account, actual implementations will perform border checks and adapt the average computation respectively. The result of such a thresholding step can be seen in Figure 5.2b.

5.2 Ellipse Detection

After binarizing the input image we proceed to detect all ellipses that resemble our fiducial features and could potentially be part of a tag. This is done in a two step approach. First, we generate a set of ellipse candidates, represented by oriented bounding boxes. These are then refined by fitting ellipses and performing some rudimentary filtering.

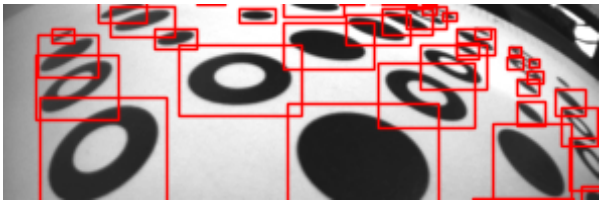


Figure 5.3: Example of overlapping ellipse candidates when using axis-aligned bounding boxes.

5.2.1 Ellipse Candidate Detection

The basic idea behind our ellipse candidate generation is to simply run a connected components analysis on the binary image and regard each connected component as a potential fiducial feature, an ellipse. While this crude approach obviously results in many false positives, i.e. many ellipse candidates which are not valid ellipses, it has a high recall, guaranteeing to include basically all actual ellipses while still restricting our overall search space. For our implementation we use the two-pass connected components algorithm introduced by Wu et al. [37]. Interested readers may refer to our implementation or the original paper for more details. After retrieving all connected components in an input image our initial approach to generating ellipse candidates was to simply use axis-aligned bounding boxes encompassing each connected component. An ellipse would then be fitted to the image area, described by the bounding box. While this worked quite well in most cases, the nature of axis-aligned bounding boxes means that they will overlap in some edge cases when ellipses are distorted heavily and captured at roughly 45° angles. See Figure 5.3 for a visual example. Due to this issue we decided on using oriented bounding boxes to describe ellipse candidates. After connected components labeling, we perform an additional pass through the labeled image to find the centroids $\boldsymbol{c}_i \in \mathbb{R}^2$ of each component. We then compute the covariance matrix of each component as

$$\Sigma_i = \frac{1}{|M_i| - 1} \sum_{\boldsymbol{x} \in M_i} (\boldsymbol{x} - \boldsymbol{c}_i)^T (\boldsymbol{x} - \boldsymbol{c}_i). \tag{5.4}$$

Here M_i denotes the set of coordinates of all pixels belonging to the component with index i. The orientation of a bounding box encompassing all component pixels can now be found by performing an eigendecomposition of the covariance matrix. The two eigenvectors $\boldsymbol{n}_{i,1}$ and $\boldsymbol{n}_{i,2}$ of Σ_i can be seen as normals to the edges of the oriented bounding box. To recover the dimensions of the bounding box, a final pass over each connected component is needed in which each initial box spanned by the two normals $\boldsymbol{n}_{i,1}$, $\boldsymbol{n}_{i,2}$ and the component centroid $\boldsymbol{c}_i$ is iteratively enlarged to enclose all component pixels. As a final step in the ellipse candidate generation, we filter out bounding boxes that are either too small or too large. In our testing we found that bounding boxes with an area smaller than a few pixels (e.g. $4 - 8$) or ones with a maximum dimension

$$\max(bbox\_width, bbox\_height) > \frac{1}{3} \min(image\_height, image\_width) \tag{5.5}$$

are unlikely to resemble valid ellipses that are part of a marker. An exemplary result of the ellipse candidate generation process can be seen in Figure 5.2c.

5.2.2 Ellipse Fitting

In order to fit actual ellipses to the image areas selected by the ellipse candidate search described in the previous section, we use the gradient based ellipse estimation algorithm introduced by Ouellet et al. [34]. Recall from Section 3.1 the dual representation of an ellipse, more generally a conic. Ouellet et al. employ the dual conic constraint which states that a line $\boldsymbol{l} \in \mathbb{R}^3$ is tangent to a conic if it satisfies $\boldsymbol{l}^T \boldsymbol{C}^{-1} \boldsymbol{l} = 0$. If a set of tangent lines $\boldsymbol{l}_i = [a_i, b_i, c_i]$ is given in homogenous coordinates, the dual conic matrix can be directly estimated using a linear least-squares approach by minimizing

$$\Phi(\boldsymbol{\theta}) = \sum_{i \in R} (\boldsymbol{l}_i^T \boldsymbol{C}^{-1}(\boldsymbol{\theta}) \boldsymbol{l}_i)^2. \tag{5.6}$$

Here, R is the set of tangent lines to be taken into account and $\boldsymbol{\theta} \in \mathbb{R}^6 = [A^*, B^*, C^*, D^*, E^*, F^*]^T$ is the six-dimensional parameter vector describing $\boldsymbol{C}^{-1}$. Incorporating the constraint $4AC - B^2 = 1$, which restricts the conic matrix to pure ellipses, into Equation 5.6 we obtain a modified matrix form

$$\left[\sum_{i \in R} \boldsymbol{k}_i \boldsymbol{k}_i^T \right] \boldsymbol{\theta}' = \sum_{i \in R} -\boldsymbol{k}_i c_i^2, \tag{5.7}$$

where $\boldsymbol{k}_i = [a_i^2, a_i b_i, b_i^2, a_i c_i, b_i c_i]^T \in \mathbb{R}^5$. The modified parameter vector $\boldsymbol{\theta}' \in \mathbb{R}^5$ contains the first five elements of $\boldsymbol{\theta}$ as the introduced ellipse constraint forces $F^* = 1$ by fixing the scaling. This formulation can be solved using a singular value decomposition (SVD). The in-depth derivation behind this formulation can be found in the original paper.

To apply this theoretical framework to an actual image, Ouellet et al. simply consider a line through each pixel $\boldsymbol{x}_i \in \mathbb{R}^2$ described by the image gradient $\nabla \boldsymbol{I}_i = [I_{ui}, I_{vi}]^T$ at that pixel. The gradient defines the normal of a line passing through the pixel as

$$\boldsymbol{l}_i = [I_{ui}, I_{vi}, -\nabla \boldsymbol{I}_i^T \boldsymbol{x}_i]^T. \tag{5.8}$$

When given a bounding box B we can now find the ellipse that bests fits the pixels contained in B by solving Equation 5.7 and incorporating Equation 5.8 for each pixel $\boldsymbol{x}_i \in B$. We obtain the inverse conic matrix $\boldsymbol{C}^{-1}$ from which we can obtain ellipse parameters such as axis lengths, angle and center, as described in Section 3.1.

5.2.3 Hollow Ellipse Fitting

Special care needs to be taken during ellipse estimation for our markers, as the original method by Ouellet et al. is not able to properly estimate black ellipses with white interiors. These, however, are an important aspect of our marker design's payload encoding. If we simply incorporate all gradient pixels in a given ellipse candidate bounding box, the resulting ellipse will be skewed if it contains a white interior circle as can be seen in Figure 5.4. This is

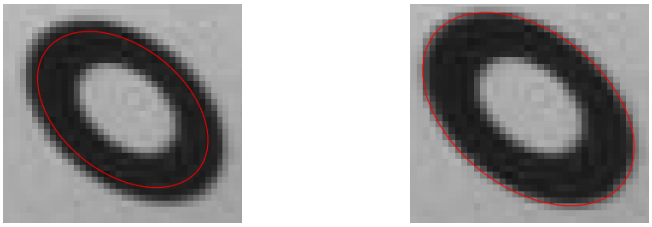


Figure 5.4: Detection of hollow ellipses without and with filtering gradient pixels.

because the gradients of the pixels belonging to the border between the actual black ellipse and the interior white ellipse are pointing in the opposite direction of the gradients of the actual ellipse border pixels. Regarding these pixels during estimation will therefore lead to a slightly smaller ellipse, as it will be skewed towards the inner white circle. To combat this problem we filter the gradient pixels which are to be included in Equation 5.7. We use the bounding box center as a rough estimate of the ellipse center and only include pixels for which the gradient direction represents a white to black transition from the pixel towards the estimated ellipse center. This works quite well in practice, as the bounding box center is guaranteed to lie within the inner white circle if the ellipses are large enough. Some exemplary results of the ellipse fitting process can be seen in Figure 5.2d. The figure also shows that due to the very crude ellipse candidate generation which regards all connected components as potential ellipses we obtain a large number of false positive ellipse detections after the ellipse fitting.

5.3 Marker Candidate Detection

After detecting and fitting all ellipses present in an input image, the next step in our detection pipeline is to generate marker candidates. A marker candidate represents some assortment of 9 ellipses which fit the geometry of our 3×3 circle grid. The basic idea behind our detection process is to regard the local neighborhood of each detected ellipse and check if eight neighboring ellipses and the ellipse itself fit the distinct geometric pattern of a 3×3 grid. The straightforward approach of only checking the eight nearest neighbors of each ellipse unfortunately does not work in practice. In some cases, for example when looking at a marker at a very low viewing angle, other image artifacts or ellipses of other nearby markers can be closer to the center ellipse than some of the other ellipses belonging to the same marker. Such an example can be seen in Figure 5.5. In order to accommodate these edge cases we therefore have to consider a larger number of closest neighboring ellipses when looking for a valid marker.

In our implementation we use the excellent nanoflann library [38] to run a k-nearest-neighbor search for every ellipse in the input image. Here $k \geq 8$ is a parameter of our detection pipeline and can be adapted depending on the use case. Once we obtain the set of k neighbors, denoted as N, for a given ellipse $i \in \{1, \ldots, n\}$ we perform a couple of heuristic filtering steps to determine whether a subset of 8 ellipse neighbors $N_i' \subseteq N_i$ with $|N_i'| = 8$

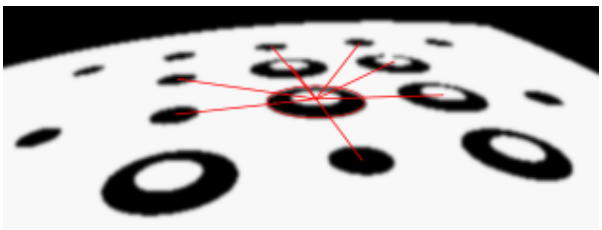


Figure 5.5: A marker captured at a low viewing angle surrounded by additional ellipses. The eight nearest neighbors of the markers central ellipse are highlighted and do not correspond to the eight ellipses making up the border of the 3×3 circle grid.

fit the geometry of a 3×3 grid together with ellipse i as the center. In the following $\boldsymbol{c}_i \in \mathbb{R}^2$ denotes the center of ellipse i. First of all we compute the distance of each neighbor to the center ellipse as

$$d_j = \|\boldsymbol{c}_j - \boldsymbol{c}_i\|, j \in N_i. \tag{5.9}$$

We also search for the neighbor $k \in N_i$ closest to the center ellipse. We compute the angle between each neighbor and the closest neighbor k relative to the center ellipse:

$$\theta_j = \text{atan2}(c_{k_y} c_{i_x} - c_{k_x} c_{i_y}, c_{k_x} c_{i_x} + c_{k_y} c_{i_y}). \tag{5.10}$$

We use the closest ellipse as a reference point as it is guaranteed to be part of the marker as long as no additional artifacts are present inside the 3×3 grid. The list of neighbors is then sorted by the relative angle θ for easier processing. The sorted list of neighbors combining the relative angle and distance then provides the basis for a number of heuristic filtering steps which can be summarized as

1. Discard outlier ellipses with too small angular differences.
2. Find an ellipse lying opposite of the closest neighbor w.r.t the center ellipse.
3. Find the closest ellipse in each marker half as divided by the line through the closest ellipse and the center.
4. Perform a brute force search to find the best way of placing the remaining ellipses into a 3×3 grid.
5. Perform a final sanity check such that the resulting marker candidate fits a 3×3 grid.

We will now explain this process step by step and also provide the rationale behind each heuristic.

Discarding outliers

If we revisit the example shown in Figure 5.5 and assume $k = 15$ we can see that all ellipses in the image would be considered as neighbors of the markers center ellipse. Performing

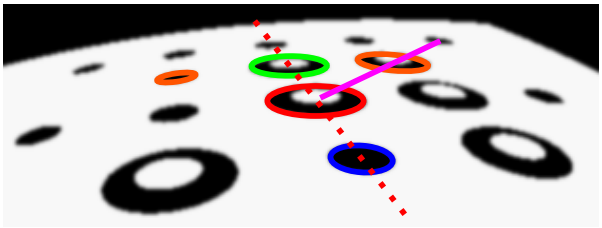


Figure 5.6: Illustration of the first couple of heuristic filtering steps. 1) The closest neighbor (green) to the center of a potential marker candidate (red) is considered as part of the marker candidate. 2) Relative angles of each neighbor are computed w.r.t. the closest neighbor as well as the distance of each ellipse to the center. 3) Outliers with small angular differences are discarded. The farther ellipse of the two intersected by the purple line cannot be part of the candidate marker. 4) The ellipse opposing the nearest neighbor (blue) is also considered to be part of the marker. 5) The closest neighbors (orange) in each marker half as described by the dotted red line through the closest neighbor and its opposite are also considered as part of the candidate marker.

a brute force search to find a subset of 9 ellipses which best fits a 3×3 grid right away would be very expensive as this is done for each ellipse in the input image. To reduce the number of possibilities we perform an initial filtering step to discard ellipses which obviously cannot be part of a marker since another ellipse lies closer to the center at roughly the same direction w.r.t. the center ellipse. This case is illustrated in Figure 5.6 by the purple line. We can describe this process formally by using the sorted list of neighbors indices N_s as

$$N'_s = \{j \in N_s \mid \nexists j' \in N_s, j' \neq j : \text{diff}(\theta_j, \theta_{j'}) \leq \theta_{thresh} \wedge d_{j'} < d_j\}, \tag{5.11}$$

where θ_{thresh} describes the minimum angular difference between two ellipses in a valid marker and $\text{diff}(\theta_j, \theta_{j'})$ denotes the difference between the two angles taking potential wrap-around at 360° into account. In practice, we found that a threshold of $\theta_{thresh} = 8°$ works quite well. Put plainly this means we discard all ellipses where the angular difference to another neighboring ellipse is smaller than a given threshold and the other ellipse is closer to the center. This step is quite important in cases where the recorded marker is surrounded closely by other regular geometric patterns such as other markers and is a prerequisite for marker composability introduced in Chapter 6.

Finding the opposite ellipse to the closest neighbor

A property of the the geometry of a 3×3 grid we found empirically, is that the relative angular difference of opposing ellipses in the grid will always be roughly 180°, regardless of viewing angles and camera distortions. Graphically one can illustrate this by drawing a line from any marker ellipse situated at the border of the grid through the center ellipse (see the dotted red line in Figure 5.6). This line will always roughly intersect the marker ellipse opposite. Starting from the ellipse closest to the marker center we use this property to look

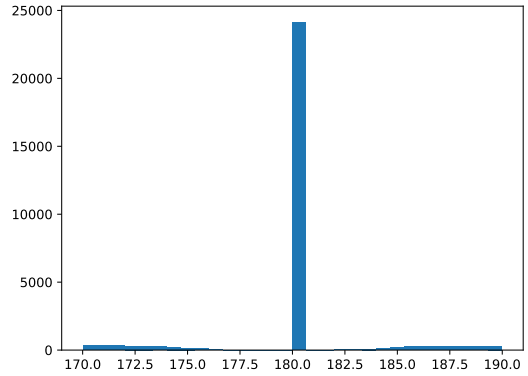


Figure 5.7: Histogram of the angular difference between the marker ellipse closest to the marker center and its opposing ellipse in the 3 x 3 circle grid in a marker observed from 28900 different simulated viewpoints with viewing angles of down to 5°. The full range of simulated angular differences is shown, no angles smaller than 170° or larger than 190° were measured.

for its opposing neighbor which, if there is one, we then can also be sure belongs to the same marker. We validated this property by simulating a marker from multiple viewpoints and recording the angles between the closest neighbor and its opposite. The results are illustrated in Figure 5.7, which shows the angular difference of the closest neighbor to its opposite ellipse, highlighting the consistent angular difference of 180° for a multitude of different viewing angles. We therefore consider the ellipse for which

$$180^\circ - \theta_{tolerance} \leq \theta_j \leq 180^\circ + \theta_{tolerance} \tag{5.12}$$

holds also to be part of the candidate marker we are trying to find. In case this constraint holds for multiple ellipses the one closest to the marker center is chosen. In practice we found a $\theta_{tolerance} = 15^\circ$ to work quite well, which is supported by our findings shown in Figure 5.7.

Find the closest neighbor in each marker half

Going back to the example shown in Figure 5.6, the candidate marker now consists of the center ellipse shown in red, the closest neighbor shown in green and its opposing ellipse highlighted in blue. As a last step before resorting to a brute force search to find the set of ellipses best fitting a 3×3 grid, we employ one more heuristic to restrict our search space. When dividing the candidate marker into two halves using a line through the closest neighbor, the center and its opposite, shown as a dotted red line in Figure 5.6, the closest neighbor in each of those halves, shown in orange, will also be part of the actual marker in almost all cases. This means that out of k initial nearest neighbors, 4 in addition to the center ellipse are certainly part of the candidate marker. Excluding ellipses discarded during outlier detection, at most $k - 4$ ellipses still need to be considered for the final marker candidate. In the case of

$k = 8$ this leads to a trivial solution as there exists only one possibility to arrange 9 points in a 3×3 grid which can easily be found using the relative angles computed at the beginning. If $k > 8$ we need to perform an exhaustive search over all remaining possibilities.

Exhaustive search for all possibilities of placing remaining ellipses in a 3 x 3 grid

Figure 5.8 shows the state of our example after the previous heuristic step. There are a number of possibilities to arrange the remaining ellipses into a 3×3 grid. As we want to easily be able to distinguish between corner and edge ellipses in the grid for later code word decoding, we will follow the same convention to order ellipses in a tag as introduced in Section 4.3 to encode the actual payload in a marker. Ellipses in a tag are ordered with a corner point first, then progressing clockwise around the center with the center ellipse coming last. A possible ordering for a candidate marker in the example shown in Figure 5.8 would therefore be the sequence $(3, 4, 5, 6, 7, 8, 9, 1)$, with the center point being omitted as its position in the ordering is always constant. Fortunately, we do not have to regard all subsequences of length 8 of the set of ordered neighbors N_s as we can make use of some constraints. First of all, we already know the closest neighbor and its opposite to be part of the marker candidate, essentially cutting the space of possibilities in half as we can regard each marker half separately. Additionally we can restrict the set of possibilities for each marker half to those that include the respective closest neighbor in each half as discussed above. To formalize this, let $S = (1, \ldots, n), n \geq 8$ be the sequence of neighboring ellipses ordered by relative angle to the closest neighbor $c = 1 \in S$. Let the ellipse opposite the closest neighbor be denoted as $o \in S$ and the closest ellipse in each marker half as $h_1, h_2 \in S$. An ordering $S' \subseteq S$ of ellipses that could represent a valid marker candidate would have to fulfil

$$\begin{aligned} S' = &(1, x_1, x_2, x_3, o, x_4, x_5, x_6) \\ &h_1 \in (x_1, x_2, x_3) \\ &h_2 \in (x_4, x_5, x_6) \\ &1 < x_1 < x_2 < x_3 < o < x_4 < x_5 < x_6. \end{aligned} \tag{5.13}$$

Since at this point we do not know if the closest neighbor is a corner or an edge point in the circle grid, we have to regard both options $S'_1 = S$ and $S'_2 = (x_1, x_2, x_3, o, x_4, x_5, x_6, 1)$. To distinguish which of these possible arrangements best fits the layout of our grid, we compute a crude, but in practice quite effective, linear interpolation error and select the arrangement with the smallest error as the marker candidate for the selected center ellipse. Given an ellipse ordering S', we define this error as

$$e_{S'} = \sum_{i \in \{1,3,5,7\}} \| \frac{1}{2}(\boldsymbol{c}_i + \boldsymbol{c}_{(i+2 \mod 8)+1}) - \boldsymbol{c}_{i+1} \|, \tag{5.14}$$

where $\boldsymbol{c}_i$ denotes the ellipse center coordinate of the i-th ellipse in S'. Intuitively, for a sequence S' we compute the ideal location of an edge ellipse as the linear interpolation of its adjacent corner ellipses, as would be the case in the undistorted grid. The sum of the deviation of the four edge ellipses from their ideal positions is then used as a rough alignment

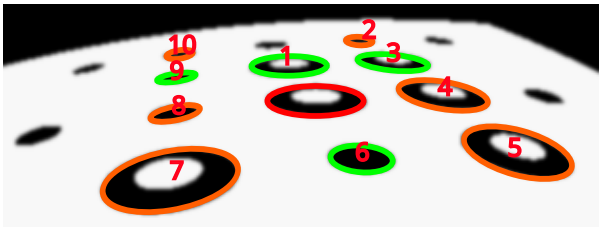


Figure 5.8: All ellipse neighbors that could be part of a marker candidate centered on the red ellipse. Ellipses which are certainly part of the marker are shown in green, while orange ones still have to be checked. The ellipses are enumerated in the order of their relative angle to the closest neighbor w.r.t. the center ellipse.

error. While this approach obviously does not accurately describe real edge positions in the presence of large distortions, the estimate is good enough in practice to reliably select the correct subset of ellipses as the best marker candidate and also reliably distinguishes edge points from corner points. After this step we therefore obtain an ordered subset $S' \subseteq N$ of the set of neighboring ellipses with the first element of the sequence guaranteed to be a corner in the 3×3 marker grid.

Final marker candidate sanity check

In order to filter out more false positive detections we perform a final sanity check on the marker candidate S' by exploiting the properties of angular difference between opposing ellipses which we discussed previously. So far, we only used the property illustrated in Figure 5.7 to constrain our search space by looking for the ellipse roughly 180° opposite to the nearest neighbor. In this final step we check whether every ellipse in our marker candidate besides the center ellipse has an opposite with a relative angular difference of approximately 180°. We hence enforce for every ellipse j that

$$\exists j' \in S' : j' \neq j \wedge 180^\circ - \theta_{tolerance} \leq |\theta_{j'} - \theta_j| \leq 180^\circ + \theta_{tolerance}, \tag{5.15}$$

and discard the marker candidates not fulfilling this constraint.

5.4 Code Word Decoding

The final step when detecting markers before being able to estimate the marker pose is to check whether the payload of a given tag actually encodes a valid identifier. Given a marker candidate which fits the geometry of our 3×3 circle we therefore must first distinguish the different kinds of ellipses present to recover the payload. As introduced in Section 4.3, we differentiate between three kinds of ellipses to encode a digit in our code words: large, small and large hollow ones. Most fiducial systems that encode information in the size or location of their fiducial features such as AprilTag compute a homography to undistort the detected marker before extracting the payload. We opted to not use this approach mostly due to

the fact that using homographies with ellipses can be quite difficult, as the homographic projection of an ellipse does not have to actually be an ellipse. This makes it harder to work with ellipse parameters such as area and axis lengths as these are difficult to extract after undistorting. We instead try to approximate the undistorted size of ellipses by relating their size in an image with the distance to neighboring ellipses. Assume we are given a marker candidate consisting of ellipses with indices $i \in (1, \ldots, 9)$, ordered such that the first ellipse is located at the corner of a grid and the last ellipse being the grids center one. We then approximate the radius of the ellipse with index i when undistorted to a circle as

$$r_i = \frac{r_e}{d_n \cdot d_g}, \tag{5.16}$$

$$r_e = \| \boldsymbol{L}_i \left[\cos(\theta_n), \sin(\theta_n)\right]^T \|, \tag{5.17}$$

with d_g denoting the true distance between circles when the marker was printed, d_n the distance to a neighboring ellipse in the detected marker candidate grid and θ_n the angle between said neighbor and the ellipse we are looking at. $\boldsymbol{L}_i$ simply describes the linear transform of ellipse i as introduced in Section 3.1. Intuitively, we are assuming that the distance between neighboring ellipses in the 3×3 grid is distorted in roughly the same way as the ellipses in that direction. We therefore approximate the undistorted size of the ellipse by relating the measured distance between neighboring ellipses to the measured size of an ellipse in the direction of that neighbor. By also dividing through the known true distance between marker ellipses we transform the approximated size into real world units of measurement. To make this process more robust we compute this approximation by using all neighbors of an ellipse instead of just one and averaging over the estimates. For a border ellipse the estimate is computed relative to its adjacent corner ellipses and the grid center. Analogously, the adjacent edge ellipses and the center are used for corner ellipses and all edge points are used to estimate the center ellipse size. To distinguish between large and small circles in a candidate marker, we compare the estimated diameter with the known diameter set when printing a marker. As a final step we also need to determine which of the large ellipses are hollow, i.e. contain an inner white circle. For each large ellipse we simply compute the average intensity of the binary image in the area covered by the ellipse. To tolerate inacuraccies during ellipse fitting we first scale the actual ellipse to a smaller size which would still encompass a potential inner, white circle. If the average intensity of this downscaled ellipse is larger than a given threshold, i.e. the inside of the ellipse is white, we determine a large ellipse to be hollow. A digit is then assigned to each ellipse in a marker candidate according to the coding scheme described in Section 4.3.

After determining all ellipse types and their respective digits we can assemble the marker payload. One goal of this process is to recover the marker's orientation from the Lyndon word encoded in its four corner points. As described in Section 4.3, a code word is encoded into a marker with the top left ellipse representing the first digit, then progressing clockwise around the grid border with the last digit in a code always encoding the center ellipse. Since the detection process described in the previous sections already distinguishes between corner and edge points in a marker candidate we can immediately construct a code word that

represents the marker by simply starting at one of the corner points and then following our encoding convention. Looking back at Figure 5.8 there are four possible code words for this marker, given by rotating the marker by multiples of 90°: 122221212, 222121122, 212112222 and 211222212. To determine which of these four (if any) describe the marker in its original rotation, we first extract the word encoding the markers corners at positions $1, 3, 5, 7$ in the code word. If this matches one of the 8 valid Lyndon words, we have recovered the markers orientation. Otherwise we need to check the remaining rotations. Such a code word rotation by 90° can simply be achieved by shifting the first 8 digits by 2. After validating the corners and recovering the orientation we can finish the decoding process by checking any used error detection or correction mechanisms as explained in Section 4.3.

5.5 Pose Estimation

Finally, after detecting a valid fiducial marker, we can proceed to recover its 3D position with respect to the camera. Since this is only possible in applications with calibrated cameras, this step is obviously omitted when markers are used for camera calibration themselves. As explained in Section 3.3, in order to recover the 3D pose of an object, a set of correspondences between 2D and 3D points are required. In our case these correspondences are the 2D ellipse centers in the detected marker image and the ideal 3D positions of our tag circles in a 3×3 grid in a metric coordinate frame. To solve the PnP problem with the correct scale right away, we incorporate the known distance between circles d into the 3D grid. We obtain the correspondences between centers $\boldsymbol{c}_i' \in \mathbb{R}^2$ and their respective ideal positions in the 3D grid $\boldsymbol{c}_i = [i_1 \cdot d, i_2 \cdot d, 0]^T \in \mathbb{R}^3$ with $i_1, i_2 \in \{0, 1, 2\}$ being the grid indices. We then apply the P3P algorithm [30] in a RANSAC scheme as well as the UPNP [32] algorithm to generate several initial pose estimates as initializations for a subsequent non-linear optimization step. We select the pose after non-linear refinement which produces the smallest pixel reprojection error as our final estimate of the marker position. As a small sanity check to discard erroneous detections we discard all markers that produce a reprojection error larger than a certain threshold e_{thresh}. The pixel reprojection error for a pose $p_j = (\boldsymbol{R}_j, \boldsymbol{t}_j)$ given a calibrated camera projection function π is computed as

$$\sum_{i=1}^{9} |\pi(\boldsymbol{R}_j \boldsymbol{c}_i + \boldsymbol{t}_j) - \boldsymbol{c}_i'|. \tag{5.18}$$

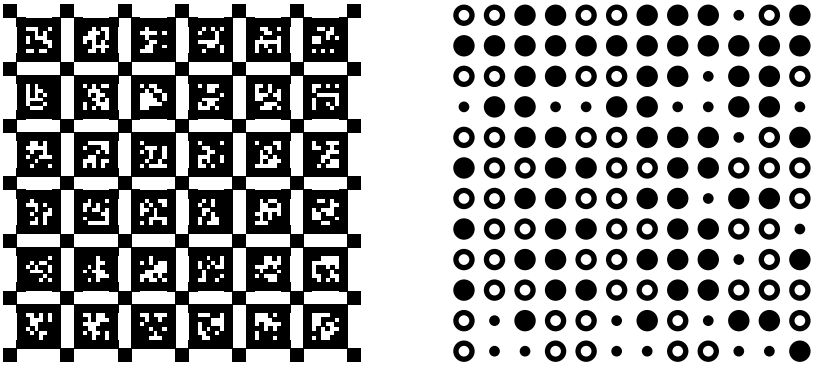
In practice we found that a threshold of $e_{thresh} = 25$ pixels works well for images of size 848×800. A notable difference of our marker design to square markers is that we are able to utilize the full nine point correspondences of our 3×3 grid for pose estimation instead of just four corners of a square border. This should in theory lead to a more accurate and stable pose estimate similar to methods such as Pi-Tag or TopoTag.

6 Composable Marker Grids

In this chapter we propose a novel way of composing fiducial markers into larger grids by exploiting the geometric structure of our tags in combination with our approach to payload encoding. Using current methods such as AprilTag or ArUco, the way of building larger patterns is to simply stack multiple markers into a grid, see Figure 6.1. The main application for such marker grids is camera calibration, as it produces more 2D to 3D point correspondences per image which can be used for camera model fitting compared to single markers. Using marker grids for camera calibration also provides a significant advantage over more classic calibration patterns such as checkerboards or circle grids, as e.g. implemented by the calibration toolbox Kalibr [39]. These patterns generally cannot be detected reliably if parts are obstructed or only a subset of the pattern is observed. Being able to still detect a partial calibration pattern is, however, especially important for the calibration of fisheye cameras in order to cover all areas of the camera field of view. This is because the large distortion at the edges of the image area may lead to parts of a calibration pattern being cut of or rendered unusable. A drawback of the straightforward AprilGrid approach is that in the case of detection errors a whole tag with multiple point correspondences, 4 in the case of AprilGrid, is missing instead of just a single point correspondence. Other approaches such as TopoTag [23] or LFTag [24] support larger markers up to sizes of 4×4, but adapt their detection algorithm for each marker size, making it harder to implement. Instead of stringing together separate markers, we generate a large circle grid in which every 3×3 subgrid represents a valid fiducial, increasing both point correspondence density as well as error tolerance due to marker overlap. Compared to AprilGrids our method is more robust in terms of error tolerance, as the wrongful or missing detection of one circle only leads to one missing point correspondence since every circle is part of more than one marker. With AprilGrids a missed tag detection leads to all four corner points of that tag not being available. Finally, as our fiducial markers are designed to work well with heavily distorted camera lenses, these larger grids can also easily be used for fisheye and large field-of-view camera calibration.

6.1 Grid Generation

More complex requirements on grid layouts unfortunately come with the caveat of a much more involved grid generation process. A brute force approach of generating possible grids and checking whether each 3×3 subgrid forms a valid marker is not feasible due to its exponential computational complexity. To generate valid grids we therefore resort to a different method of incrementally building grids with a backtracking approach. First, all



AprilGrid This work

Figure 6.1: Comparison between AprilTag grids and this work's composable marker grid, both providing 12×12 point correspondences.

possible markers in all four 90° rotations are precomputed, resulting in $7776 = 4 \cdot 1944$ tiles. At this point we use individual marker codes without any additional error detection which we will explain in more detail later on. Then the first of these is selected as a starting point of the larger grid, i.e. the top left 3×3 subarea of the larger grid is filled. Afterwards, this starting subgrid is extended with overlapping markers. In case of a dead end, meaning the desired grid size has not been reached while the current incremental grid cannot be extended, we backtrack one or multiple steps. This process is illustrated in Figure 6.2 for the generation of a small 4×4 grid. To start with, we take a single marker and attempt to find another tag which can be overlapped on the bottom two rows of the 3×3 grid. The same is done for overlaps to the right and bottom right to fill up the 4×4 grid. If no overlapping tag can be found after a step before reaching the desired grid size we backtrack until there are valid, overlapping options. While this process is able to generate all possible valid grids of a certain size it is still very computationally expensive with an exponential worst case complexity. As shown in Table 6.1, we can only feasibly use this approach for grid sizes of up to 6×6. The results also show an overwhelming number of possible 4×4, 5×5 and 6×6 grids. This is quite interesting as it would enable us to use a composed grid for normal marker applications and obtain a very large number of unique identifiers.

Grid size	Runtime	Number of grids
4×4	$2.64s$	$262,144$
5×5	$213s$	$47,807,136$
6×6	$22h50m$	$181,718,080$

Table 6.1: Number of possible marker grids for different grid sizes and their respective generation runtime with our incremental approach.

Regarding our inability to generate the full range of existing marker grids for larger sizes, we argue that for most applications only a small number of actual large marker grids will

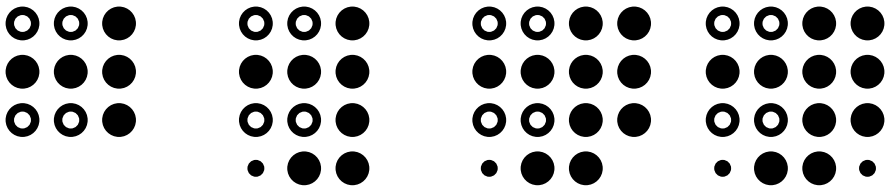


Figure 6.2: Generating a marker grid by incrementally fitting individual tags.

suffice. E.g. for camera calibration only one unique grid is required in most use cases which can easily be generated and distributed in advance. Because of this we opt for another approach to generate grids larger than 6×6. We formulate our grid constraints using the theorem solver Z3 [40] which is able to very quickly determine whether a valid circle grid exists for a given grid size and generate its structure. Unfortunately, this approach does not enable us to provide the total number of possible grids for larger sizes. Regardless, even in use cases where multiple unique grids are required, using Z3 is a valid option as we are able to quickly generate multiple unique grids by simply adding logical constraints to exclude already known grids. With Z3 we were able to determine that there exist valid grids of sizes of up to 19×19 circles. This is quite interesting as one could come to expect these marker grids to be infinitely tileable. As we are starting from a limited set of initial markers, it is conceivable and also the case here, that after a certain grid size no more overlapping tiles can be found to extend the marker grid. A deeper insight into the theoretical background of this phenomenon would be quite interesting, but unfortunately is out of scope for this work.

Recursive grids

There are, however, some other theoretical properties of our marker grids worth taking a closer look at. If we were to exclude the previously stated fact of grids only existing up to a size of 19×19, it could be possible that recursive grid patterns existed, i.e. larger grids which themselves would be infinitely tileable with overlap. While this is not the case for our library of possible 3×3 markers, there are recursive grid patterns which can be tiled infinitely when relaxing the requirements for marker, or rather grid overlap. If allowing neighboring grids to overlap by one row or column instead of two rows or columns, we can generate a number of infinitely repeatable marker grids. The smallest such repeatable pattern is of size 4×4, the largest of size 14×14. We currently do not have a direct application for these repeating grids as they are harder to detect, since each subpattern is not unique in the context of the larger, tiled grid. Nevertheless, we still regard this property as potentially useful for future applications.

Non-repeating grids

Another property that might be desirable in marker grids is that each individual 3×3 fiducial is unique within the grid. This enables easier detection as the marker identifier would suffice to determine the position in the grid upon a successful detection. Since this constraint limits

the space of possibilities quite drastically, the largest grid which fulfills the requirement is of size 11×11. Compared to the common choice for AprilGrid of using a 6×6 grid of markers for calibration purposes which provides 12×12 point correspondences this still seems sufficient for many uses cases.

6.2 Grid Detection

Even with being able to efficiently generate large composable marker grids, the most important aspect remains their accurate and robust detection. Because we require each 3×3 subgrid to constitute a valid fiducial marker we can use the same detection pipeline that we use for individual tags as introduced in Chapter 5. Some of the heuristic detection steps discussed in Section 5.3 are quite crucial for a reliable detection of individual markers which are part of larger grids. Due to the very regular geometric patterns and overlapping markers, the likelihood of erroneous detections is significantly larger when compared to a single tag application. For one, it is quite important to consider a larger number of ellipse neighbors when looking for marker candidates as not all of the 8 closest ellipses to a marker center are part of the actual marker when looking at larger grids from low viewing angles.

To stitch together individual marker detections into an overall marker grid, there are two options. The easiest is to make use of non-repeating grid patterns to immediately determine the position of a marker in a grid. This would require the layout of the used marker grid to be known in advance as would be the case in an AprilGrid-like application where only one distinct grid is used. The second, more complex option is to incrementally construct a grid detection from the overlapping individual marker detections. We did not implement this approach but a possible process could be:

1. Gather all individual 3×3 marker detections.
2. Find neighboring detections that constitute a valid overlap.
3. Aggregate neighboring detections into candidate grids.
4. Iteratively merge candidate grids with overlapping areas.

One problem with this approach which could arise for grids larger than 6×6 is that of one grid being detected as multiple smaller ones. This could be the case if e.g. the fourth column of circles in a 7×7 grid is not detected correctly, effectively splitting the grid in half. As such a case is quite unlikely, one could perform some sanity checking and discard these detections.

Error tolerance

As discussed in Section 4.3, our marker design employs a two step approach to error detection. First, we enforce the Lyndon words, which encode tag rotation in the four corners, to have a Hamming distance of 2. Second, we include an optional error detection or correction procedure, like e.g. a checksum, in the remaining 5 payload digits. As stated in Section 6.1 we

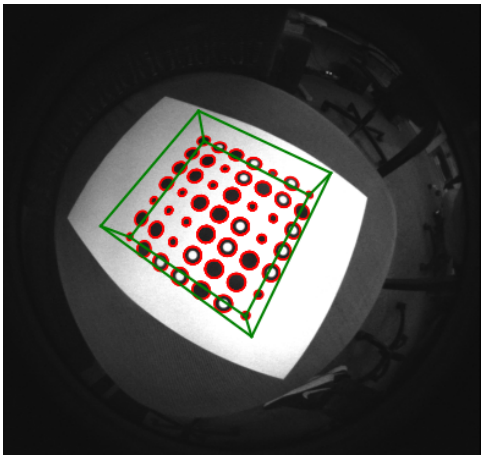


Figure 6.3: Detection of a 7×7 composed marker grid. We use the previously introduced approach of detecting a non-repeating marker grid by directly mapping each individual 3×3 grid to the correct position in the grid. Pose estimation is performed using all 49 point correspondences.

use the full number of possible code words $8 \cdot 3^5 = 1944$ to generate our composable grids, therefore opting to not use any sort of error correction for the 5 payload digits of each tag. Due to the overlapping nature of individual marker code words in a composed grid, a global error detection scheme can be implemented, as each individual circle is generally part of multiple 3×3 markers in a grid. This could be used to detect erroneous code words based on neighboring markers in a composed grid by checking detected marker overlap for validity. To not limit the number and maximum size of possible grids, we deem this level of error tolerance to be sufficient for most applications.

6.3 Applications of Marker Grids

We propose two main applications for our composable marker grids. Large fiducial grids are especially useful for camera calibration as the main criteria there is to generate as many 2D to 3D point correspondences as possible to be used when fitting a camera model. A higher density of points therefore either means more correspondences per frame or more robust detection when using fewer points and therefore larger features. Additionally, it is essential that the markers can also be detected from different viewing angles and in the presence of large camera distortions such that they can be used with a multitude of different camera models. As shown in the comparison with AprilGrid at the start of this chapter in Figure 6.1, our method produces slightly larger fiducial features at the same point density while using circular features instead of line intersections which should lead to more accurate estimates and more robust detections.

The second application we have in mind for our grids is to enable markers of varying sizes while using the same detection process. Methods such as TopoTag or LFTag provide

the option of using 3×3 or 4×4 markers each, but in turn have to adapt their detection algorithm for each marker size. With our composable markers one could choose to use 4×4, 5×5 or even larger grids as individual markers which would provide more point correspondences for e.g. pose estimation. As shown in Table 6.1, this would also result in an enormous number of unique grid identifiers. An additional benefit of using composable grids as individual fiducials is the increased error tolerance to missed circle detections. In the case of e.g. specular reflections on a marker it is possible that the image binarization discards some circles which would lead to a missed detection in the case of a single 3×3 marker. Composed marker grids represent a larger tolerance to occlusion as each individual circle, apart from the grid corners, is part of at least two markers which means that a grid can still be detected if one or more circles are missing.

7 Evaluation

This chapter describes the experimental results of the fiducial marker design introduced in the previous chapters. Our evaluation will be done on two main datasets. One generated artificially in a 3D simulation, the other recorded using an Intel RealSense Tracking Camera T265 [41] with a motion capture (MoCap) system providing ground truth for pose estimation. We will highlight our marker's performance with regards to detection accuracy, pose estimation accuracy and detection speed and finally, also showcase some failure cases.

To slightly improve the detection accuracy and error tolerance of our method in this evaluation, we implement the checksum approach to error correction described in Section 4.3 where the center ellipse of a marker is chosen such that the sum of the code word digits is divisible by 3.

All of the presented evaluations are run single-threaded on a desktop PC with an Intel Core i7-6700 CPU with 3.40GHz and 16 GB of RAM.

7.1 Datasets

We use both a simulated dataset as well as real images for several reasons. When evaluating a fiducial marker in terms of theoretical detection accuracy, parameters such as viewing angle and viewing distance are more easily controllable in a simulated environment whereas real images include use cases with difficult, realistic lighting conditions and image noises.

7.1.1 Synthetic

For our synthetic dataset we implement a 3D simulation of our marker using OpenGL in which one marker of size $10cm \times 10cm$ is captured by a virtual camera at a specific viewing angle and distance. Each evaluation is done with different camera models. We use a pinhole model with a focal length of $280mm$ and a Kannala Brandt model with the same parameters as the RealSense T265 camera used to capture our real world dataset. All simulated images have a resolution of 848×800 pixels, same as the RealSense T265. To also test our k-nearest neighbor detection scheme we draw additional ellipses around the marker to mimic the situation of additional circular objects being present in a scene close to the marker as well as the detection within a marker grid. Some examples are shown in Figure 7.1.

Evaluation Methodology

We will primarily use the simulated data to showcase our marker designs performance with respect to viewing distance and viewing angle. Multiple synthetic images are generated

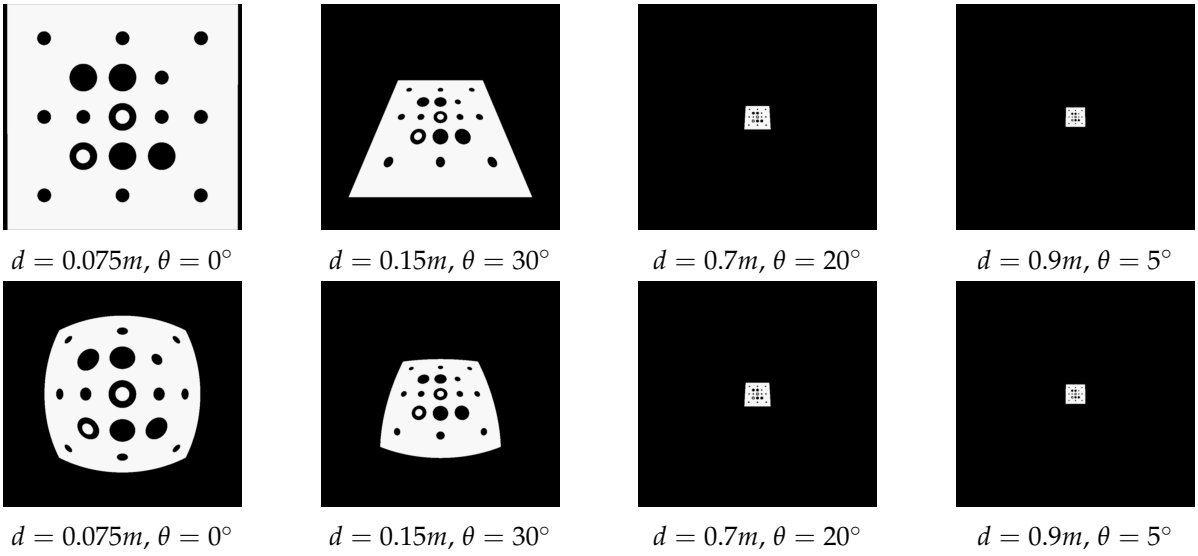


Figure 7.1: Examples of simulated markers with a size of $10cm \times 10cm$ using a pinhole camera model (top row) and Kannala Brandt camera model (bottom row) at respective distance d to the marker and viewing angle θ.

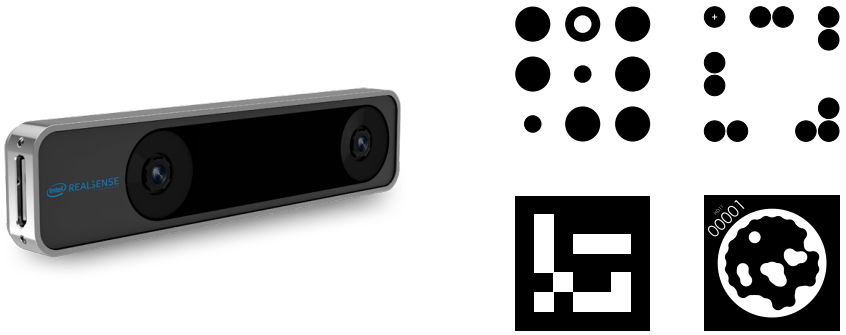
for each viewing angle between $0°$ and $90°$ in increments of $0.5°$. This is done for viewing distances of $0.075m$, $0.15m$, $0.4m$, $0.7m$ and $0.9m$. We generate multiple unique images for each combination of viewing angle and viewing distance by introducing a slight jitter in the camera position of $\pm 0.1mm$.

7.1.2 Real World Examples

As the basis for our real world evaluation we record several short sequences of our marker design printed on a sheet of A3 paper in a grid with three other fiducial markers. We select ArUco [6] as implemented in OpenCV [42], Pi-Tag [14] and STag [7] as reference fiducial systems to compare against. ArUco, as it is one of the most widely used systems and has proven reliable in real world applications. STag, as it promises a small improvement upon ArUco while using similar methods and Pi-Tag, as it represents a fiducial marker quite similar to ours with respect to using circles arranged in a square as the base fiducial features. The four markers are laid out in a 2×2 grid, as seen in Figure 7.2b, with each marker having a size of $10cm \times 10cm$ and a spacing of $4cm$ to neighboring markers. For ArUco we use a marker from the "Original" library and for STag one from the "HD11" library.

Recorded Sequences

Our recorded sequences cover a wide range of use cases, from varying distances and angles to different lighting conditions. We also include a sequence with noticeable motion blur. A full overview of all sequences can be found in Table 7.1. Important to note here is that each



(a) Intel RealSense T265 (b) Marker layout

Figure 7.2: Showcase of our real world dataset recording setup. (a) Intel RealSense Tracking Camera T265 [41]. (b) Layout of the markers used in our real world evaluation dataset: From top left to bottom right, our design, Pi-Tag [14], ArUco [6] and STag [7].

of the three sequences *low-angle-close*, *low-angle-mid* and *low-angle-far* are made up of four shorter sequences. As a low viewing angle on the test marker layout represents an advantage to the two markers closer to the camera, we record four sequences at low angles looking at the marker grid from each of the four possible directions.

Hardware

To record our sequences, we are using the Intel RealSense Tracking Camera T265 [41] which features a gray-scale stereo camera setup with built-in accelerometer and gyroscope, also called an inertial measurement unit (IMU). An image of the device can be found in Figure 7.2a. Its two stereo cameras manage a resolution of 848×800 with a field of view of $163° \pm 5°$ at a framerate of $30Hz$. We calibrate the camera using a recording of an AprilGrid calibration pattern which we use to fit a Kannala Brandt model for each of the two cameras. For this we use the calibration utilities of the Visual SLAM toolbox Basalt [43]. To obtain a comparable ground truth to be used when evaluating marker pose estimation accuracy, we use a motion capture system to track the position of the camera rig with respect to some arbitrary world coordinate frame. As the rotational center of the MoCap markers placed on the camera rig generally does not align with that of the cameras, we again use Basalt to obtain the transform between the MoCap measurements and the IMU coordinate frame (and therefore the camera coordinate frames) of the capture rig. This process works by solving a joint optimization problem for the transformations of camera to IMU and IMU to MoCap based on the MoCap poses, AprilGrid observations and IMU measurements. Secondly, due to our recording setup being split between different devices, the MoCap measurements and camera images are not time aligned. As clocks may drift over time, this time offset has to be computed and applied for each evaluation sequence individually. Basalt provides such a utility in which the time offset is computed by again aligning the gyroscope measurements with the MoCap rotational velocities and selecting the time offset which produces the smallest alignment error.

Sequence	low-angle-close (lac1)	low-angle-mid (lam1)	low-angle-far (laf1)	mixed-motion-1 (mm1)	mixed-motion-2 (mm2)	mixed-motion-3 (mm3)
Duration [s]	46.3	49.6	41.7	21.5	15.7	19.0
No. Frames	1388	1485	1214	646	470	570
Description	Low angle tilt at close distance	Low angle tilt at mid distance	Low angle tilt at far distance	Mixed motion	Mixed motion	Mixed motion

Sequence	pan-tilt-close (ptc1)	pan-tilt-mid (ptm1)	pan-tilt-far (ptf1)	planar-close (pc1)	planar-mid (pm1)	distance-1 (d1)
Duration [s]	23.0	22.1	25.0	12.7	15.9	17.4
No. Frames	690	663	709	379	477	519
Description	Combined pans and tilts at close distance	Combined pans and tilts at mid distance	Combined pans and tilts at far distance	Planar, close movement	Planar, mid range movement	Varying distances at constant angle

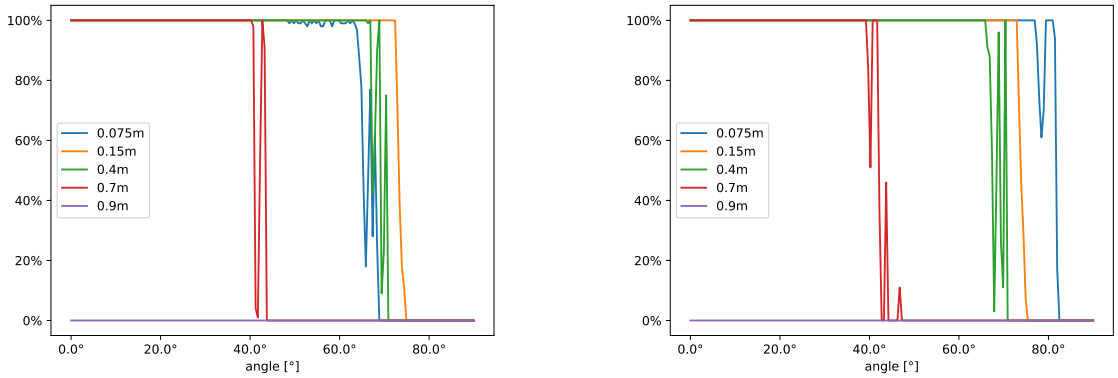
Sequence	distance-2 (d2)	shadows-1 (s1)	shadows-2 (s2)	shadows-3 (s3)	shake
Duration [s]	16.6	10.7	16.1	21.8	8.4
No. Frames	497	321	481	653	252
Description	Varying distances at constant angle	Markers in shadows	Markers in shadows	Markers in shadows	Motion blur movements

Table 7.1: Overview of all recorded sequences showing the respective length of the sequence in seconds, the number of image frames per sequence and a short description.

Evaluation Methodology

We follow the same procedure for all of the evaluations using our real world dataset. Every image in a sequence is run through each of the four marker detection pipelines. If a marker is detected we obtain the coordinates of its 2D feature points as well as the marker identifier. In the case of ArUco or STag these 2D points are the four corners of the marker, for Pi-Tag and our marker they are the centers of the ellipses making up the marker. We first compare the detected identifier with the correct one to rule out false positive detections for each marker. For better comparability we decided to not use the pose estimation built into some of the marker implementations and instead use the process introduced in Section 3.3 for all four markers. We also apply our false positive thresholding based on pixel reprojection error with regards to the detected fiducial features for each marker after pose estimation. As pose estimation is not possible in all applications, e.g. for camera calibration, we will show the number of false positives before performing pixel reprojection error thresholding.

7.2 Detection Precision



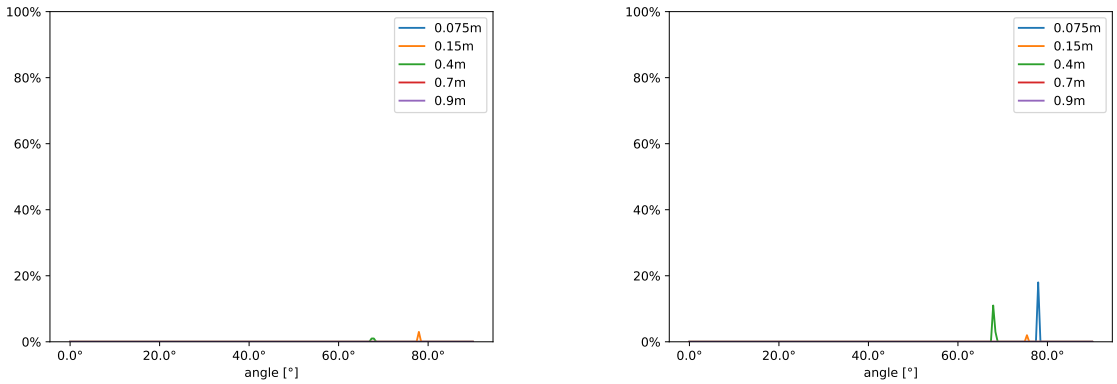
(a) Pinhole Camera

(b) Kannala Brandt Camera

Figure 7.3: Percentage of correct detections of a marker on 100 simulated frames plotted across viewing angles up to 90° for multiple viewing distances. Results are shown for a simulation using (a) a pinhole camera model as well as (b) a Kannala Brandt camera model.

This section will illustrate the performance of our novel fiducial marker design with regards to detection precision. As the base metric for this evaluation we will use the number of correct marker detections compared to the number of false positive detections in a given sequence. As we can see from the evaluation results on the simulated dataset in Figure 7.3, our marker design achieves consistent detections even in low viewing angles for viewing distances up to $0.4m$. When increasing the distance to $0.7m$ we can see a sharp drop in detections starting at angles around 45° to 50°. Finally, markers are not detected reliably at a distance of $0.9m$

as the individual ellipses making up a marker are only a few pixels in size in a captured image. The rate of false positive detections shown in Figure 7.4 highlights one of the current problems of our detection pipeline. When viewing a marker at a low angle it is quite hard to accurately distinguish between the different types of ellipses leading to code word decoding errors even with the geometry of a marker being detected correctly. The figures show this behavior quite clearly for viewing distances of 0.075, 0.15m and 0.4m, as the false positive rate has a short spike at the same point where the the detection accuracy decreases as shown in Figure 7.3.



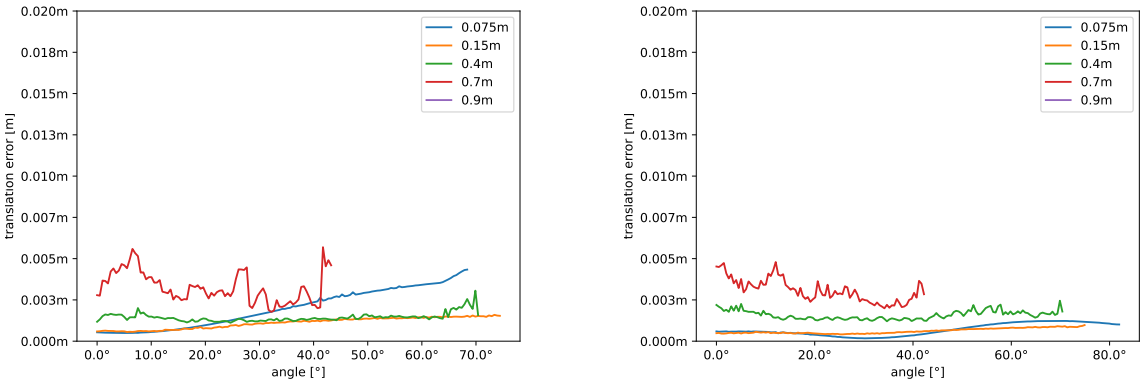
(a) Pinhole Camera

(b) Kannala Brandt Camera

Figure 7.4: Percentage of false positive detections of a marker on 100 simulated frames plotted across viewing angles up to 90° for multiple viewing distances. Results are shown for a simulation using (a) a pinhole camera model as well as (b) a Kannala Brandt camera model.

We compare the detection accuracy of our marker design with ArUco, Pi-Tag and STag in Table 7.2. The results highlight that while being outperformed by ArUco in most cases, our method clearly outclasses Pi-Tag as another circle based marker system and is comparable to STag in most cases. We would like to note that due to not assuming a projective geometry for initial marker candidate detection and code word decoding, our detection system is able to outperform even ArUco in the sequences *pan-tilt-close-1* and *planar-close-1*, where the camera is panned across the marker at a very close distance leading to large distortions when the marker is at the edge of the camera field-of-view. In many other sequences such as *low-angle-close-1*, *mixed-motion-2* or *shadows-1* our method achieves results which are quite comparable to those of ArUco. We would also like to highlight that our markers are still detectable quite reliably in the presence of larger motion blur as shown by the results for the sequence *shake*. One notable shortcoming of our method is its shorter detection range when compared to ArUco and STag as shown by the results for the sequences *low-angle-far-1* and *pan-tilt-far-1*.

When regarding the number of false positive detections shown in Table 7.3 it is clear that even though our method produces a noticeable amount of false positives for simulated edge



(a) Pinhole Camera

(b) Kannala Brandt Camera

Figure 7.5: Mean pose translation error of marker detections on 100 simulated frames plotted across viewing angles up to 90° for multiple viewing distances. Results are shown for a simulation using (a) a pinhole camera model as well as (b) a Kannala Brandt camera model. Errors are only shown for correct marker detections, hence the cutoff at 45° of the $0.7m$ graph.

cases this does not matter too much in real world applications with no false positive detections across all sequences.

7.3 Pose Estimation Accuracy

We evaluate the pose estimation accuracy of our method on our simulated dataset by comparing the estimated pose computed by the detection pipeline with the ground truth pose of the camera in the OpenGL simulation. We use the mean translation error (Figure 7.5) between the estimated pose and the ground truth as well as the mean rotation error (Figure 7.6) to showcase the pose estimation accuracy across a wide range of viewing angles and viewing distances. The mean pose translation error is simply computed as

$$e_{trans} = \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{t}_i - \boldsymbol{t}_{gt,i}\| \tag{7.1}$$

for a number of n images with respective estimated marker pose $g_i = (\boldsymbol{R}_i, \boldsymbol{t}_i)$ and ground truth pose $g_{gt,i} = (\boldsymbol{R}_{gt,i}, \boldsymbol{t}_{gt,i})$. Similarly the mean rotation error is computed as

$$e_{rot} = \frac{1}{n}\sum_{i=1}^{n}\arccos\left(\frac{\mathrm{tr}(\boldsymbol{R}_i\boldsymbol{R}_{gt,i}^T) - 1}{2}\right), \tag{7.2}$$

which represents the average angular difference between the ground truth and the estimated pose. We can clearly see from Figure 7.5 and Figure 7.6 that the pose estimation is especially

Marker System	d1	d2	lac1	laf1	lam1	mm1	mm2	mm3	ptc1	ptf1	ptm1	pc1	pm1	s1	s2	s3	shake
ArUco	**56.35%**	**59.36%**	**83.37%**	**79.23%**	**87.39%**	**89.78%**	**95.12%**	**93.17%**	55.65%	**15.51%**	61.54%	54.88%	**59.12%**	41.12%	9.56%	**75.19%**	71.03%
STag	37.88%	38.03%	63.00%	49.69%	79.65%	86.69%	94.06%	91.77%	60.58%	6.35%	**63.65%**	62.80%	49.48%	**41.43%**	**61.54%**	72.89%	**71.43%**
Pi-Tag	10.38%	10.87%	39.87%	0.18%	1.89%	0.00%	0.00%	0.00%	7.54%	0.56%	0.45%	0.26%	0.00%	17.45%	17.88%	22.97%	0.00%
Ours	39.04%	40.24%	82.34%	16.28%	79.11%	79.72%	88.11%	80.91%	**71.01%**	0.00%	46.30%	**66.23%**	28.72%	15.26%	0.00%	58.04%	66.27%

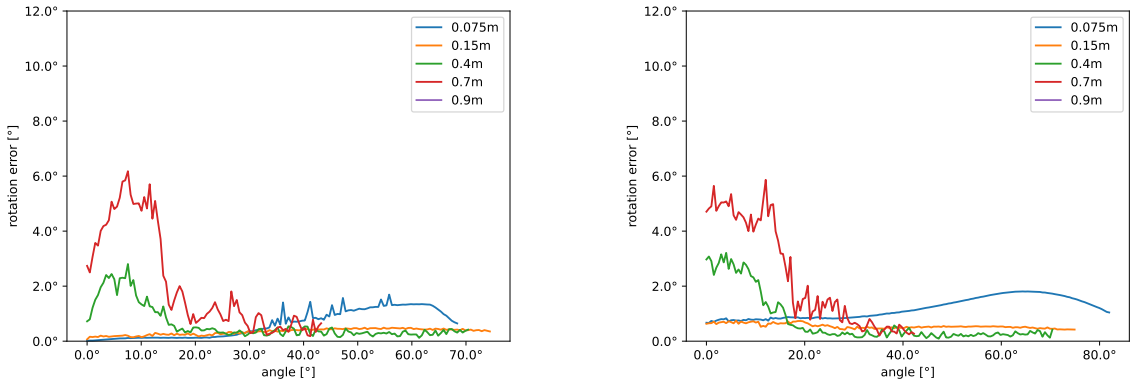
Table 7.2: Detection accuracy of our method compared to ArUco, Pi-Tag and Stag across our different real world datasets.

Marker System	d1	d2	lac1	laf1	lam1	mm1	mm2	mm3	ptc1	ptf1	ptm1	pc1	pm1	s1	s2	s3	shake
ArUco	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**
STag	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**
Pi-Tag	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**
Ours	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**	**0.00%**

Table 7.3: Percentage of frames with false positive detections of our method compared to ArUco, Pi-Tag and Stag across our different real world datasets.

Marker System	d1	d2	lac1	laf1	lam1	mm1	mm2	mm3	ptc1	ptf1	ptm1	pc1	pm1	s1	s2	s3	shake
ArUco	0.3369	0.3309	0.0096	0.3047	**0.0547**	0.2088	0.0974	0.1599	0.0442	0.4106	0.1232	0.0244	0.2186	0.0149	**0.0083**	0.0459	0.0808
STag	0.2048	0.1759	0.1024	0.3009	0.1064	0.1410	0.0913	0.1372	0.0853	0.3601	0.1274	0.1228	0.4186	0.0603	0.0947	0.1152	0.1186
Pi-Tag	0.0903	**0.0071**	0.0074	nan	nan	nan	nan	nan	0.0087	**0.2395**	0.1429	nan	nan	0.0420	0.0471	0.0485	nan
Ours	**0.0390**	0.0448	**0.0036**	**0.1879**	0.0734	**0.0329**	**0.0277**	**0.0377**	**0.0049**	nan	**0.0660**	**0.0056**	**0.1363**	**0.0136**	nan	**0.0084**	**0.0089**

Table 7.4: Root mean square translation error of pose estimation based on our method compared to ArUco, STag and Pi-Tag across all real data sequences. Sequences with too few detections to accurately compute an error are marked with nan.



(a) Pinhole Camera

(b) Kannala Brandt Camera

Figure 7.6: Mean pose rotation error in degree of marker detections on 100 simulated frames plotted across viewing angles up to 90° for multiple viewing distances. Results are shown for a simulation using (a) a pinhole camera model as well as (b) a Kannala Brandt camera model. Errors are only shown for correct marker detections.

stable for close viewing distances of less than $0.4m$. The larger spikes in estimation error at viewing angles lower than 20° for larger viewing distances can be explained due the fact that all ellipses lie on the same plane. This means artifacts such as camera jitter or imprecise pixel sampling in the simulation have a larger effect on the pose estimation if the marker is viewed head on instead of at a larger angle.

Accuracy on real data

Evaluating the pose estimation accuracy on our real dataset is more complicated, since the ground truth is the camera pose with respect to an arbitrary world frame as recorded by a motion capture system. To compare the estimated marker poses with the ground truth we perform a standard trajectory alignment between the marker pose trajectory and the camera trajectory, as these poses are set in different coordinate frames. Given a set of ground truth camera poses $\boldsymbol{Q}_{1:n} = \{\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_n \mid \boldsymbol{Q}_i \in SE(3)\}$ and corresponding estimated camera poses (inverse of the estimated marker pose) $\boldsymbol{P}_{1:n} = \{\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n \mid \boldsymbol{P}_i \in SE(3)\}$ these two trajectories can be aligned by finding a transform $\boldsymbol{S} \in SE(3)$ that minimizes the translation difference between each pair of poses. We now denote the pairwise pose difference between the two trajectories as

$$\boldsymbol{F}_i = \boldsymbol{Q}_i^{-1} \boldsymbol{S} \boldsymbol{P}_i. \tag{7.3}$$

A very common error metric in many 3D computer vision applications is to use the translation component $trans(\boldsymbol{F}_i)$ to compute the absolute trajectory error (ATE). The ATE is often

aggregated across the whole trajectory as a root mean square error (RMSE) formulated as

$$RMSE_{ATE}(\boldsymbol{F}_{1:n}) = \left(\frac{1}{n} \sum_{i=1}^{n} \| trans(\boldsymbol{F}_i) \|^2 \right)^{1/2}. \tag{7.4}$$

Table 7.4 shows the RMSE ATE for ArUco, STag, Pi-Tag and our method on our real world dataset for each of the individual dataset sequences. The results show that our marker design outperforms both ArUco and STag in terms of pose estimation accuracy. This is not surprising since both of those methods only use the four corners of their respective square marker border as point correspondences for pose estimation. Pi-Tag also achieves decent performance with low trajectory errors across all sequences as it uses the centers of twelve circles as point correspondences similar to our method. It is important to note, however, that the actual detection accuracy of Pi-Tag shown in Table 7.2 is much lower than that of all other three methods, which might indicate that markers are only detected in very favorable conditions which might skew pose estimation results.

7.4 Detection Runtime

An important aspect of fiducial marker designs is their runtime and ability to be parallelized, as many applications require real time capabilities. Table 7.5 shows the respective runtime of our detection pipeline steps for an 848×800 image with some clutter and one marker present. While our implementation has not been that heavily optimized for runtime it is quite clear that the connected component labeling process takes up over 60% of the runtime with the image binarization and search for ellipse candidates making up most of the remainder. This is not surprising as theses steps all perform at least one full pass over the input image,

Thresholding	Connected Components	Ellipse Candidate Search	Ellipse Fitting	Marker Candidate Search	Code Word Decoding	Pose Estimation	**Total**
3.77	16.00	3.90	0.89	0.31	0.04	1.13	**26.04**

Table 7.5: Runtime of the individual detection pipeline steps in milliseconds.

with the connected component labelling algorithm even performing two full passes. Still, our method achieves a per-frame runtime lower than the magical threshold of 33.3ms required for 30 frames per second in many real time applications. It is also notable that our method is faster than both the implementation of STag and Pi-Tag, as shown in Table 7.6, and is only outperformed by ArUco which has been optimized heavily, as the implementation has been part of OpenCV for quite a while. The slight difference in runtime shown in Table 7.5 and

Marker System	Runtime
ArUco	6.0ms
STag	55.9ms
Pi-Tag	69.2ms
Ours	27.8ms

Table 7.6: Average runtime per frame of the detection pipelines of ArCuo, STag, Pi-Tag and our method across all sequences in our real world dataset.

Table 7.6 of our method stems from our experimental setup, as we use Python bindings of our C++ implementation for large scale evaluations and pure C++ for smaller examples such as the one-off runtime evaluation show in Table 7.5.

We would also like to note that the detection pipeline can be parallelized quite easily as steps like image thresholding and parts of the connected component labelling process can be tackled in a divide and conquer approach. Additionally all of the later stages starting at the ellipse fitting operate on individual ellipses or marker candidates and can therefore also be parallelized.

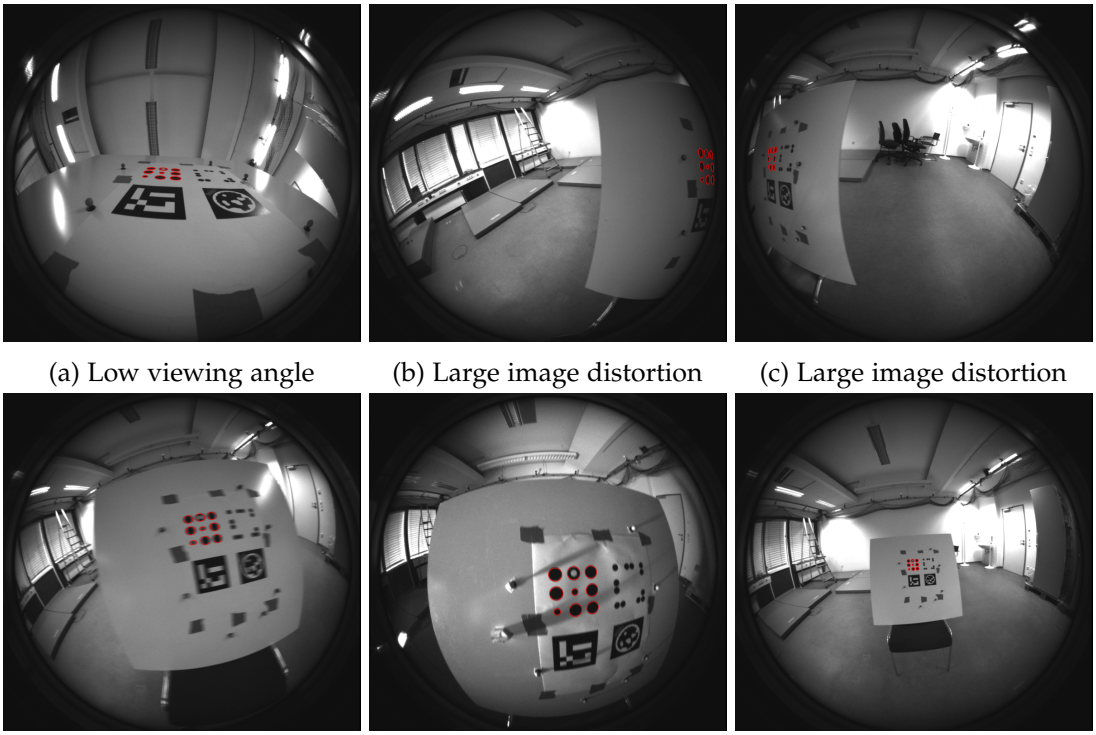
The only drawback of our method in terms of runtime is that the connected component labeling process can generate a large number of ellipse candidates in very cluttered scenes. This means we have to run a larger number of ellipse fittings as well as k-nearest neighbor searches which are quite expensive. Another effect of this property are dynamic changes in runtime as it is not solely dependant on the size of the input image, but also on the clutter (or distinct connected components) in the image, which might be undesirable in applications where a constant and predictable runtime for each processing step is required.

7.5 Qualitative Results

To also provide some more visual examples of our methods performance, Figure 7.7 shows some marker detections in very challenging situations. We can see that our markers are detected correctly at very low viewing angles, large viewing distances and even in the presence of significant motion blur or difficult lighting conditions with parts of the marker lying in shadow. We would also like to highlight that markers can be detected even at the very edge of the camera field-of-view which is especially valuable for the use case of camera calibration as it enables us to cover the whole camera lens when calibrating.

7.6 Failure Cases

This section aims to investigate some failure cases of our marker design. We will first introduce the results of an experiment in which we compare the false positive rate of our marker to that of ArUco and STag on a large scale image dataset which contains no actual marker images. Secondly, we will highlight some qualitative failure cases of our markers.



(a) Low viewing angle (b) Large image distortion (c) Large image distortion

(d) Significant motion blur (e) Difficult lighting (f) Large viewing distance

Figure 7.7: Qualitative results of marker detections in challenging situations.

7.6.1 False Positive Analysis

To give a better overview of our marker detection accuracy in more realistic settings outside a controlled laboratory environment we use a subset of the large scene recognition dataset provided by LabelMe [44] containing roughly 2920 images from different, mostly outdoor scenes, as well as the indoor scene recognition dataset [45] containing 15,620 images to represent markerless scenes. We showcase the number of false positive detections of ArUco, STag and our method in Table 7.7. As we can see from the table, our method produces a significantly larger amount of false positive detections than both ArUco and STag which is one of the current shortcomings of our method. This mostly stems from two aspects of our marker design. We first of all make very few assumptions about the geometry of a marker when performing initial detection steps which leads to more marker candidates in the early stages of our pipeline when compared to the straightforward and constrained search for quads in ArUco and STag. Additionally, we currently do not implement a more sophisticated approach to error detection than enforcing a Hamming distance of two within the Lyndon words which encode rotation in the tag corners and a simple checksum for the center ellipse. Since our overall library size is more constrained than that of larger ArUco variants and STag, adding a stricter error detection encoding would decrease our library size quite significantly, but we argue this would be tolerable as most applications do not need multiple hundred

Marker System	LabelMe	Indoor Scene Recognition
ArUco	**110 (3.77%)**	**193 (1.24%)**
STag	293 (10.03%)	308 (1.97%)
Ours	2672 (91.51%)	3146 (20.15%)

Table 7.7: False positive detections on the LabelMe [44] and indoor scene recognition [45] datsets of ArUco, STag and our method. Pi-Tag was excluded from this analysis due to its very expensive runtime on high resolution images. We show the percentage of images which produced at least one false positive detection in brackets.

unique markers. We would also like to add that even though a very large false positive rate is quite undesirable in a marker design, it does not matter much for many practical applications. In scenarios where pose estimation is performed using a calibrated camera, one can easily perform a final filtering based on the relative pixel reprojection error of the estimated pose. In the calibration use case a large false positive rate can be detrimental but can be adjusted for by using a fixed calibration grid which rules out many erroneous detections. Additionally, when using composed grids for camera calibration as introduced in Chapter 6, one can perform additional error detection and correction on the marker grid level. As a general note for the results in Table 7.7: The reason for the number of false positives of all markers being relatively higher on the LabelMe dataset is due to the images generally having a higher resolution of up to 2048×1536 pixel compared to the much lower resolution of around $200 - 400 \times 200 - 400$ pixels of the indoor scene recognition dataset. A larger image leads to more details being present which in turn increases the probability of an erroneous detection.

7.6.2 Qualitative Examples

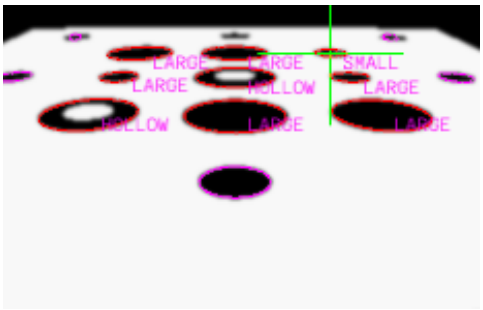
We want to use this section to give some qualitative examples of failed or erroneous detections of our marker design and also explain the reason behind those failures. As highlighted in Table 7.7 in the previous section, our method is quite prone to false positive detections, predominantly in images with high resolution. This can be seen again in Figure 7.8, especially in the left image, in which our method produces a large number of false positive detections. The reason for this behavior is the large amount of small image detail, in this case the foliage of a tree, which produces an enormous number of possible ellipses after ellipse candidate detection and ellipse fitting. Each part of the foliage which results in a separate connected component after thresholding is seen as an ellipse which could possibly be part of a marker. Due to the sheer number of detected ellipses, in this case 3524, the probability of one combination of 9 such ellipses matching the geometry of a tilted or distorted 3×3 grid is very high. Since we do not implement a sophisticated error detection scheme in our marker payload encoding, we currently cannot rule out all of the resulting false positive marker candidates. It can be seen though, that most of these false positive detections are somewhat irregular in terms of relative size and position of the detected ellipses. It is therefore reasonable to assume that they could be filtered out with additional heuristics, or would

Figure 7.8: Examples of false positive detections on the LabelMe dataset [44]. In images with high frequency image detail (left) erroneous detections are much more likely.

be discarded by the previously described reprojection error thresholding in case images are taken with a calibrated camera.

The second most common failure case which leads to false positive or missed detections is that some digits in a marker's code word are not recognized correctly. As shown in Figure 7.9a, it is possible that our current method of decoding the ellipse types from a marker candidate detection produces invalid code words, as e.g. small ellipses are accidentally detected as large ones. This stems from the fact that our current approximative ellipse size computation explained in Section 5.4 is not able to accurately capture the full range of possible viewing angles and their respective ellipse distortions.

The reason for our markers not achieving good detection recall for large viewing distances is that small and hollow ellipses grow too small or degenerate after adaptive thresholding, rendering them undetectable with our current ellipse detection pipeline. This can be seen in Figure 7.9b.



(a) Ellipse type detection failure

(b) Missed detections of small and hollow circles

Figure 7.9: Examples of wrongful of missed detections of our method. (a) Failure of our ellipse type decoding process. The small ellipses at the middle horizontal edges of the marker grid are erroneously detected as large ellipses leading either to a missed or false positive detection. (b) Missed detections of ellipses at a large viewing distance. Small ellipses are not detected due to too small sizes and hollow ellipses are broken up during adapting thresholding.

8 Conclusion and Future Work

This thesis introduces a novel design for fiducial markers based on a 3×3 grid of circles. The proposed method is specifically tailored to work well with large distortion cameras such as fisheye and large field-of-view lenses. Compared to traditional square fiducial markers such as ArUco [6] or STag [7] tags are detected based on the individual circles and their relative geometrical layout to each other instead of searching the image for large quads representing a square border around a marker. Since smaller image features are generally less prone to distortion, in addition to distorted circles still being recognizable as ellipses, our method achieves robust detection recall even in very distorted images. Similar to other circle based methods such as Pi-Tag [14] we also achieve a better pose estimation accuracy than ArUco or STag as our method uses the position of all nine circles in a marker as point correspondences for marker localization instead of just the four corners of a square marker border.

Another key feature of our marker design is its composability. Due to the geometric layout and coding system of our 3×3 circle grids, individual, overlapping markers can be assembled into larger grids without changing the underlying detection process. This can be useful for usage in camera calibration applications similar to AprilGrid in which the robust detections of our markers, even at the edges of distorted camera field-of-views, would really shine. A second potential use case is the ability of exploiting the very large number of unique 4×4, 5×5 and 6×6 marker grids to dramatically increase the library size of potential applications by using composed grids as individual fiducials instead of single 3×3 markers.

There are of course a number of possible improvements and avenues for future work to the approach presented in this thesis. For one, we only shortly introduced possible ways of incorporating more sophisticated error detection and error correction capabilities into our marker's coding system. Extending our initial and very simple idea of using Lyndon words for rotational uniqueness in addition to a crude checksum for the remaining marker payload could improve detection precision in many applications by eliminating some of the occurring false positive detections. We also only shortly glanced over possible implementations for detection schemes of composed marker grids in Chapter 6, which in addition to a more theoretical approach to marker grid generation could warrant some more in-depth research. Especially when looking at concrete options of performing error detection and correction not on single markers but on composed grids as a whole. In terms of general detection performance as shown in Chapter 7, our method is still lacking behind the implementation of ArUco in OpenCV. Investigating ways of improving the *k*-nearest-neighbor tag detection scheme presented in Chapter 5, as well as the approach to ellipse type detection would probably be the best way of improving detection recall to be on par with ArUco. Additionally, it would be quite interesting to see more experimental results on a wider range of camera lenses and resolutions.

Overall we regard our marker system as a valid alternative to state of the art methods such as ArUco in applications where a high detection recall is required even in heavily distorted images, as well as in use cases with stricter requirements on pose estimation accuracy. Our approach to marker composability also opens the door to more dynamic usages with differently sized markers, large numbers of 2D to 3D point correspondences and library sizes of up to multiple million uniquely identifiable grids.

Bibliography

[1] H. Kato and M. Billinghurst. "Marker tracking and HMD calibration for a video-based augmented reality conferencing system". In: *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*. 1999, pp. 85–94. DOI: 10.1109/IWAR.1999.803809.

[2] M. Fiala. "ARTag, a fiducial marker system using digital techniques". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. IEEE. 2005, pp. 590–596.

[3] D. Wagner and D. Schmalstieg. "Artoolkitplus for pose tracking on mobile devices". In: (2007).

[4] J. Wang and E. Olson. "AprilTag 2: Efficient and robust fiducial detection". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016.

[5] J. Wang and E. Olson. "AprilTag 2: Efficient and robust fiducial detection". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4193–4198.

[6] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer. "Speeded up detection of squared fiducial markers". In: *Image and Vision Computing* 76 (2018), pp. 38–47. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2018.05.004. URL: https://www.sciencedirect.com/science/article/pii/S0262885618300799.

[7] B. Benligiray, C. Topal, and C. Akinlar. "STag: A stable fiducial marker system". In: *Image and Vision Computing* 89 (2019), pp. 158–169.

[8] J. DeGol, T. Bretl, and D. Hoiem. "Chromatag: A colored marker and fast detection algorithm". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1472–1481.

[9] L. B. Gatrell, W. A. Hoff, and C. W. Sklair. "Robust image features: Concentric contrasting circles and their image extraction". In: *Cooperative Intelligent Robotics in Space II*. Vol. 1612. International Society for Optics and Photonics. 1992, pp. 235–244.

[10] Y. Cho, J. Lee, and U. Neumann. "A multi-ring color fiducial system and an intensity-invariant detection method for scalable fiducial-tracking augmented reality". In: *In IWAR*. Citeseer. 1998.

[11] L. Naimark and E. Foxlin. "Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker". In: *Proceedings. International Symposium on Mixed and Augmented Reality*. IEEE. 2002, pp. 27–36.

[12] F. Bergamasco, A. Albarelli, E. Rodolà, and A. Torsello. "RUNE-Tag: A high accuracy fiducial marker with strong occlusion resilience". In: *CVPR 2011*. 2011, pp. 113–120. DOI: 10.1109/CVPR.2011.5995544.

[13] L. Calvet, P. Gurdjos, C. Griwodz, and S. Gasparini. "Detection and Accurate Localization of Circular Fiducials under Highly Challenging Conditions". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 562–570. DOI: 10.1109/CVPR.2016.67.

[14] F. Bergamasco, A. Albarelli, and A. Torsello. "Pi-Tag: A fast image-space marker design based on projective invariants". In: *Machine Vision and Applications* 24 (Aug. 2013). DOI: 10.1007/s00138-012-0469-6.

[15] J. Sattar, E. Bourque, P. Giguere, and G. Dudek. "Fourier tags: Smoothly degradable fiducial markers for use in human-robot interaction". In: *Fourth Canadian Conference on Computer and Robot Vision (CRV'07)*. IEEE. 2007, pp. 165–174.

[16] A. Xu and G. Dudek. "Fourier tag: A smoothly degradable fiducial marker system with configurable payload capacity". In: *2011 Canadian Conference on Computer and Robot Vision*. IEEE. 2011, pp. 40–47.

[17] E. Costanza and J. Robinson. "A Region Adjacency Tree Approach to the Detection and Design of Fiducials." In: (2003).

[18] E. Costanza, S. B. Shelley, and J. Robinson. "D-touch: A consumer-grade tangible interface module and musical applications". In: (2003).

[19] C. N. Klokmose, J. B. Kristensen, R. Bagge, and K. Halskov. "BullsEye: High-Precision Fiducial Tracking for Table-Based Tangible Interaction". In: ITS '14. Dresden, Germany: Association for Computing Machinery, 2014, pp. 269–278. ISBN: 9781450325875. DOI: 10.1145/2669485.2669503. URL: https://doi.org/10.1145/2669485.2669503.

[20] R. Bencina and M. Kaltenbrunner. "The design and evolution of fiducials for the reactivision system". In: *Proceedings of the Third International Conference on Generative Systems in the Electronic Arts*. Vol. 2. Monash University Publishing Melbourne, VIC, Australia. 2005.

[21] R. Bencina, M. Kaltenbrunner, and S. Jorda. "Improved topological fiducial tracking in the reactivision system". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*. IEEE. 2005, pp. 99–99.

[22] M. Kaltenbrunner and R. Bencina. "reacTIVision: a computer-vision framework for table-based tangible interaction". In: *Proceedings of the 1st international conference on Tangible and embedded interaction*. 2007, pp. 69–74.

[23] G. Yu, Y. Hu, and J. Dai. "TopoTag: A Robust and Scalable Topological Fiducial Marker System". In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 27.9 (2021), pp. 3769–3780.

[24] B. Wang. "LFTag: A Scalable Visual Fiducial System with Low Spatial Frequency". In: *2020 2nd International Conference on Advances in Computer Technology, Information Science and Communications (CTISC)*. 2020, pp. 140–147. DOI: 10.1109/CTISC49998.2020.00030.

[25] A. Fitzgibbon, M. Pilu, and R. B. Fisher. "Direct least square fitting of ellipses". In: *IEEE Transactions on pattern analysis and machine intelligence* 21.5 (1999), pp. 476–480.

[26] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Vol. 7. Cambride University Press, 2010.

[27] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An invitation to 3-d vision: from images to geometric models*. Vol. 26. Springer, 2004. DOI: 10.1007/978-0-387-21779-6. URL: doi.org/10.1007/978-0-387-21779-6.

[28] V. Usenko, N. Demmel, and D. Cremers. "The Double Sphere Camera Model". In: *Proc. of the Int. Conference on 3D Vision (3DV)*. Sept. 2018. URL: https://arxiv.org/abs/1807.08957.

[29] J. Kannala and S. S. Brandt. "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.8 (2006), pp. 1335–1340.

[30] L. Kneip, D. Scaramuzza, and R. Siegwart. "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation". In: *CVPR 2011*. IEEE. 2011, pp. 2969–2976.

[31] V. Lepetit, F. Moreno-Noguer, and P. Fua. "Epnp: An accurate o (n) solution to the pnp problem". In: *International journal of computer vision* 81.2 (2009), pp. 155–166.

[32] L. Kneip, H. Li, and Y. Seo. "Upnp: An optimal o (n) solution to the absolute pose problem with universal applicability". In: *European Conference on Computer Vision*. Springer. 2014, pp. 127–142.

[33] L. Kneip and P. Furgale. "OpenGV: A unified and generalized approach to real-time calibrated geometric vision". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1–8.

[34] J.-N. Ouellet and P. Hébert. "Precise ellipse estimation without contour point extraction". In: *Machine Vision and Applications* 21.1 (2009), pp. 59–67. DOI: 10.1007/s00138-008-0141-3. URL: https://doi.org/10.1007/s00138-008-0141-3.

[35] J. P. Duval. "Factorizing words over an ordered alphabet". In: *Journal of Algorithms* 4.4 (1983), pp. 363–381.

[36] D. Bradley and G. Roth. "Adaptive Thresholding using the Integral Image". In: *J. Graphics Tools* 12 (Jan. 2007), pp. 13–21. DOI: 10.1080/2151237X.2007.10129236.

[37] K. Wu, E. Otoo, and K. Suzuki. "Optimizing two-pass connected-component labeling algorithms". In: *Pattern Analysis and Applications* 12.2 (2009), pp. 117–135.

[38] J. L. Blanco and P. K. Rai. *nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees*. https://github.com/jlblancoc/nanoflann. 2014.

[39] P. Furgale, J. Maye, J. Rehder, and T. Schneider. *Kalibr*. 2014. URL: https://github.com/ethz-asl/kalibr (visited on 05/06/2022).

[40] L. de Moura and N. Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and J. Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78800-3.

[41] Intel Corporation. *Intel® RealSense™ T265 product website*. 2019. URL: https://www.intelrealsense.com/tracking-camera-t265/ (visited on 08/09/2019).

[42] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[43] V. Usenko and N. Demmel. *Basalt*. 2019. URL: https://gitlab.com/VladyslavUsenko/basalt/ (visited on 08/20/2019).

[44] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. "LabelMe: a database and web-based tool for image annotation". In: *International journal of computer vision* 77.1 (2008), pp. 157–173.

[45] A. Quattoni and A. Torralba. "Recognizing indoor scenes". In: *2009 IEEE conference on computer vision and pattern recognition*. IEEE. 2009, pp. 413–420.