



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

# REAL-TIME CAMERA POSE ESTIMATION FOR AUGMENTED REALITY

URČENÍ POZICE KAMERY V REÁLNÉM ČASE PRO ROZŠÍŘENOU REALITU

DOCTORAL THESIS

DISERTAČNÍ PRÁCE

AUTHOR

AUTOR PRÁCE

Ing. ISTVÁN SZENTANDRÁSI

SUPERVISOR

ŠKOLITEL

prof. Ing. ADAM HEROUT, Ph.D.

BRNO 2016

## Abstract

Fiduciary markers form the base for camera pose estimation for many Augmented Reality applications, when a fast and robust solution is required. In this thesis an efficient marker-based camera pose estimation is outlined for Uniform Marker Fields with several real-world applications. The proposed algorithm is highly efficient and works in real time even on multiple platforms, including mid-range smartphones. On-screen markers are used to establish relative pose between devices for task-migration and information reaccess. In movie production, the described camera pose estimation as part of the chromakeying process, provides content creators real-time preview. Results show that the described detection algorithm performs comparably to other marker-based methods and it is several times faster. The implemented applications provide viable – cheaper and faster – alternative to existing solutions.

## Abstrakt

Definované markery tvoří základ určování polohy kamery pro velké množství aplikací s rozšířenou realitou, v případě že jsou přísné požadavky na rychlost a robustnost. Tato práce popisuje účinnou metodu pro určení pózy kamery pomocí Uniformního pole markerů a několik realistických aplikací na bázi popsané metody. Metoda je velice výpočetně levná a poskytuje spolehlivou detekci pro několik výpočetních platforem, včetně běžných chytrých telefonů. Markery jako část zobrazené informace na monitorech jsou použité v této práci pro určení relativní orientaci mezi poskytovatelem obsahu a uživatelským zařízením, sloužícím pro výběr prvků uživatelského rozhraní při interakci a migraci úkolů. Ve filmářském průmyslu poskytuje popsaná metoda pro zjištění polohy kamery jako součást klíčování pozadí filmářům živý náhled virtuální scény. Výsledky ukazují, že popsaná metoda pro detekci pole markerů má srovnatelnou úspěšnost a přesnost v porovnání s ostatními metodami na bázi markerů a je několikrát rychlejší. Aplikace zahrnuté v této práci podle výsledků testů jsou životaschopné – rychlejší a levnější – alternativy k existujícím řešením.

## Keywords

Camera pose estimation, Fiduciary Markers, Uniform Marker Field, Checkerboard Detection, Overlapping Markers, Augmented Reality, de-Bruijn Tori, Perfect Maps, Real-Time AR Localization, Visually Appealing Markers, Task Migration, Uniform Marker Fields, Document Reaccess, Fast Localization, Visual Codes, Multi-Device Interaction, Chroma Key, Green Screen, MatchMoving, Film Tricks, Movie Production

## Klíčová slova

Určení polohy kamery, Definitivní markery, Detekování šachovnic, Překryté makery, Rozšířená realita, de-Bruijn Tori, Perfektní mapy, Určování polohy v reálném čase, Klíčování, Greenscreen, Filmářské efekty

## Bibliographic citation

István Szentandrás: Real-time Camera Pose Estimation for Augmented Reality, doctoral thesis, Brno, Brno University of Technology, Faculty of Information Technology, 2016

# Real-time Camera Pose Estimation for Augmented Reality

## Declaration

I declare that this dissertation thesis is my original work and that I have written it under the guidance of prof. Ing. Adam Herout, Ph.D. All sources and literature that I have used during my work on the thesis are correctly cited with complete reference to the respective sources

.....  
István Szentandrás  
8. april 2016

## Acknowledgment

I would like to thank Adam Herout, Jiří Havel, Rudolf Kajan and Michal Zachariáš for being excellent colleagues and helping me along. I would like to thank all members of Department of Computer Graphics and Multimedia for creating a creative and productive academic environment.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Summary of Contributions . . . . .	2
1.2	Authorship . . . . .	3
1.3	Text Structure . . . . .	4
<b>2</b>	<b>Camera Pose Estimation and Tracking in Augmented Reality Systems</b>	<b>5</b>
2.1	Camera Calibration . . . . .	7
2.2	Perspective- $n$ -Point Problem . . . . .	9
2.3	Vision-based Tracking Techniques . . . . .	10
2.3.1	Approaches Based on Fiduciary Markers . . . . .	11
2.3.2	Camera Tracking Based on Feature Points . . . . .	19
2.3.3	Model Based Tracking Methods . . . . .	21
2.4	Hybrid Tracking Methods . . . . .	22
2.5	Augmented Reality Applications . . . . .	23
<b>3</b>	<b>Uniform Marker Fields</b>	<b>27</b>
3.1	Orientable Window Arrays as Marker Fields . . . . .	27
3.2	Synthesis of Binary $n^2$ -Window Arrays . . . . .	29
3.2.1	Random Maps . . . . .	30
3.2.2	Genetic Algorithm for Synthesis of Window Arrays . . . . .	32
3.2.3	Synthesized Window Arrays . . . . .	33
3.3	Detection of Marker Fields and Camera Pose Estimation . . . . .	33
3.3.1	Overview . . . . .	34
3.3.2	Extraction of Edgels . . . . .	35
3.3.3	Identifying Vanishing Points . . . . .	36
3.3.4	Finding Two Fans of Grid Lines . . . . .	37
3.3.5	Extraction of the Checker-Board Modules . . . . .	37
3.3.6	Location Recovery and Camera Pose Estimation . . . . .	38
3.3.7	Experimental Results . . . . .	39
3.4	Five Shades of Gray . . . . .	42
3.4.1	Improved Uniform Marker Fields Design . . . . .	42
3.4.2	Recognition of Planar Greyscale Grids of Squares . . . . .	43
3.4.3	Experimental Results . . . . .	47
3.4.4	Realistic Dimensions of the Marker Field . . . . .	52
3.5	Follow-up and Related Research . . . . .	54
3.5.1	Generalizations and Adaptations of UMF . . . . .	54
3.5.2	Indoor Localization by UMF . . . . .	56

<b>4</b>	<b>Fast Detection of Grids and Matrix Codes</b>	<b>59</b>
4.1	Motivation . . . . .	60
4.2	QR code Detection . . . . .	61
4.3	Rapid Detection of Regions with a Possible QR code . . . . .	62
4.3.1	Processing in Tiles . . . . .	62
4.3.2	Multiscale Processing . . . . .	64
4.4	Experimental Results . . . . .	65
4.4.1	Candidate Position Search . . . . .	66
4.4.2	Detector Evaluation . . . . .	66
4.4.3	Speed Evaluation . . . . .	69
4.5	Conclusions . . . . .	70
<b>5</b>	<b>On-screen Markers</b>	<b>71</b>
5.1	Screen-to-Screen task migration using Uniform Marker Fields . . . . .	72
5.1.1	Previous Work and Motivation . . . . .	73
5.1.2	Proposed System Architecture . . . . .	75
5.1.3	Implemented Solution – Chrome and Android . . . . .	79
5.1.4	Experiments: Empirical Tests and User Study . . . . .	81
5.1.5	Conclusion . . . . .	87
5.2	Continuous Task Migration using Natural Features . . . . .	88
5.2.1	Marker Tracking and Natural Key-points Based Detection . . . . .	89
5.2.2	Content Requester Android Application . . . . .	90
5.2.3	Experiments . . . . .	91
5.2.4	Conclusion . . . . .	94
<b>6</b>	<b>Poor Man’s Virtual Camera</b>	<b>97</b>
6.1	Motivation . . . . .	97
6.2	Greenscreen Marker Field . . . . .	99
6.3	Matchmoving With Shades of Green . . . . .	101
6.3.1	Selection of Suitable Shades of Green . . . . .	101
6.3.2	Mapping . . . . .	102
6.3.3	Matting for Visualization . . . . .	103
6.3.4	Practical Set-Up: Projection . . . . .	103
6.3.5	Practical Set-Up: Special Canvas . . . . .	104
6.4	Experiments and Evaluation . . . . .	104
6.4.1	Unity 3D Plug-In . . . . .	104
6.4.2	Detection Rates . . . . .	105
6.4.3	Computational Complexity . . . . .	106
6.4.4	Preview Precision . . . . .	107
6.4.5	Matting Precision . . . . .	107
6.4.6	Sensitivity to Edge Contrast . . . . .	107
6.5	Results . . . . .	108
<b>7</b>	<b>Conclusion</b>	<b>111</b>



# Chapter 1

## Introduction

Augmented Reality (AR) is viewed as a variation of Virtual Environments. While immersed in a Virtual Environment, the user is limited to seeing only virtual objects. Augmented reality, on the other hand, simply enhances the real world with virtual objects. Azuma [9] proposed a commonly accepted definition of augmented reality systems from 1997. Such system is required to have three characteristics: combines real and virtual environment, it is interactive in real time and registered in 3D. This definition does not allow simple 2D overlays or (non-interactive) movie effects. On the other hand, augmented reality based applications on contemporary mobile devices clearly fit all the required characteristics.

Presumably, the first augmented reality interface was introduced in the 1960's by Sutherland [153]. Since then, augmented reality has found utilization in many fields from industry through military to medical care and training. The availability and the number of possible applications of augmented reality systems rose recently considerably thanks to the advances in mobile computing, which made powerful, yet small and cheap multimedia devices a commodity. Nonetheless, the progress of technology on the hardware side still lacks behind in unobtrusiveness and a breakthrough for augmented reality to be used in everyday life is still in the future.

Aside from the hardware limitations, the current state of research in technology is mostly focused on sub-parts of an augmented reality system, which only enable the theoretical usage of such systems. Exhaustive research on user interactions, on integration into real-world applications and establishing common usage patterns is still missing. In other areas, especially in real-time realistic rendering of virtual objects and tracking of real-world objects, great progress has been made.

This work is focused on real-time camera pose tracking. The current state of the art in several areas, such as marker-based tracking, feature point matching and tracking has achieved great maturity. Objectively, there is still progress to be made in this field to enhance robustness, reduce computational complexity and increase scalability to mobile devices. In this work I included an overview of the above mentioned tracking of the camera pose relative to real-world objects. I pinpointed current limitations and established possible future directions with a focus on methods with extremely low computational and memory requirements.

For augmented reality applications and other similar problems in computer vision, camera localization within a captured scene is crucial. Camera localization can be done either by using fiducial markers [43] or without them (by using PTAM [79], keypoint template tracking [167], homography-based [127], etc.). In this thesis, I investigated and improved upon marker-based approaches.

I also analyzed new use-cases and scenes where fiducial markers are acceptable. A challenge in this area is the requirement for detection and localization algorithm to be extremely fast (to

work in real time on mid-level ultramobile devices) and to localize the camera from a single frame (i.e. without temporal tracking and mapping). I have done comprehensive testing of the proposed systems and algorithms.

For many use-cases it is acceptable to allow for perfectly planar markers – placed on a tabletop, a wall, computer screen, etc. The challenge in this case is that the marker must cover a large planar area and, at the same time, it must be reliably detected even from a small visible portion of the marker. Also, the detection must be invariant to high degrees of perspective distortion and to varying lighting conditions (direct light, shadows, different lighting intensities). What marker design and corresponding detection algorithm can meet these requirements and, at the same time, be aesthetically appealing?

Conventional approaches use disjoint markers such as the ARTag [41], ALVAR [2], or CAL-Tag [8]. Marker-based solutions such as ARTag and ALVAR (a number of other similar solutions exists) are using square black-and-white markers with their identity digitally encoded. One part of the marker’s design is used for the marker’s localization (typically the outer black/white rim) and another part is used for distinguishing between individual markers (typically inner content of the square). An array of such individual markers is used to cover a larger planar area. CALTag [8] alters black-and-white with white-and-black (inverse) markers and attaches the markers one to another.

An approach to be able to detect the camera position even from a small part inside a larger marker are the Random Dot Markers [166] by Uchiyama et al. They are detecting and tracking fields of randomly displaced dots on a solid background using geometric features. The field of random dots can also be used as a deformable marker [165].

None of the above mentioned approaches fulfills the requirements set as the goals of this work: scalable size, reliable and robust detection, and efficient algorithm suitable for ultra-mobile devices. ALVAR, ARTag, CALTag and similar approaches have very efficient algorithms, but allow only small individual markers and require complex setup and calibration, if a larger area is to be used. Random Dot Markers’ marker design theoretically allows for scalable sizes, but the detection algorithm is far from efficient.

These limitations lead to the development of Uniform Marker Fields by me and my colleagues. My most important contribution was the research of efficient algorithmic approaches and their maximally efficient implementation on multiple computing platforms. The marker design and synthesis were done as a joint research with Michal Zachariáš, Adam Herout, Jiří Havel – my contribution to these parts was secondary. Based on the developed technology of UMF, we opened space for a few distinct applications.

## 1.1 Summary of Contributions

This thesis contributes to the state of the art of fast camera localization using artificial markers. It explores the possibilities of designing artificial markers for efficient and precise camera pose estimation and opportunities to utilize these advantages in the fields of augmented reality, movie production and human-computer interactions. This work describes the design decisions and proposed algorithms for efficient detection of the Uniform Marker Fields.

The proposed algorithms and proposed utilization of the designed systems are general in the sense that they are not limited to Uniform Marker Fields. Parts of the detection algorithm for the Uniform Marker Fields and applications are separate contributions and are ready to be re-used in other areas. The most notable of these contributions follow.

### **Efficient detection of planar grid structures using vanishing points.**

In the industrial manhattan world, the occurrence of tile-based structure is frequent. Many



marker based approaches rely on tiles of black and white fields to encode information (ARTag, ALVAR, QR code) or to get reliable points for the camera pose estimation. However, these approaches use only corners or special local image features to localize the markers (silent areas, length-ratio on line segments, circular patterns, etc.). The proposed method in this thesis uses a global approach to detect the grid of tiles as a whole. This is the key part of the camera pose estimation algorithm for the Uniform Marker Fields.

### **Novel approach to real-time virtual camera.**

Contemporary virtual camera systems used in movie production to replace image segments with virtual objects use complex and expensive hardware and software setups. A challenging component in these systems is the real-time camera pose estimation for live scene previews and storyboarding. This thesis describes an approach that works on commodity mobile devices in real time.

### **Estimating relative pose for human computer interactions.**

The growing number of user-owned smart devices equipped with camera opened the door towards new inter-device interaction techniques. Visual one-time transfer of data with limited size between devices already exists (QR codes, VR codes). This thesis discusses a novel interaction technique that uses continuous information flow for interaction. This is achieved by establishing and tracking the smart devices' camera pose relative to the information provider.

### **Cross-platform efficient implementation of proposed methods in real-world use cases.**

With technological advancement, the number of available computational platforms grows. Mobile architectures focus on low power consumption with rich support for auxiliary sensors, while desktop architectures aim for maximum possible performance and ease of development. The methods described in this thesis were implemented with both these platforms in mind. An efficient, low-memory footprint algorithm is especially important on mobile platforms, where the computational power is relatively low.

## **1.2 Authorship**

Although most of the work presented in this thesis is my own, some parts resulted from a collaboration with colleagues.

Adam Herout has contributed to my work with many ideas and consultations. He proposed first the usage of de Bruijn sequences as a perspective direction to solve the limitations of state-of-the-art marker designs. The initial visual design and synthesis of such markers (Uniform Marker Fields) were done as a joint research with Michal Zachariáš, Adam Herout and Jiří Havel – my contribution to these parts was secondary. My contribution related to Uniform Marker Fields was the proposal of an efficient detection method, its refinement into a practical algorithm and experimental evaluation. Jiří Havel's research and consultations were pivotal during this process, who laid down the mathematical bases of efficient vanishing point detection.

Markéta Dubská proposed the first basic principle of using fiduciary markers as part of a green-screen for cheap camera pose estimation in movie production. I refined this idea into a practical algorithm and tested in the experiments presented in this thesis.

Rudolf Kajan proposed the system for continuous inter-device communication. This system included the module for relative pose estimation using the camera stream as a bases for interactions. The parts relevant to this module in this thesis are my own work.

## 1.3 Text Structure

This thesis covers a wide range of research areas ranging from computational geometry, through low-level computer vision to real-time rendering. Given the difficulty to separate state of the art and contributions without sacrificing clarity for the reader and natural flow of the text, I describe some of the findings from secondary research areas only as they become relevant.

The discussion of existing publications for the main research area: augmented reality and camera pose estimation are described in Chapter 2. This is followed by the introduction of Uniform Marker Fields (Chapter 3). In order to prove practical value of the Uniform Marker Fields, it was adapted to fit various settings. These are described along with the state of the art of the relevant research fields in Chapter 5 and Chapter 6.

## Chapter 2

# Camera Pose Estimation and Tracking in Augmented Reality Systems

Augmented reality systems are complex systems consisting of many individual sub-parts or sub-modules from a broad range of fields concerning computer vision, computer graphics, hardware sensors, robotics, etc. The research on augmented reality is in consequence highly fragmented.

An up-to-date definition of an augmented reality system is elusive in the literature. Almost 20 years before writing this thesis, Azuma [9] provided a definition, that has been generally accepted by the research community. He states, that an augmented reality system has to have the following three characteristics:

1. Combines real and virtual
2. Interactive in real time
3. Registered in 3-D

This definition was associated only with visual systems and did not consider haptic environments, although the ambiguity of the definition allows them. There is also confusion, if 2-D overlays could be considered part of this definition. Azuma [9] states, that “it does not include film or 2-D overlays”. Yet, 2-D overlays, where only the on-screen position of the overlay is changed to match the real 3-D object’s on-screen position is accepted as an Augmented Reality system in the literature [148]. Such system clearly does not have to fulfill the third point of the definition and only partially the first point.

Recent advancements in the field are pushing the boundaries of what could be considered an augmented reality system. *SoftAR* [129] changes only the appearance of real objects to translate haptic into visual feedback, challenging the concept of mixing real and virtual objects in an AR system. Similarly, *IllumiRoom* [70] brings the virtual world out from the conventional screen-space dimensions. Bork et al. [22] combine visual and auditory feedback showing that multiple sensory feedback is beneficial. These are only a few examples to demonstrate the wide spectrum AR applications cover and how open the concept of an augmented reality system have become.

The overwhelming majority of contemporary AR systems solutions use exclusively visual information. Even the strict definition of augmented reality systems by Azuma [9] does not specify the characteristics of devices used for registering 3D position. Besides optical sensors, as the most common sensor used at present, magnetic, acoustic, inertial, GPS, mechanical and other sensors can be used. Using captured images alone for 3D registering is sometimes insufficient and require relatively large computation power.

The aforementioned inertial and location sensors all represent open-loop systems. All of them have their advantages and disadvantages [138], but in recent systems they are rarely used alone. They appear primarily in combination with visual sensors. They are used for prediction, enhancing precision and robustness. Research in this area is mainly focuses on how to combine the inputs from different sensors. Newman et al. [109] proposed an approach, called Ubiquitous Tracking, where heterogeneous and widespread tracking sensors are automatically and dynamically fused. More recently Oskiper et al. [117] combined monocular camera, MEMS-type inertial measuring unit with 3-axis gyroscope, accelerometers and a GPS unit using an error-state extended Kalman Filter for wide-area augmented reality. An exception to using visual sensors as primary sources is a system developed by Newman et al. [108] based on an ultrasonic tracking system called Bat system, which could be used for wide-area augmentation using head-mounted display and a PDA.

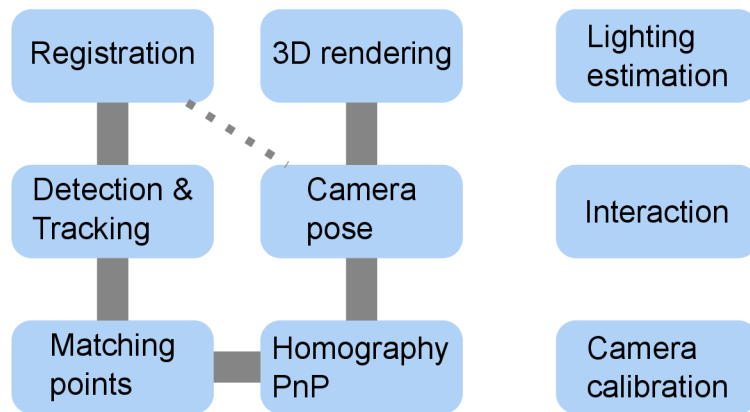


Figure 2.1: Simplified model of an augmented reality system. For each iteration, the system starts with gathering sensory input (*Registration*) and provides the user with an augmented view (*3D rendering*). Visual data, as the main source for camera pose estimation, needs to be further processed (*Detection & Tracking*, *Matching points*, *Homography – PnP* calculations). Other sensory information, like GPS or IMU, can be directly used for camera pose estimation. An AR system optionally includes several other modules. I gave three examples: *Lighting estimation* for realism, *Interaction* with virtual objects, *Camera calibration* for improved precision.

For the remainder of this work, I will focus on augmented reality systems using cameras as main input sensors. Figure 2.1 contains a simplified model of such a system. The first step is registration – acquiring the input from different sensors. The visual data is then further processed to find important edges, corners, other reliable feature points and markers in the image. These points are then matched based on the model to 3D positions. The correspondences between the 2D and 3D points can be used to calculate a homography, to get the global 6 degrees-of-freedom camera pose (position, rotation). As discussed above, other sensor input can optionally be used to improve precision (dashed line between *Registration* and *Camera pose* in Figure 2.1). Using the knowledge about the camera’s internal parameters and its position and rotation, the system is able to augment the captured image or video with virtual objects.

The three further modules: *Lighting estimation*, *Camera calibration* and *Interaction* are also important to achieve realistic results, good user experience and precise camera position estimation. Unfortunately, the research on interaction with the virtual environment is still in early stages.

Lighting model estimation so far has been approximated using light probes or secondary fish-eye cameras. Real-time lighting estimation from visual data is currently an active area of research. Jachnik et al. [66] presented an algorithm to estimate light-field represented by spherical harmonics

using specular surfaces by separating the diffuse and specular components of lighting. Gruber et al. [50] estimated diffuse-only lighting using real-time geometric reconstruction with the help of an RGBd camera. Even though both the aforementioned solutions ran in real-time, they both used a powerful desktop GPUs for computations. There is still research to be done before lighting estimation would be possible on mobile platforms.

## 2.1 Camera Calibration

In conventional AR systems, the user sees the augmented world through a display. The display is usually head mounted or hand-held. The display provides the user a view of the real world augmented with virtual objects. In order to correctly place virtual objects into the field of view of the observer device, a precise camera pose has to be calculated relative to the real world.

Before any homography calculations could take place, the algorithms usually require prior knowledge about the camera's intrinsic parameters to be able to calculate a metric 3D camera pose [155]. This information is either provided by the manufacturer of the lenses and chips, or it is acquired using camera calibration. For the camera calibration for monocular AR applications a pin-hole camera representation is used with constant intrinsic parameters. Techniques estimating these parameters can be classified into two main groups: photogrammetric calibration and self-calibration.

For *photogrammetric calibration* a well defined 3D object with precise dimensions is observed with the camera in several positions. The object can be a full 3D model [40], 2D plane [187, 115] or even a simple 1D line [188]. An unconventional calibration method for static pin-hole camera's was proposed by Dubská et al. [37]. They used the trajectories of automotive vehicles with statistical data of their sizes to compute vanishing points and consequently estimate the road-side camera's intrinsic and distortion parameters.

The second class of techniques, *self-calibration*, does not require any calibration objects. Instead, most methods in this category restrict allowed camera movement [57] or rotation [101] and presume a static scene. Luong et al. [90] showed that three images taken with the same camera of a static scene are sufficient to recover both intrinsic and extrinsic parameters of the camera. Given the large number of parameters required by these methods, however, they tend to provide less stable and less reliable results.

A well-known and often used method for camera calibration was proposed by Zhang et al. [187, 186], thanks to its simplicity and easy setup. It uses several images of plain chess-board structure with different rotations and positions. For the next part I will follow their notation. Their algorithm assumes a pinhole camera, where the relationship between the projected  $\tilde{\mathbf{p}}$  2D point and real 3D position  $\tilde{\mathbf{P}}$  is given by:

$$s\tilde{\mathbf{p}} = \mathbf{A} \begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix} \tilde{\mathbf{P}}, \quad (2.1)$$

where  $s$  is a scale factor,  $\mathbf{R}$  and  $\mathbf{t}$  are the rotation and translation of the camera, and  $\tilde{\phantom{x}}$  means homogenous coordinates. In their camera model,  $\mathbf{A}$  is the camera intrinsic matrix given by

$$\mathbf{A} = \begin{pmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \quad (2.2)$$

with  $(u_0, v_0)$  being the principal point,  $\alpha$  and  $\beta$  the scale factors for the  $u, v$  axis and  $\gamma$  parameter describing the skewness between them. Given this model, the approach described by Zhang et al. and also generally used in the literature is:

1. Detect corners or other well known points in every input image.

2. Use the 2D – 3D correspondences to estimate the camera projection matrix ( $3 \times 3$  homography matrix):  $\mathbf{H} = \mathbf{A} (\mathbf{R} \ \mathbf{t})$ . This projection matrix has 8 degrees of freedom. An initial estimate of the projection matrix can be acquired by linear least squares. The estimate is often further refined using non-linear optimization techniques (e.g. Levenberg-Marquardt [110]) minimizing  $\min_{\mathbf{H}} \sum_i |\mathbf{p}_i - \hat{\mathbf{p}}_i|$ , where  $\hat{\mathbf{p}}_i$  is the estimated position.
3. Recovering the intrinsic matrix  $\mathbf{A}$  from the projection matrix  $\mathbf{H}$  has a closed-form solution. Given the intrinsic matrix the extrinsic parameters  $\mathbf{R}$  and  $\mathbf{t}$  can be recovered in closed form [188] as well.
4. Until now the radial ( $k_1, k_2, k_3$ ) and tangential ( $p_1, p_2$ ) distortion coefficients with  $(u_0, v_0)$  as the center were ignored. Based on the literature, for desktop cameras the distortion is dominated by the first two radial components [188, 163]. These can be estimated using linear least-squares solution. More elaborated distortion modeling was found to cause numerical instability [98, 163].

$$s (x' y' 1) = \mathbf{A} (\mathbf{R} \ \mathbf{t}) \mathbf{P} \quad (2.3)$$

$$x'' = (x' - u_0), y'' = (y' - v_0), r = \sqrt{x''^2 + y''^2} \quad (2.4)$$

$$x = x' + k_1 r^2 + k_2 r^4 + k_3 r^6 + 2p_1 x'' y'' + p_2 (r^2 + 2x''^2) \quad (2.5)$$

$$y = y' + k_1 r^2 + k_2 r^4 + k_3 r^6 + p_1 (r^2 + 2y''^2) + 2p_2 x'' y'', \quad (2.6)$$

where  $s$  is an arbitrary scalar and  $x, y$  are the true image coordinates.

5. Refining  $\mathbf{A}$ ,  $\mathbf{R}$ ,  $\mathbf{t}$  and the distortion coefficients through a non-linear optimization (e.g. Levenberg-Marquardt [110]).

In augmented reality applications where the camera stream is directly used for rendering, several assumptions are made usually about the camera model. Most of these assumptions relate to the properties of the rolling-shutter cameras that are used in prevalent majority in the literature and the limitations of real-time rendering engines. The skewness  $\gamma$  between the axis is set to 0, assuming  $90^\circ$  between the  $x$  and  $y$  axis. In reality, rolling shutter camera's  $\gamma$  depends on the movement speed of the camera and sensor frequency. Modeling this behavior would require the prior knowledge of these and is not available during calibration from a static set of images.

During the rendering of virtual objects, the pixels are assumed square and the center of projection is set to the center of the image. To match the square pixel assumption during calibration, a constant focal length is assumed:  $\alpha = \beta = f$  focal length. Depending also on the used rendering software, often the supported camera model might be limited and ignores the camera distortion altogether for maximum performance. With fully calibrated camera model used to recover camera pose and no support for camera distortion from the rendering software, the virtual objects would be misplaced.

A limitation of the described calibration procedure is the assumption of constant parameters. Changing the zoom level of the camera causes a non-linear change in camera parameters and are difficult to model for a continuous range of lens settings. Another difficulty arises with determining the current zoom level in real time. Variable-parameter lenses, hence, are not commonly used in Augmented Reality or Machine Vision [176].

In photogrammetric calibration, the most used calibration objects are chessboard patterns. The disadvantage of these approaches is that the whole chessboard containing the control points for the calibration has to be visible in the images. This is problematic especially at the corners. Oyamada et al. [120] proposed a method to lift this limitation. In order to allow partially occluded control points, they proposed the use of Random Dot Markers [166]. In a synthetic experimental evaluation

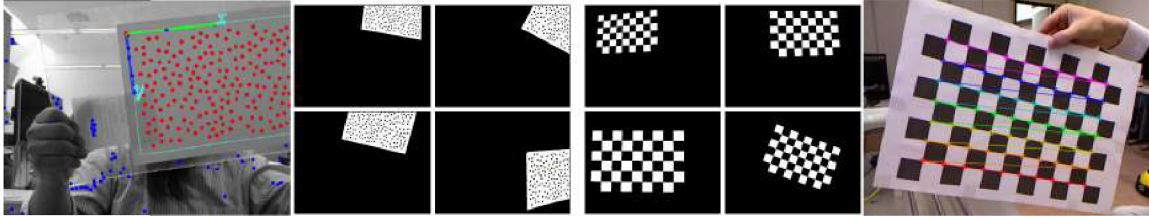


Figure 2.2: Random Dot Markers compared to chessboard based calibration. Random Dot Markers [120] can be partially hidden at corners.

they showed that the correctly detected control points near the corners and edges of the calibration images increased the accuracy and stability of the camera parameter estimation (See Figure 2.2<sup>1</sup>).

## 2.2 Perspective- $n$ -Point Problem

In some computer vision applications, simple 2D homography between consecutive frames might be enough to track the camera movement. In augmented reality, however, most applications rely on acquiring the full camera position and orientation relative to a known origin. In these calculations, the camera’s intrinsic parameters are assumed to be available (see Section 2.1). The problem of determining the camera pose given the correspondences between 2D and 3D points and the intrinsic parameters is known as *Perspective- $n$ -Point*, where  $n$  refers to the number of correspondences. From this section onward, I stick to a notation most often used in the literature for Augmented Reality which is slightly different from the previous section.

Generally,  $PnP$  solving algorithms try to solve the equations given by the pin-hole camera model:

$$\mathbf{p}_i \approx \mathbf{K}(\mathbf{R}, \mathbf{t})\mathbf{m}_i \quad (2.7)$$

for unknowns  $3 \times 3$  orthogonal rotation matrix  $\mathbf{R}$  and translation vector  $\mathbf{t}$  for each correspondence  $(\mathbf{p}_i, \mathbf{m}_i)$ . Points  $\mathbf{p}_i$  are the undistorted 2D projected points expressed as a column vector in homogeneous coordinates and 3D homogeneous column vectors  $\mathbf{m}_i$  of the model points.  $\mathbf{K}$  is the camera intrinsic matrix:

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \quad (2.8)$$

where  $f_x, f_y$  are the focal lengths expressed in the same units as the projected points and  $(c_x, c_y)$  is the principal point. These equations are equivalent up to notation to equations (2.2) and (2.1) with the assumption of perpendicular axes.

Solving  $PnP$  accurately is computationally expensive even with known camera calibration. Quan et al. [130] described an often used solution for fixed  $n = 3$  (combined with RANSAC for  $n > 3$ ). Usually a fourth point is used for disambiguation between possible solutions. This is used with markers like ARTag [41] or ARToolkit [76], which have exactly 4 corner points. For  $n > 3$  stable non-iterative approaches have complexity of  $O(n^5)$  [130] or even  $O(n^8)$  [6]. From iterative approaches, Lu et al. [89] described a very accurate algorithm, though slower than non-iterative algorithms without a good initial pose.

More recently, Lepetit et al. [84] proposed a non-iterative algorithm combined with Gauss-Newton optimization algorithm with  $O(n)$  complexity –  $EPnP$ . They express the 3D points as a

<sup>1</sup>Image from: <http://nicolas.burrus.name/index.php/Research/KinectCalibration>

weighted sum of four virtual control points. Thus, they reduce the camera pose estimation to these four points only. Their solution assumes a simplified pinhole camera discarding all distortion. The undistortion of the projected points takes place before applying this method. They reduce the  $PnP$  problem to computing the eigenvectors of a  $12 \times 12$  matrix and a constant number of quadratic equations. They propose to improve the precision of this closed form solution using Gauss-Newton scheme. They optimize four projection parameters for the virtual control points using the distance relationships between them. According to their testing, this improved the precision of their method to comparable levels to LHM [89] (named after the authors: Lu, Hager, Mjolsness). They also showed that using  $EPnP$  as the initial guess for LHM, they could reduce the number of needed iterations required by LHM.

In an augmented reality setting, the temporal camera pose dependence between successive frames provides a good enough initial guess for iterative methods to reduce the number of needed iterations. Methods like  $EPnP$  are still useful during initialization and when the tracking gets lost.

The rising availability of builtin inertial measurement units in camera racks and smartphones provides opportunities to aid the camera pose estimation. These auxiliary sensors can help create more robust tracking, but also can be used to reduce the camera pose's degrees of freedom for the  $PnP$  problem. Sweeney et al. [154] introduced a robust camera pose estimation from only two 2D-3D correspondences assisted by prior knowledge of the gravity direction. Their method requires only solving a single quadratic equation.

Although there has been active research in all areas concerning augmented reality systems, tracking methods for camera pose estimation have high popularity [189]. In Figure 2.1, this concerns mostly the four bottom sub-systems: Detection & Tracking, Point matching, Homography or  $PnP$  calculations and Camera pose calculations. Section 2.3 summarizes the current state of the art in this area.

## 2.3 Vision-based Tracking Techniques

Vision-based tracking have been a very active area of research in augmented reality. It allows to calculate the camera position with high accuracy compared to other sensor based techniques. They represent closed loop systems, since they can use results from previous steps and correct errors dynamically.

Vision-based tracking methods can be separated into two main classes based on the used information from the image: feature-based and model-based. The feature-based methods try to find a correspondence between 2D image feature points and 3D world frame coordinates. Feature-based methods can be further split into two groups based on the type of the features used for detection: fiducial marker based tracking methods (Section 2.3.1) and natural image feature based tracking (Section 2.3.2).

Model based tracking methods explicitly use the features of tracked objects, which have a 3D model known beforehand. This technique is often combined with methods based on natural features. The texture of objects provides more easily trackable features and is usually more dominant than the shape of the objects. These approaches are discussed in Section 2.3.3.

As mentioned at the beginning of Chapter 2, computer vision methods alone are not sufficiently robust against rapid camera movement. Several hybrid methods have already been discussed above. A more detailed discussion of the state-of-the-art hybrid methods is in Section 2.4.



### 2.3.1 Approaches Based on Fiduciary Markers

Historically and also in recent literature many augmented reality based research is using *fiduciary markers* to reliably establish the camera position within the scene. The first vision-based camera pose tracking techniques achieved real-time camera pose estimation thanks to these fiduciary markers. Such markers are typically two-dimensional planar black-and-white images placed into the scene. When the camera captures a marker in its entirety or a significant portion of it, the homography between the observed projection and the known location within the scene can be accurately computed.

By (fiduciary) markers in this and the following sections a synthetic 2D pattern is meant, which was designed to be easily found and used as a source for 2D to 3D correspondences. Popular designs of fiduciary markers consist of two components: geometrical features which help localize the marker in the processed image and features defining the identity of the marker (for example [41, 43, 76]). That allows for placing several (or many) markers into one scene and their efficient detection. Usage of several markers displaced within the scene is necessary to allow for free movement of the camera within the scene.

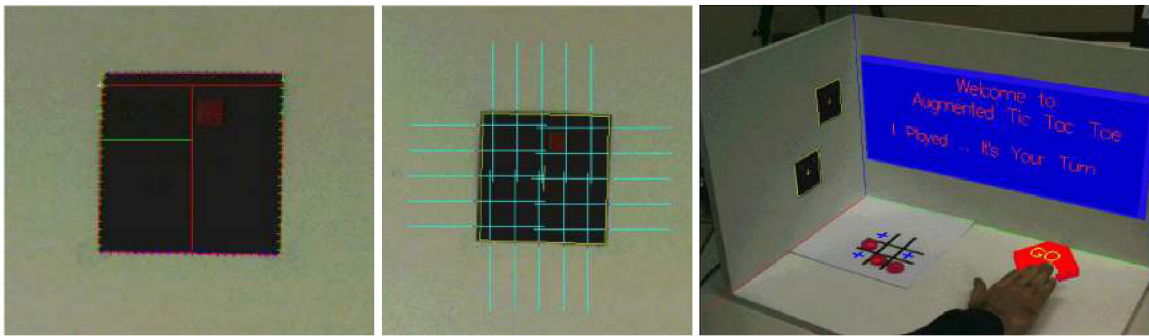


Figure 2.3: Landmark detection and identification by Stricker et al. [150] on the left. Tracking in the center and on the right an example application.

Rekimoto [136] in 1998 has shown an approach to combine pattern recognition and pose calculations. He used square markers with binary encoded information in combination with adaptive thresholding and connected component analysis for marker detection. Stricker et al. [150] also presented an optical tracking system which tracked square markers, but it was able to track linear features of landmarks, too. To achieve real-time performance, they used scanlines and edge pixel connection to get edge lines in the initiation step (Figure 2.3). After the position was successfully detected, they used simple 2D linear extrapolation as predictions for the tracking. To uniquely identify each blob or rectangle in the image, they used a labelling area with binary codes represented by red color overlay.

#### ARToolkit

The ARToolKit library was first presented in 1999 [76]. Even though the detection algorithm has disadvantages and newer, more robust methods were introduced, it is still used in research in augmented reality as a fast and simple solution. The detection algorithm of the markers is based on binarization with adaptive threshold (see Figure 2.4). Image regions after the thresholding, with contours that could be fitted by four line segments, are considered candidate positions for markers. The identity of the markers is determined by first rectifying and then matching the contents of the regions to a set of templates. The corners of the marker are then matched to their known

3D positions. Based on these correspondences, the algorithm estimates and iteratively refines the camera pose to achieve higher precision.

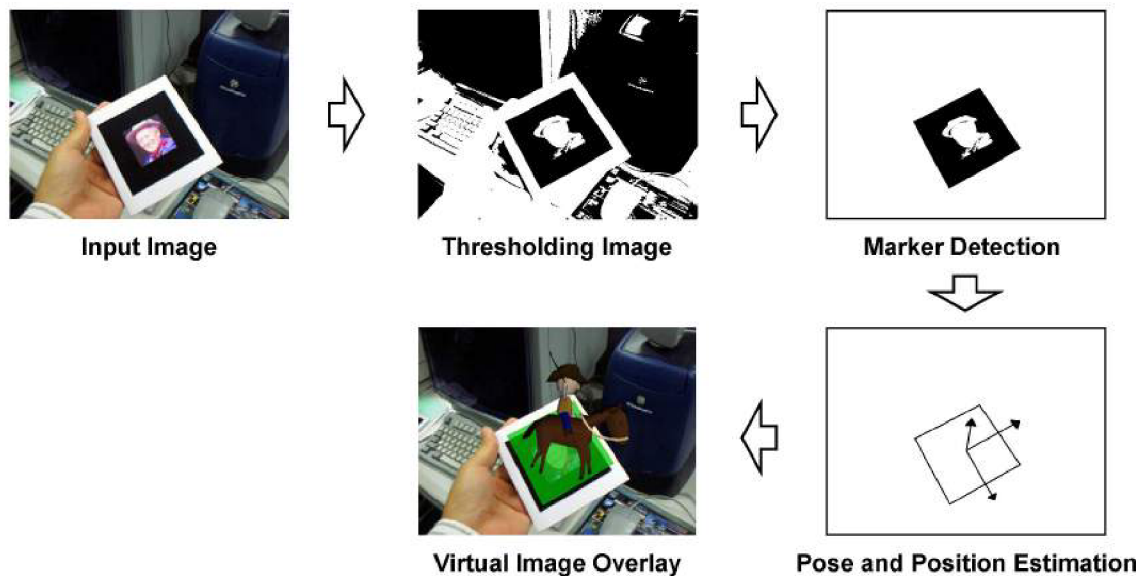


Figure 2.4: ARToolkit [77] tracking process.

The disadvantages of the described detection algorithm are the relatively expensive template matching and adaptive thresholding. ARToolkit detection algorithm lacks robustness against occlusion of the borders. The use of template matching and computing correlation to verify and identify markers might cause high false positive rates or mismatch between markers with growing library size of templates or in less ideal shots in some situation (motion blur, lighting, depth of field, etc.). The number of marker templates with ARToolkit is also limited by computational performance. They store twelve prototypes for each marker in an effort to cover all four rotations and different relative lighting conditions.

### ARTag

ARTag introduced by Fiala [41] has tried to solve these problems (Figure 2.6). They combined *Data matrix* coding for marker identification with the rectangular thick border shaped markers used in ARToolKit. In the detection algorithm, they replaced the adaptive thresholding of image regions with thresholding of extracted edges. These changes improve detection performance and require lower computational complexity.

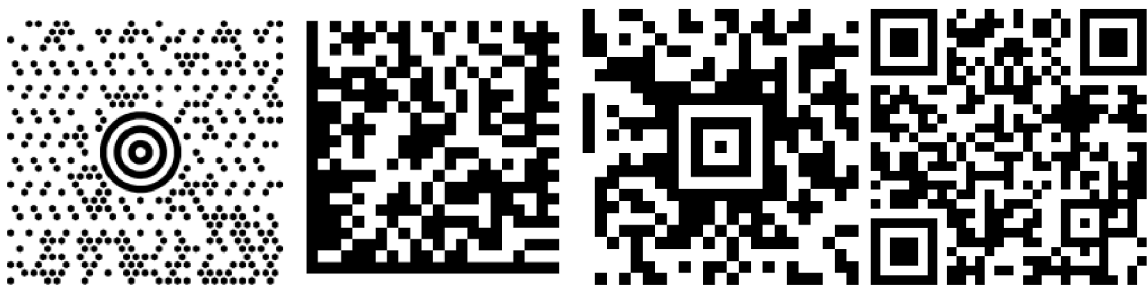


Figure 2.5: From left to right MaxiCode, Data Matrix, Aztec Code and QR-code.

The most notable change introduced by ARTag is replacing the template image with digitally encoded information. There have been 2D barcodes already available to encode digital information, like MaxiCode used by the US Postal service, Data Matrix, Aztec code [1], QR-Code [3] to mention a few (see Figure 2.5). These barcodes have been designed to be easily localized and to maximize data density and error-correction capabilities. They lack reliable reference points to be used for camera pose estimation compared with ARToolkit markers with the 4 distinctive corners. ARTag tried to combine the two approaches by replacing the image template in ARToolkit markers with encoded binary information.

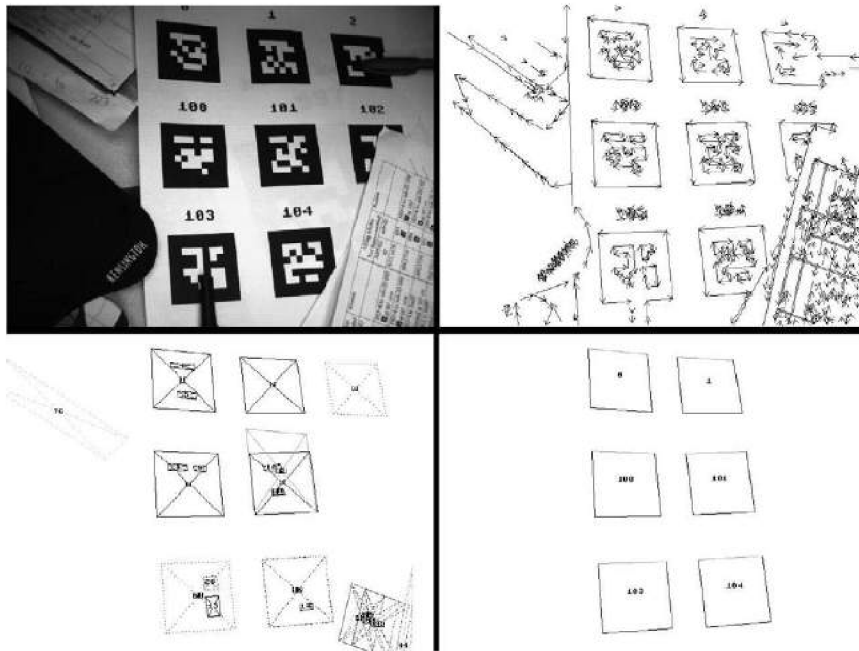


Figure 2.6: ARTag [41] marker detection algorithm.

ARTag contains a 2D  $6 \times 6$  digital symbol array representing the encoded marker ID. They use 10 bits to store the ID and the extra 26 bits are used to provide redundancy and to ensure that the markers are uniquely identifiable in all 4 rotations. The ID encoding algorithm uses an XOR operation first to shuffle the bits, followed by Cyclical Redundancy Check (CRC) and Forward Error Correction (FEC) algorithms. The error checking and redundancy bits are also beneficial to rule out any false positives.

ARTag also improves upon the quad detection from ARToolkit. Instead of binary image segmentation, the algorithm uses edge pixels over a predefined threshold linked into segments. These segments are further combined into quad segments. Using the segments, the algorithm is able to detect the marker outlines even if a corner is missing or the edge segments are broken by occlusion using heuristics of segments that almost meet.

In their experiments, Fiala et al. [41] show vastly superior performance when compared to ARToolkit. ARTag still suffers from several design flaws inherited from ARToolkit. The individual markers are small and provide only 4 points relatively close together in projected image space for camera pose estimation. The camera pose estimation precision hence still suffers causing unpleasant jitter in the AR interface.

In a follow-up work, Fiala [42] combined several markers on a single plane of known relative position and rotation to improve the reliability of the camera pose detection (see Figure 2.7). He used this approach to provide a “magic mirror” system with acceptable accuracy and delay. One

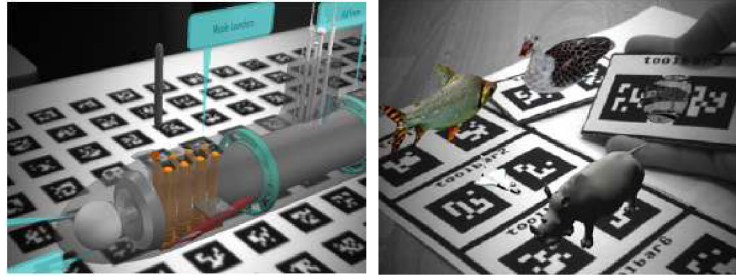


Figure 2.7: Several ARTag markers combined into a single plane [42].

disadvantage of this approach is that individual markers are still detected and decoded separately, wasting computational time. The second disadvantage is that the relative position and rotations between markers contributing to a single reference frame have to be annotated manually. Only after each marker was detected, their corners are extracted and combined together based on the annotations to improve camera pose estimation precision.

### Composite Markers

When relying solely on tracking based on fiduciary markers, any frame of the camera must contain at least one of the markers and the marker must be large enough in order to compute precise camera location. These requirements are contradictory. Installing markers everywhere is unreasonable, since they become too conspicuous and difficult to calibrate and set-up. Multiple marker solutions also still do not deal well with varying observing distance even when using differently sized markers. Even though a marker is always visible in the scene, a close-up on a large marker or a small marker observed from long distance makes marker identification and subsequently camera pose estimation difficult.

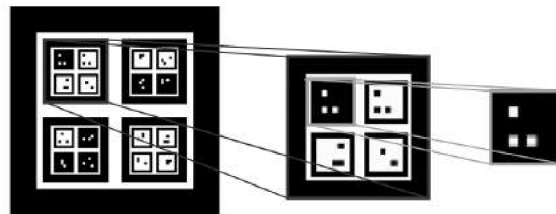


Figure 2.8: An example of Nested Markers [159] and their sub-markers.

Nested Markers [159] try to solve the contradictory requirements of small enough marker size to fit in the frame and large enough to realize accurate geometric registration by nesting a number smaller markers into a larger one. Their marker design is depicted in Figure 2.8. Their detection algorithm is still based on ARToolkit algorithm. The visual representation of the markers at each level are used as templates for identification. This, similarly to ARToolkit, leads to limitations on the size of the library of marker IDs due to false-positive rate and high computational complexity while identifying the marker. In their experiments, they consider a relatively low distance range with a nested marker with 3 layers (4cm to 80cm). This approach also does not consider large lateral movements.

One of the latest improved techniques based on individual square based markers was proposed by Herout et al. [59], who introduced Fractal Marker Fields (Figure 2.9). They provide the ultimate solution to the contradictory requirements faced by pure marker-based approaches. These marker

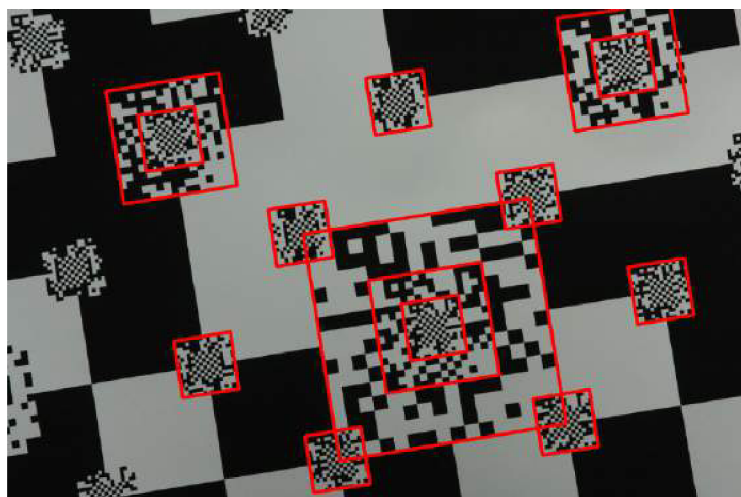


Figure 2.9: Fractal Marker Fields [59] with detected sub-markers.

fields provide guaranteed density of visible markers in every scale, solving the main problem of limited distance range useable for detection.

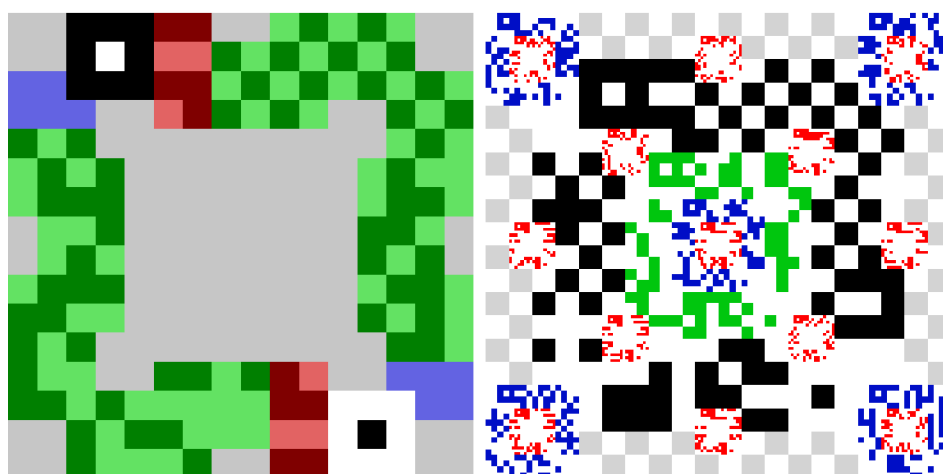


Figure 2.10: Fractal Marker Fields [59] with detected sub-markers.

Their marker design is reminiscent of the QR code and ARTag (see Figure 2.10 left) with two black and white synchronization and validation patterns combined with encoded marker ID forming a border. In Figure 2.10 the remaining blank space in the middle and corners (gray color) is used to place further markers with different scales. This design is inspired by fractal designs – hence the name. They define three main and one additional rules, how the actual „marker field“ is constructed. They state that theoretically a complete marker field containing 42 different marker sizes – levels, if the smallest markers were as small as 3 cm, the whole field would be 17 million km wide. Practically so many levels would decrease the contrast at larger scales, which could lead to loss of detection performance.

Their proof-of-concept implementation of the detection algorithm using parallel coordinates is not included in the paper. I provide here a short description of the detection algorithm, as it served as an inspiration for one of the major contributions in this work – the Uniform Marker Fields detection algorithm. Their detection algorithm was based on an algorithm for detection of chessboard grids

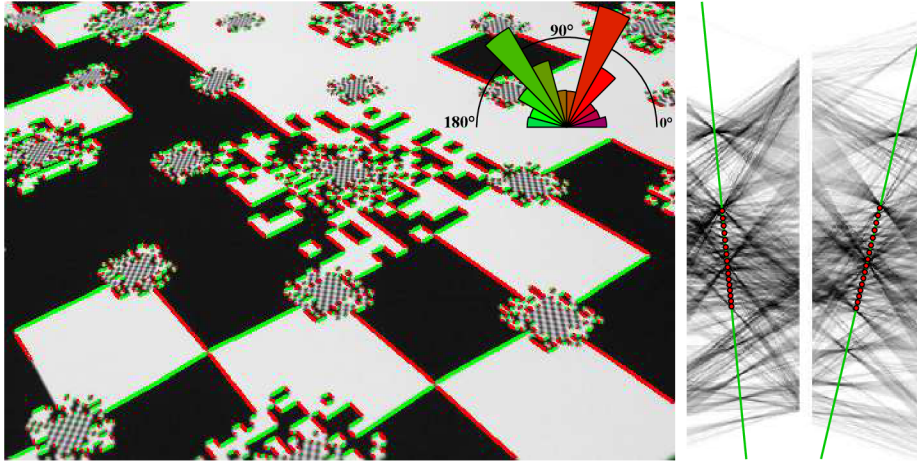


Figure 2.11: Fractal Marker Fields [59] detection algorithm using parallel coordinates. Left: Edges separated into bins based on direction. Right: the highlighted edges mapped to TS space.

and matrix codes using parallel coordinates [36]. The algorithm can be separated into four main steps:

1. **Edge extraction.** Thresholded edges extracted by a computationally cheap edge detector are separated into two dominant edge orientations. These orientations are roughly  $90^\circ$  apart. Only edges with gradient direction close to these orientations are processed further.
2. **Accumulation to the Hough Space.** For each group of edges with similar gradient directions, a small part of the complete Hough space is accumulated. The Hough space in their work is the TS space using parallel coordinates (Figure 2.11). The advantage of this space is that the vanishing points are represented as lines (green lines in Figure 2.11). The maxima in the Hough space, defining dominant lines in the image with a common vanishing point, coincide with the line representing the vanishing point.
3. **Finding Groups of Originally Parallel Lines** Another interesting property of the TS space is that after stretching the vanishing line using the information about the two dominant vanishing points, equally spaced lines in the real world are represented by equally spaced maxima in the TS space. For both orientations, equally spaced clusters of maxima are found on the stretched line to recover their frequency.
4. **Extraction and decoding of the Marker Bitmap.** Using the detected frequency of the detected maxima clusters, a parametric solution can be found for the module centers at each scale. At each scale, a bitmap is extracted and the synchronization patterns (Figure 2.10) are used to localize the actual markers. The localized markers are then decoded and verified for errors. Based on the recovered ID, the marker's real 3D position is known.

Unfortunately, FMF was not researched further to create a full augmented reality system. From the marker design it is not well defined what information would be used to recover a full 3D pose. I hypothesize three different approaches. Due to the visual resemblance between FMF and the Uniform Marker Fields proposed in this work, these hypotheses apply to Uniform Marker Fields as well.

**Global solution** The vanishing points for two axes were found during step 2. of the detection algorithm. This information is sufficient to recover the camera rotation. A single known

3D-2D correspondence is sufficient to detect the camera translation. This approach would provide a precise camera pose under the assumption that the camera was calibrated precisely and either the edge pixels or the whole image was undistorted during the vanishing point calculations. Given this assumption and precise vanishing point estimation, the computed 3D pose should be very precise. The computational complexity depends largely on the camera's distortion. With assuming insignificant distortion, this approach provides a computationally very efficient closed-form solution. The camera pose precision, however, would be highly sensitive to precise camera calibration and the precision of the vanishing point detection.

**Marker based solution** Given the detected markers, a classical marker based camera pose estimation is possible. For each successfully detected marker, the marker's outer edge segments could be refined. The intersections of lines defined by these edge segments should define precisely the marker corners. These 4 corner points can be used to recover the camera's 3D pose using a conventional closed-form P3P algorithm [130]. If multiple markers were detected successfully, the camera pose can be iteratively refined from the initial guess. Using this approach, only the corner points would have to be undistorted. This approach would theoretically provide a good balance between camera pose precision and computational complexity.

**Corner based solution** Corner points between modules of successfully detected markers can be extracted with sub-pixel precision. The real 3D position of these points can be recovered from the marker's ID. The 2D-3D correspondences could be used to recover a very precise camera pose using any  $PnP$  algorithm (Section 2.2). This approach would provide the most precise and robust camera pose estimation at the expense of high computational complexity.

Fractal Marker Fields would be an ideal solution in large-scale situations, where markers are acceptable. Practical applications of the allowed freedom in scale is limited. In most real-world augmented reality applications: human-computer interaction, 3D visualization, medical training, etc. – 2-3 scale levels at most are sufficient. The biggest disadvantage of Fractal Marker Fields is the dependence on computationally complex detection algorithms. The Hough transformation and maxima detection is theoretically the most demanding. Alternative marker detection algorithms are difficult. The synchronization patterns (Figure 2.10) are not distinct enough for quick detection as for the QR code. Detecting the frequency at each scale without the global solution of Hough transform could also prove difficult. The undeniable advantages of the checkerboard structure and the effective use of vanishing points and line parametrization served as great inspiration for this thesis.

### Random Dot Markers

One disadvantage of traditional square markers like ARToolKit [76] and ARTag [41] is the black frame. ARTag was able to reconstruct the marker frames if they were partially occluded, but neither approach is able to handle occlusion of corner points or a completely covered edge.

Uchiyama et al. [166] used randomly scattered dots as fiducial markers (Random Dot Markers – Figure 2.12 left). Compared to traditional markers with square patterns, Random Dot Markers require slightly larger area, so that the camera could recognize the individual points for detection. On the other hand, random dot markers are more robust against occlusion.

The detection algorithm assumes black points on white background. The algorithm first binarizes the image using a threshold to extract the black dots. The centroid of the extracted dot segments are used as keypoints. Each keypoint is then matched with the database and votes for the individual

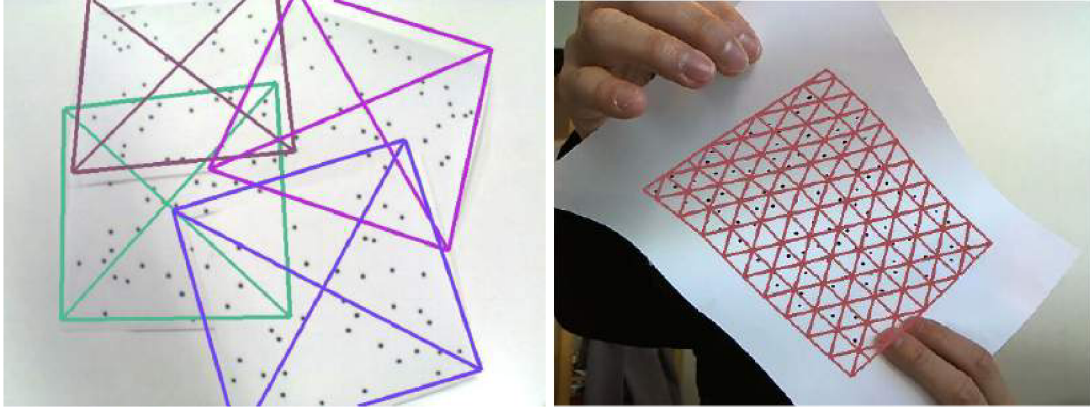


Figure 2.12: Random Dot Markers [166] demonstrating robustness against occlusion. **Right:** Deformable Random Dot Markers [165] with detected mesh.

marker ID. In their work, they reference Locally Likely Arrangement Hashing (LLAH) [106] as the matching algorithm.

In the original work describing LLAH for document page matching, Nakai et al. [106] use two different geometric invariants for the keypoint descriptions: cross-ratio and affine invariant. The cross-ratio uses 5 neighbor co-planar points  $A, B, C, D, E$ :

$$\frac{P(A, B, C)P(A, D, E)}{P(A, B, D)P(A, C, E)}, \quad (2.9)$$

where  $P(A, B, C)$  is the area of the triangle with vertices  $A, B$  and  $C$ . The affine invariant requires only 4 points as the ratio between bordering triangles:

$$\frac{P(A, C, D)}{P(A, B, C)}. \quad (2.10)$$

The former is invariant to perspective transformation, while the latter is invariant only to affine transformation. Unexpectedly, Random Dot Markers uses the affine invariant, which would theoretically give worse results under perspective transformation. Since perspective view of the markers is very common in augmented reality applications, this choice should theoretically hinder the detection performance.

The descriptors for a keypoint are computed as the affine invariants from 4 points for all combinations  $\binom{n}{4}$ ,  $n > 4$  of  $n$  neighbor points. These descriptors are then hashed for fast retrieval. Overall, the matching algorithm has a very efficient  $O(N)$  computational complexity, where  $N$  is the number of keypoints in the image. The camera pose estimation uses the correspondences between 2D positions of successfully identified keypoints and their known 3D positions from the database for the given marker.

The advantage of Random Dot Markers is that it is not constrained by a square area and it has excellent robustness against occlusion. Theoretically any shape can be used for the marker. The disadvantages of their approach lie in the memory-intensive keypoint extraction, questionable choice of descriptors, and the sensitivity of the detection algorithm to the chosen dot size. Large dot size slows down segmentation and makes it difficult to extract the dot centroid precisely under various transformations. Small dot size limits the distance from where the marker is still detectable. Small dots also make detection difficult under motion blur, where the dots simply disappear unlike corner points of conventional markers like ARTag and ARToolKit.



Since the geometric descriptor uses only local arrangements of points, the Random Dot Markers could also be applied to slightly curved surfaces (Figure 2.12 [165]). This approach allows planar mapping of textures. For example it can be used to replace the dot marker on a T-shirt with arbitrary 2D image. Full 3D camera pose estimation from a deformed surface would be unstable.

### Alternative Approaches

Before the introduction of ARTag by Fiala [41] in 2005, Zhang et al. [185] compared leading approaches of square marker based detection algorithms. Marker based approaches were considered almost fully researched and only a few publications were created on this topic (except for the previously mentioned). Marker based approaches are still being utilized in the development of prototypes and actual systems either as the primary source for camera calibration and tracking or as reference frame during the tracking initialization.

Apart from square bordered markers, some researchers also explored ring shaped fiducial markers [27], circular shaped marker clusters with various parameters [171]. An example of more recent non-square markers apart from the aforementioned Random Dot Markers is reacTIVision [74].

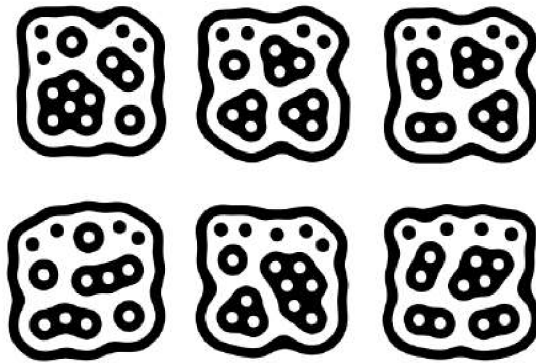


Figure 2.13: Several reacTIVision [74] markers.

ReacTIVision uses “amoeba” fiducials (see Figure 2.13). It consists of black and white segments forming a tree by inclusion. The detection algorithm is based on segmenting out each region and finding the sub-pixel precise positions for the node centers in the image. They used their framework to create “reacTable” – a multi-user table-based instrument for music generation.

The amount of research on fiducial marker based tracking has been on decline in recent years. This suggests that the research in this area achieved high maturity. Marker based approaches are slowly replaced by natural feature based tracking. However, where simplicity, precision and computational efficiency is critical, marker detection based methods are still superior and widely used method.

### 2.3.2 Camera Tracking Based on Feature Points

Marker based tracking methods have the disadvantage of the need to use special equipment or printed patterns. They could also be seen in some cases as unnecessary visually disturbing artifacts. Instead of markers, naturally occurring image features, such as points, lines, edges, and textures could be used by an augmented reality system.

Park et al. [122] integrated natural feature tracking with fiducial tracking to increase the range and robustness of vision-based augmented reality tracking. They still used fiducial markers for

initial calibration. Once the natural features' position was found they used tracking of these features to extend range and stabilize the tracked pose against occlusion and noise.

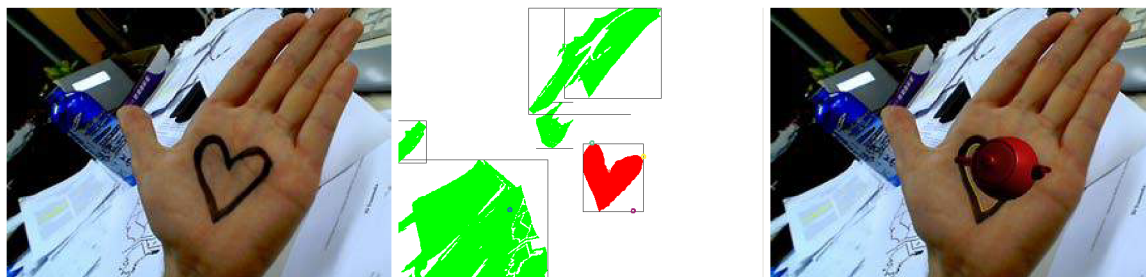


Figure 2.14: Demonstration of Robust Planar Target Tracking and Pose Estimation from a Single Concavity [34]. Left image is the input image, the middle shows the detected and classified (color-coded) MSER regions and the right image shows the augmentation.

Natural feature point base tracking has been a very active area in vision tracking research [189]. In order to get full 3D camera pose from the tracked features, these techniques often build models (see Section 2.3.3). Augmented reality systems also need highly stable feature detection methods with high repeatability rate and efficient feature matching techniques for real-time speeds. A good example of using feature detectors for tracking is the work of Donoser et al. [34]. They were able to track camera pose using Maximally Stable Extremal Regions and with one contour with at least one concavity (see Figure 2.14).

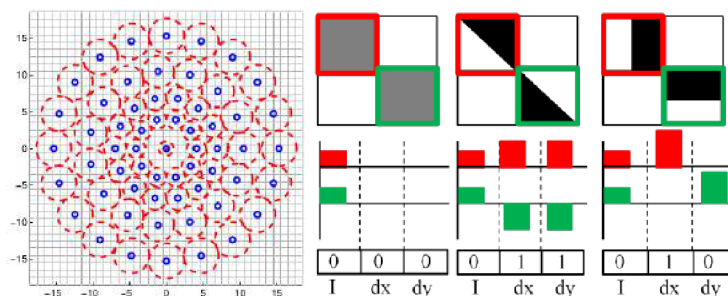


Figure 2.15: **Left:** BRISK [85] sampling pattern used in simple brightness comparison step between pairs of points (blue) in the image smoothed with given standard deviation (red circle) for the gauss kernel. **Right:** Examples of Local Difference Binary [180] descriptors for the three examples at top and the binary descriptor at the bottom.

SIFT [88] and SURF [15] have been historically the two most used and well performing feature descriptors. However, both SIFT and SURF require relatively high computational resources. For fast keypoint detection FAST [141] was proposed. Leutenegger et al. was able to achieve real-time results with FAST in combination with BRISK [85]. However, they still needed around 30 ms to match 1000 keypoints from two images. Recently Yang et al. [180] proposed a highly efficient, robust and efficient binary descriptor, called *Local Difference Binary* (LDB) with augmented reality applications for mobile platforms in mind. They claim better performance and almost twice the speed in matching when compared to BRISK. Although there are many other publications concerning fast keypoint detection and matching, they are related to the topic of this work only indirectly. A complete survey of the state-of-the-art in this area was published by Marchand et al. [95].

### 2.3.3 Model Based Tracking Methods

The latest trend in computer vision tracking techniques are model-based tracking methods. Simple feature point based methods without full 3D correspondences would make sense only for planar objects. Model based tracking methods use a 3D model of tracked features consisting of feature points, edges or other distinguishable forms.

One of the first 3D trackers was RAPID introduced by Harris et al. [54]. Their key idea was to track points that lie on high contrast edges in the images. The 3D motion then can be recovered from 2D displacement of these points. An early research on 3D model based tracking approach for augmented reality was presented by Comport et al. [30]. Some further research already existed in the related computer vision and robotics literature. Most of this research, though, was focused on motion tracking and 3D object recognition and tracking, not primarily on augmented reality.

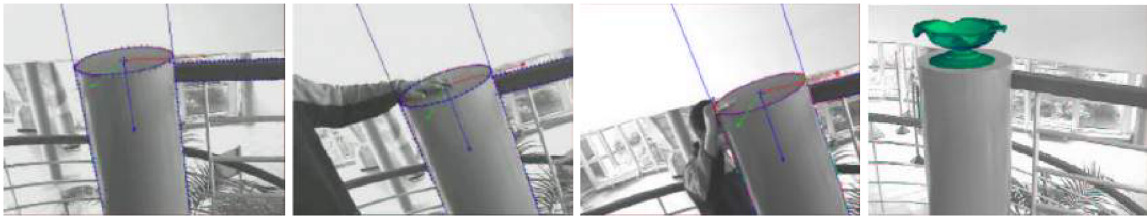


Figure 2.16: Model based tracking by Comport et al. [30] using a circle, a cylinder and two lines.

Comport et al. [30] incorporated research also from the robotics field. Specifically, they used tracking by means of visual servoing. In visual servoing the aim is to move the virtual camera to a position defined by the 2D projection image. This can be achieved by minimizing the error between the desired state's and current state's image features. They showed that if the position difference between consecutive frames is small, using the determined 3D camera pose from the previous frame, the visual servoing approach will converge to the new position. They defined several error functions for primitives like lines, ellipses, and cylinders (see Figure 2.16). They tracked these primitives using local moving edge trackers, in order to achieve real-time frame rates. To improve the robustness against outliers and occlusion, they proposed a new robust control law integrating an M-estimator.

Most modern model based tracking methods, unlike the previous solution, build their own models based on points, edges, or lines. There are two main families of approaches, depending on how the image features are being used. The first family tries to match projections of target objects based on lines and edge positions, as the algorithms described above. The second set of approaches rely on local information in the image region as outlined in Section 2.3.2.

Model based approaches that rely only on geometric properties of the objects are not scalable enough for larger scenes. In an outside area they fail to register finer geometry, like the windows on buildings. Reitmayr and Drummond [134] presented a hybrid approach, using the texture edge properties, if required, combined with the classical 3D geometry model based tracking using distinctive edges. This method is called tracking-by-synthesis. More recently, Simon [147] proposed an extension of this method to use interest points instead of simple image properties (edges, corners,...) during textured model comparison (Figure 2.17 left). They also compared the performance of Harris corner detector, SURF and FAST keypoint detectors for this application.

Most of the 3D reconstruction methods like monocular SLAM, DTAM [107], PTAM [79] (Figure 2.17 right), etc., could be also classified under model based tracking methods. Even though these methods can be extended to be used in augmented reality system, they require large amount of memory and computational power. As a consequence, it is unrealistic for them to work on

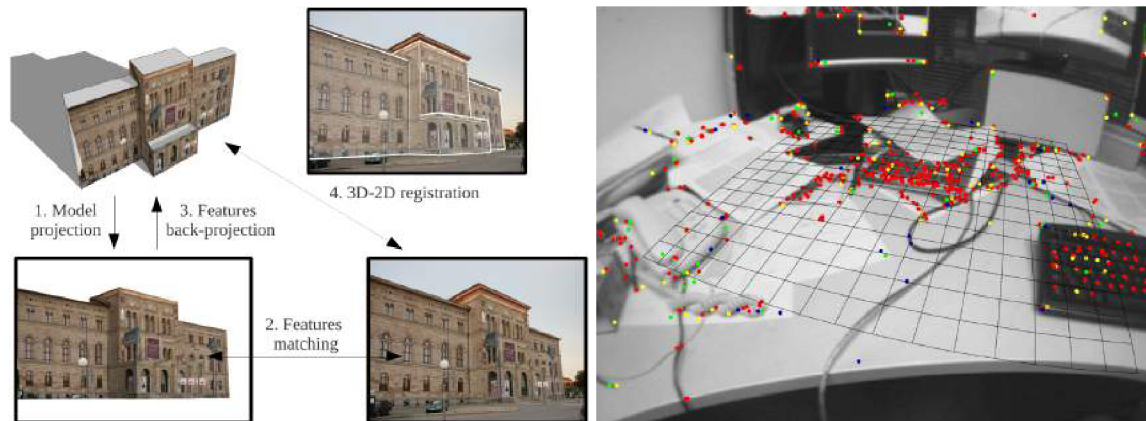


Figure 2.17: **Left:** One iteration of model based tracking combined with feature matching on textures by Simon [147]. **Right:** Parallel tracking and mapping [79] (PTAM) demonstration. The successful point observation and the maps dominant plane is shown in the image.

contemporary mobile devices and used for consumer oriented augmented reality applications. In the future, with the expansion of powerful smart phones and further research, these approaches could become mainstream. In a limited form, some of the functionality is already available in Vuforia developed by Qualcomm<sup>2</sup>. These methods only marginally relate to the content of this work. A more complete survey can be found in [95].

## 2.4 Hybrid Tracking Methods

Vision based tracking methods for augmented reality work with high precision and robustness in confined areas with complex geometry and textures. For some larger environments and situations when the camera view changes dramatically, vision based tracking methods are not sufficient. Using other type of sensors, like GPS or inertial sensors, works well in these situations. On the other hand, however, they do not provide acceptable precision due to noise and error accumulation. This led to the development of hybrid tracking methods.

Some of the research was already discussed in previous sections. I mentioned earlier the work of Park et al. [122], who combined fiducial markers and feature tracking. I also used the work Oskiper et al. [117] as an example of the combination of visual and inertial tracking (Figure 2.18). Notable newer research was done for example by Klein and Drummond [78] who combined a model-based approach with gyroscopes for rapid camera motion. Satoh et al. [142] described head tracking using visual tracking and a gyroscope to reduce the number of parameters to be estimated.

With the dawn of personal high computational performance, small form-factor devices equipped with wide variety of precise inertial and localization sensors, this research area has great potential in the near future. Until now most of these devices were focused on virtual reality (Oculus Rift<sup>3</sup>, HTC Vive<sup>4</sup>, Playstation VR<sup>5</sup>), but researchers already started exploring the possibilities of using these devices for augmented reality by using semi-transparent glasses (Microsoft HoloLens<sup>6</sup>) or using see-through cameras [116].

<sup>2</sup><http://www.qualcomm.com/solutions/augmented-reality>

<sup>3</sup><https://www.oculus.com/en-us/rift/>

<sup>4</sup><https://www.htcvive.com/us/>

<sup>5</sup><https://www.playstation.com/en-us/explore/playstation-vr/>

<sup>6</sup><https://www.microsoft.com/microsoft-hololens/en-us>

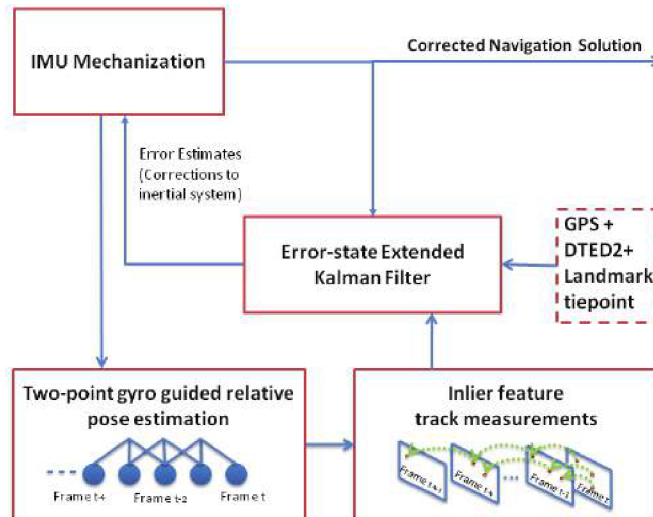


Figure 2.18: System block diagram by Oskiper et al. [117] combining multiple sensors using Error-state Extended Kalman Filter.

## 2.5 Augmented Reality Applications

In the state of the art describing mostly new and improved methods of camera pose tracking, scene modeling and visually correct rendering, there are many applications envisioned for Augmented Reality. The fields of these applications also varies widely, ranging from medical training to children book coloring. Most of the research, though, is focused on individual pieces required to create a full Augmented Reality experience. This leads to the fact that the overwhelming majority of the showcased applications are in the form of demos or eye-catching presentations. Industry-ready use cases of augmented reality are almost completely non-existent. In the state of the art also comprehensive user evaluation of the proposed systems is missing. The enabling technologies and the commodity of high-performance smartphones represent a landmark, which could boost this research area in the near future. This thesis also focuses on these realistic use cases and presents several use cases of AR including a cheap match-moving solution and efficient inter-device content acquisition.

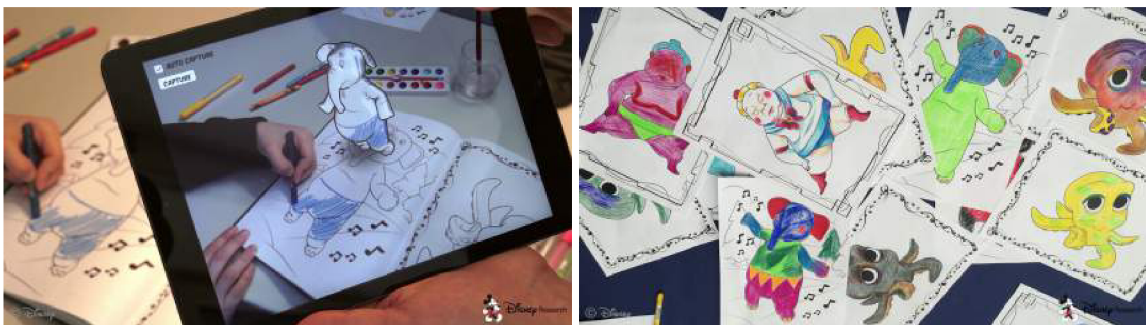


Figure 2.19: Interactive coloring book with live animated characters textured based on the drawing. The templates for augmentation are on the right.

This sections describes some of the latest presented applications in the state of the art that were developed into a fully functional prototype, ready for user testing. All of the described prototypes

rely heavily on moving camera pose estimation and use a hand-held or head-mounted display. I also attempted to include applications from various research areas: training, entertainment, remote interactions or productivity tools.

Recently, Magnenat et al. [94] from Disney Research showcased an application for an interactive coloring book (see Figure 2.19). Their detection algorithm uses the contour binary image. To extract this binary image from the camera frame, they use luminance channel processed by adaptive threshold and smoothed with Gaussian smoothing with standard deviation  $\sigma = 1$  to remove the staircase effect caused by binarization. To detect the template and acquire the 2D-3D correspondences for the camera pose, they use BRISK feature point detector and descriptor. The disadvantage of using feature point descriptors on binary images is the low discriminatory power between feature points. In their work, they had to use complex heuristics to filter out outliers and in some cases they added complex visual patterns around the drawing to achieve reliable camera pose estimation and tracking. For frame-to-frame tracking, they used Lukas-Kanade tracker [183] to achieve real-time frame-rates on tablet devices.

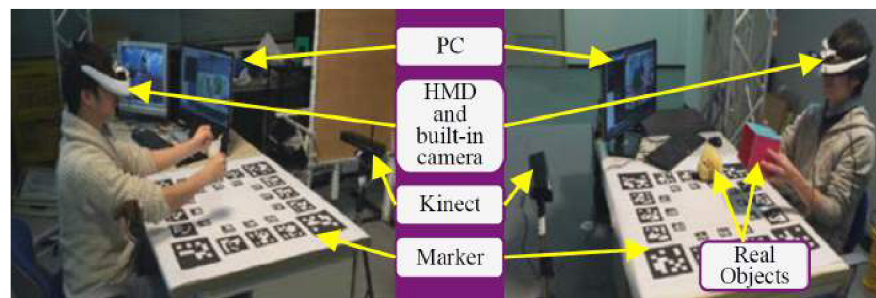


Figure 2.20: Remote Mixed Reality System Supporting Interactions with Virtualized Objects [179].

A recent work from Yang et al. [179] is a good example how markers are used as the base for camera pose estimation even in contemporary AR applications. They used marker based tracking combined with kinect sensor and video see-through head mounted display to create a collaborative desk. They proposed a method to share real objects from one side by virtualizing them on the other side. The shared virtual object could then be controlled by the second participant.

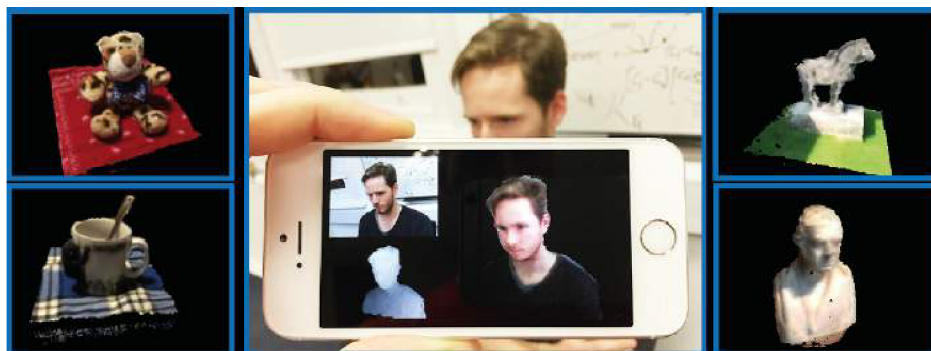


Figure 2.21: MobileFusion: Real-time Volumetric Surface Reconstruction and Dense Tracking On Mobile Phones [114].

An outstanding demonstration of the growing computational power of smart-phones and gradually maturing state of dense tracking algorithms is the work of Ondruska et al. [114]. They demonstrated an application supporting full volumetric surface reconstruction and dense tracking in real time on mobile phones (see Figure 2.21). During their system’s initialization they rely on

feature point tracking-based camera pose estimation to select a 3D cube region of interest (ROI) to be reconstructed. After initialization, they rely completely on dense tracking and reconstruction. The continuously refined volumetric representation of the region of interest is used to track the camera using a dense feature-free monocular approach. In order to achieve real-time frame-rates, they distributed the computational work between the phone's CPU and GPU. They used the GPU for stereo depth computation, volumetric model update and raycasting for the camera pose tracking. On the CPU, they utilized SIMD (NEON) instructions to speed up processing during the camera pose optimization step. The most obvious limitation of their method as a general camera pose tracking is that the ROI needs to be visible in every frame. For the chosen application, this is expected and even required for the model reconstruction. With further research, a more general volumetric representation and with the growing computational power of smart-phones, this research direction has a great potential.

At the International Symposium on Augmented Reality 2015, Hwang et al. [65] showcased an augmented reality system for museums. They created several prototypes using the latest technologies: Oculus Rift, Google Glass and Nexus 5 smartphones. They found Oculus rift problematic for mobile usage and stated the Google Glass was not suitable for AR due to narrow field of view. Their final solution used Nexus 5 in combination with Qualcomm Vuforia SDK and Unity 3D for planar targets and a modified version of PTAM [79] ported to Android. The PTAM algorithm was already mentioned in Section 2.3.3. Their work did not include a description of the necessary changes and optimizations to achieve real-time frame-rates on smartphones.

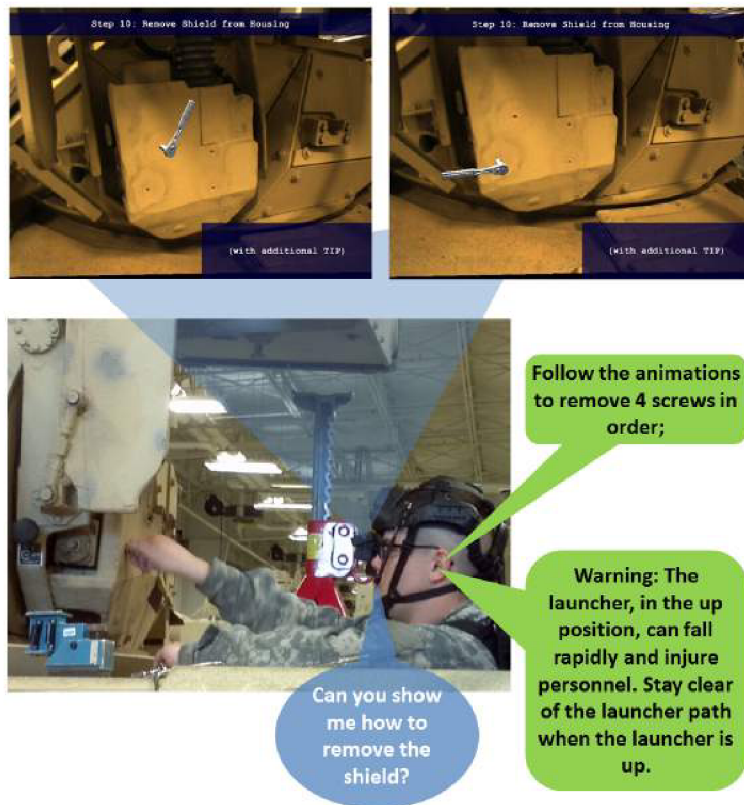


Figure 2.22: A real-time AR mentoring system with virtual assistant [190].

A frequent AR application in the state of the art is mentoring or training. Recently, Zhu et al. [190] demonstrated a wearable real-time AR mentoring system to assist in complicated maintenance and

repair tasks. Their system combined optical-see-through display with virtual voice assistant (see Figure 2.22). For head-mounted camera pose estimation they used a hybrid approach with a high-latency visual landmark matching and feature tracking modules, and a low-latency IMU prediction module. They fused the results from these modules using Extended Kalman Filter to achieve high-precision global camera pose estimation. During the initial live training test their system helped perform an advanced 33-step maintenance task.



Figure 2.23: Augmenting virtual object behind a real surface. Left side image pair shows the pig inside the box without depth hints and the right image pair shows improved depth perception with „stereoscopic pseudo-transparency“ [118].

Recently, Otsuki et al. [118] presented an AR story-telling application focusing on visualizing virtual objects behind or inside real objects (see Figure 2.23). They used an Oculus Rift (HMD) to create a see-through augmented reality display. For the the camera tracking – in this case also head-tracking – they used ARToolKit.

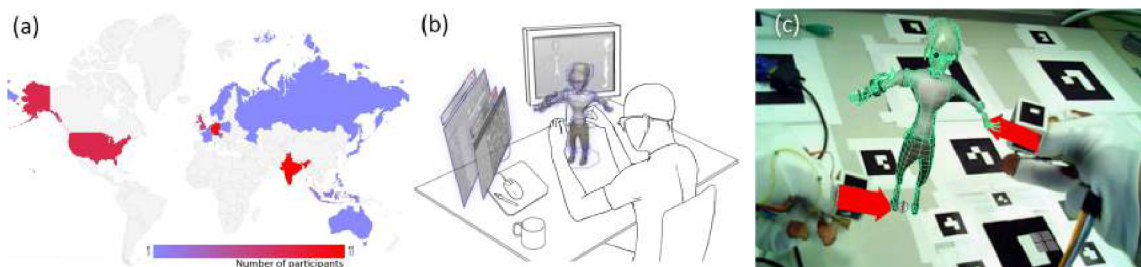


Figure 2.24: A user interface design and prototype for professional 3D media production AR system [81]. (a): the distribution of participants in their survey during the design process; (b) the envisioned AR system; (c): the implemented prototype.

One of the goals set forth in this thesis was to showcase the possibility to create real-time functional AR systems with practical use even in the real world and media production. A similar notion also lies behind the recent work from Krichenbauer et al. [81]. They explored the possibilities to create an immersive 3D UI for 3D computer graphics content creation. In order to form an idea about the requirements and expectations of such system, they conducted a survey with participants from all over the world (see Figure 2.24). For robust 6DOF camera pose estimation and hand-tool localization, they used fiducials in their prototype implementation. These examples further accentuate the fact that fiduciary markers are still the go-to solution for prototype systems, when fast and robust solution is required, and that fiduciary markers are still a relevant research area.



## Chapter 3

# Uniform Marker Fields

This chapter presents the technological core of my PhD research. First, I explain the newly proposed concept of marker fields and its particular implementation Uniform Marker Fields. I describe an algorithm for synthesis of these markers, which is based on a genetic algorithm. I describe a highly efficient detection algorithm for the detection of the Uniform Marker Fields. In later chapters the description of the basics is followed by several use-cases and the description of changes required by the specific circumstances. Lastly, I describe several additional use cases and a fast method for finding regular patterns (like Uniform Marker Fields, and QR codes) in the image for multiple scales and sizes. My main contribution in the following description are mostly the marker field generation, detection algorithms and experimental evaluation.

I have succeeded in contributing at conferences and a journal in the augmented reality field in cooperation with others. Some of the work resulted from collaboration with others: Adam Herout and Michal Zachariáš in relation to marker design and generation and Jiří Havel in relation to de Bruijn tori theory and line parametrization. In the following text I will use „we“, whenever the results described here were a collaborative effort between me and others. The description below is based on our publications.

### 3.1 Orientable Window Arrays as Marker Fields

Perfect maps are 2D arrays in which every possible rectangular subarray of a given size occurs exactly once. The perfect map can be either periodic or aperiodic. An *aperiodic*  $(m, n)$ -window array [24] is an  $k$ -ary 2D array of size  $h \times w$

$$A = (a_{i,j} \in \{0, \dots, k-1\}; 0 \leq i < h; 0 \leq j < w), \quad (3.1)$$

in which each subarray  $A_{r,c}$  of size  $m \times n$  occurs exactly once.

If all possible subarrays are used (i.e.  $(w+n-1)(h+m-1) = k^{mn}$ ), the  $(m, n)$ -window array is called *aperiodic perfect map* [133]. Opposite edges of the array can be connected together for a *periodic window array*. Of course, the windows created by the connection must also be unique (i.e.  $wh \leq k^{mn}$ ).

A periodic perfect map [99] is the 2D case of the De Bruijn torus [64]. Construction of 1D De Bruijn tori can be done by algorithms such as *prefer-one* or *prefer-opposite* [5]. From a decomposition of a N-D torus, N+1 dimensional tori can be constructed [63]. Contrary to a 1D De Bruijn sequence, the existence of a 2D torus was proven only for certain cases of  $k, h, w, m$ , and  $n$  [24]; for example when  $k$  is a prime power [123].

Aperiodic two-dimensional perfect maps have been researched extensively in the literature. Reed and Stuart [133], Ma [91] and Fan et al. [39] all gave constructions for perfect maps. Fan et al. [39] also gave proof that for each  $n$  and  $m$  there exists  $h$  and  $w$  such that  $hw = k^{nm}$  for  $k = 2$  and there exists a  $h \times w$  binary array with the  $n \times m$  window property. They labeled these  $(h, w; n, m)$ -arrays.

An alternative construction of arrays with similar properties under the name pseudorandom arrays exists in the literature. Gordon [49] and MacWilliams and Sloane [93] described binary arrays of size  $hw = 2^{nm} - 1$  such that all nonzero matrices of size  $n \times m$  appear exactly once as a window. A comprehensive description and generalization to non-binary cases of perfect maps and pseudorandom arrays was given by Etzion [38]. Perfect maps, pseudorandom arrays and arrays where not all possible windows of size  $(m, n)$  appear, but have the  $n \times m$  window property, will be referenced as *window arrays* in further description.

Unfortunately, when the orientation of the array is not known, the simple  $(m, n)$ -window property of a window array is not enough. It is possible that multiple rotations of the same window can occur in the array. *Orientable window arrays* [24] solve this problem. Orientability can be gained by using certain equivalence relation for the window property [56].

*1-orientable* arrays are ordinary  $(m, n)$ -window arrays defined earlier.

*2-orientable* arrays deal with two possible orientations (e.g. “north” vs. “south” orientation). Windows in the 2-orientable perfect maps are unique in respect to rotation by  $180^\circ$ .

*4-orientable* arrays can distinguish all four rotations of the array (e.g. “north”, “east”, “south”, “west”). The 4-orientability is reasonable only for square windows, that must be unique in respect to rotation by  $90^\circ$ . It is self-evident that 4-orientable arrays are always also 2-orientable [56].

Contrary to the 1-orientable maps, 2 and 4-orientable arrays are much less explored in the literature and no good construction algorithms existed for them before our proposed algorithm in [157] and [60]. The upper bound for the size of the 2-orientable  $(m, n)$ -window array is

$$N \leq \frac{k^{mn} - k^{\lfloor \frac{mn+1}{2} \rfloor}}{2}, \quad (3.2)$$

where  $N$  is the window count, i.e.  $N = hw$  for periodic and  $N = (h - m + 1)(w - n + 1)$  for aperiodic arrays. For the 4-orientable  $(n, n)$ -window arrays ( $n^2$ -window arrays),

$$N \leq \frac{k^{n^2} - k^{\lfloor \frac{n^2+1}{2} \rfloor}}{4}. \quad (3.3)$$

It is known that these size bounds are not tight [24]. It is not known whether any orientable perfect map exists, i.e. the inequalities may be strict.

For 2-orientable 1D window arrays, the upper bound of the size is slightly more refined, but still not tight [96]. The sequences for  $n \leq 16$  were found by brute force [24].

Similar work on 4-orientable 2D window arrays to the best of my knowledge have been missing in the literature. The only related work in computer vision and augmented reality literature using 1-orientable pseudorandom arrays was published by Morano et al. [102]. They used pseudorandom arrays submaps generated as structured light to reconstruct 3D surfaces efficiently (see Figure 3.1). Their pseudorandom array generations was done by a trial-error intuitive approach. Their work was extended by Scholz et al. [144], who printed these pseudorandom submaps on clothing. They used the printed patterns to accurately reconstruct garment motion from video (see Figure 3.2).

Binary 4-orientable window arrays can be visualized as 2D checkerboard structure, where the white and black modules are reorganized to match the values in a window array (Figure 3.3). As a more general definition, *Uniform Marker Fields* are visual patterns – fiduciary markers – made up of square modules forming a regular grid, where windows of size  $n^2$  are unique in every rotation.

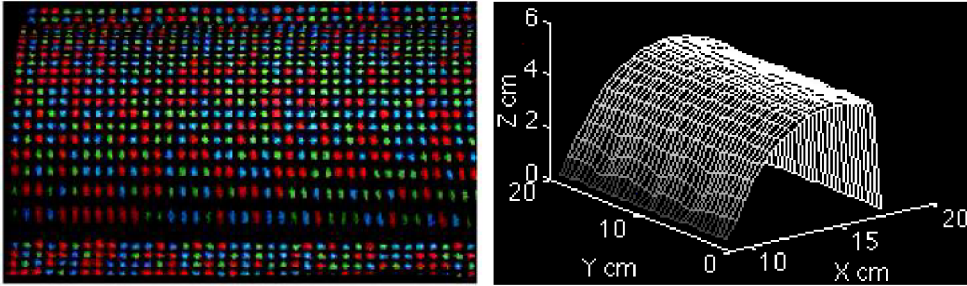


Figure 3.1: Color encoded perfect submap [102] (PSM) (left) and the 3D information reconstructed using this pattern (right).



Figure 3.2: Pseudorandom array submaps printed on cloth with 3D reconstructions of the models [144].

In this thesis, initially only binary ( $k = 2$ ) window arrays will be considered. They are easily printable and they can be easily and robustly detected in grayscale camera images. However, all the algorithms can be easily modified for higher values of  $k$ , which also led to modified marker design as seen in later sections.

### 3.2 Synthesis of Binary $n^2$ -Window Arrays

According to equation (3.3), for binary 4-orientable aperiodic binary  $n^2$ -window arrays with  $n = 3$ , a (square) map cannot be larger than  $12 \times 12$ . Our algorithm described in this section has found a number of  $11 \times 11$  arrays (Figure 3.3 left). Thus,  $3^2$ -window arrays can be used as Uniform Marker Fields, but the dimension of the field is very limited and the benefits over any existing marker designs are not very interesting. However, even for such a small array, the construction by exhaustive search is practically impossible (for a  $12 \times 12$  array, the algorithm would need to search a state space of  $2^{144}$  variants).

For the proposed binary Uniform Marker Fields,  $n = 4$  is an interesting value:  $n$  is still small enough and therefore a small fraction of the field needs to be captured by the camera in order to localize the actual view within the field. At the same time, the theoretical upper bound according to (3.3) for the dimensions of a square map is  $127 \times 127$ . By the algorithm presented in this section, 4-orientable  $4^2$ -window arrays as large as  $92 \times 92$  have been found by using a supercomputer (Figure 3.3 right). Also, rectangular arrays of similar  $w \times h$  areas have been generated – refer to Section 3.2.3 for more information.

For  $n = 5$ , the arrays of reasonable dimensions can be generated (more or less) randomly; however, the required portion of the marker field to be visible by the camera ( $5 \times 5$  modules) is unnecessarily high. For practical purposes, the  $4 \times 4$  window is therefore of most interest and in the further text, the algorithm’s properties will be explored for these window dimensions.

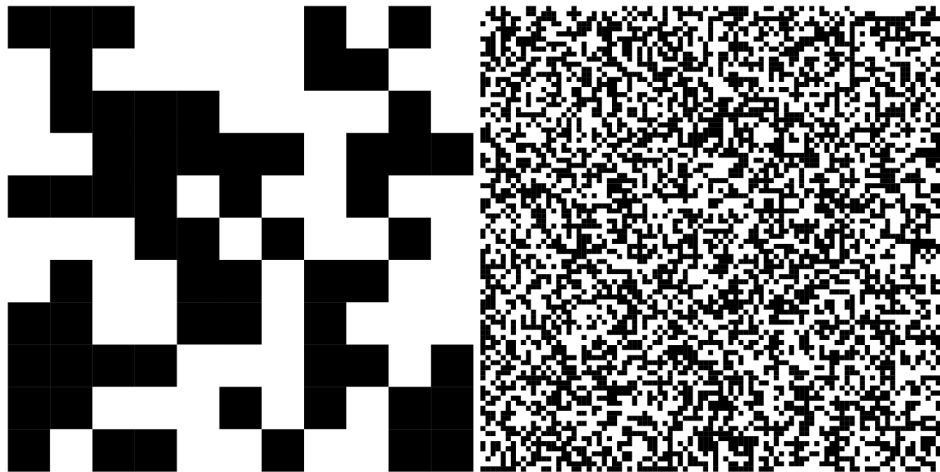


Figure 3.3: Largest generated 4-orientable  $n^2$ -window arrays using our synthesis algorithm, with  $n = 3$  on the left and  $n = 4$  on the right.

The literature does not provide any efficient construction method for 2 or 4-orientable  $n^2$ -window arrays. Burns and Mitchell [24] provided a baseline construction for 4-orientable window arrays. However, this construction is far from optimal (i.e. for a given  $n$ , the  $w, h$  dimensions are small) and, more importantly, the constructed map cannot be binary, but its arity is given by combining two helper 1D 2-orientable De Bruijn sequences. Exhaustive search is not feasible as a construction method: for example, for a  $4^2$ -window array  $90 \times 90$  modules large, the area of the map to be searched for is 8100 modules and the state space is just too large ( $2^{8100}$ ).

This section presents considerations leading to a genetic algorithm for synthesis of 4-orientable window arrays. The algorithm is still very computationally demanding; however, it can be parallelized (as described in Section 3.2.2) and executed on a large number of computation nodes. We harnessed a supercomputer of around 1 000 nodes and generated usable window arrays. The successfully generated binary arrays are described in Section 3.2.3.

### 3.2.1 Random Maps

Let us first consider a binary map  $(a_{i,j} \in \{0, 1\})$  generated randomly. The theoretical problem that a randomly generated periodic array will contain 1-orientable conflicting windows is analogous to the *birthday problem* [103]. Its probability can be approximated as the:

$$p(k, h, w, m, n) \approx 1 - \exp\left(-\frac{(hw)^2}{2k^{mn}}\right). \quad (3.4)$$

For 2 and 4-orientable arrays with square windows, the birthday conflict probability is

$$p(k, h, w, n) \approx 1 - \exp\left(-\frac{o(hw)^2}{2k^{n^2} - 2k^{\lfloor \frac{n^2+1}{2} \rfloor}}\right), \quad (3.5)$$

where  $o \in \{2, 4\}$  is the orientability. The equations above considers conflict only between different windows. There is significant probability that some windows can be self-conflicting, for example due to symmetry. The probability that an array contains self-conflicting windows is

$$p(k, h, w, n) = \left(1 - k^{\lfloor \frac{n^2+1}{2} \rfloor - n^2}\right)^{hw}. \quad (3.6)$$

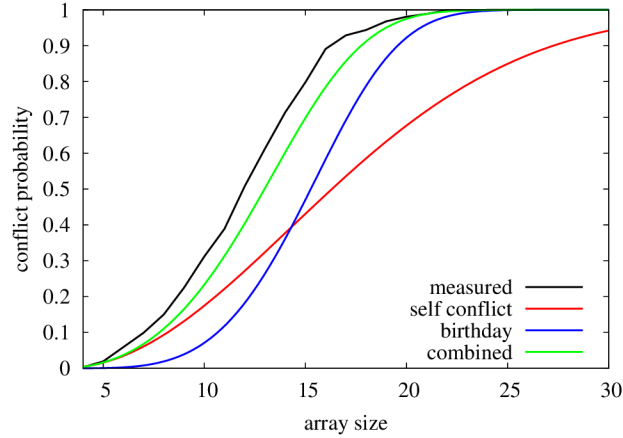


Figure 3.4: The probability of conflicting windows in a random square map of a given size (binary map,  $4 \times 4$  windows).

Figure 3.4 shows the probability of conflict for square aperiodic arrays with  $k = 2$  and  $n = 4$ . The experimental measurement was done using 1000 randomly generated arrays. The number of conflicts caused by self-conflicting windows is a significant factor. The graph shows that arrays of small sizes can be generated randomly and there is significant probability that it will still satisfy the  $n^2$  window array property.

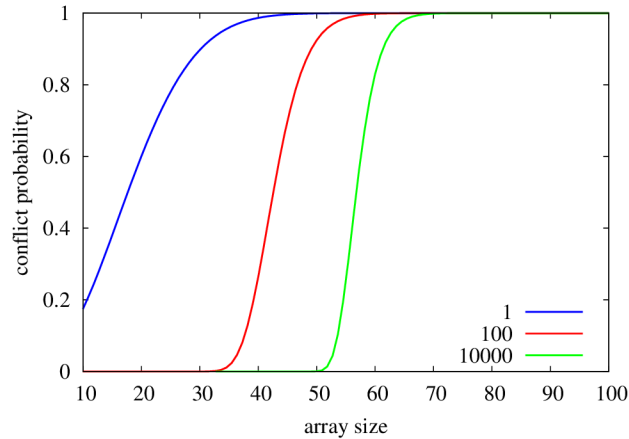


Figure 3.5: Approximate probability that the randomly “replaced” content will be in conflict with another window or itself after  $N$  trials of replacement (binary map,  $4 \times 4$  windows).

With growing array size, the conflicts become inevitable for randomly generated arrays. An intuitive solution is to “replace” conflicting windows with alternative content. For a randomly generated  $n^2$  window in a periodic map, otherwise free of conflicts, the probability that the window does not conflict with the rest of the map is roughly

$$p(k, h, w, n) \approx \left( 1 - o \frac{hw - (2n - 1)^2}{2n^2 - 2 \lfloor \frac{n^2+1}{2} \rfloor} \right)^{(2n-1)^2}. \quad (3.7)$$

The internal conflicts in the block around the window are approximated using equation (3.5). Figure 3.5 shows the probability that the new window will be in conflict with an existing window after a certain amount of tries.

Figures 3.4 and 3.5 should illustrate the possibility of generating the window array randomly. For  $k = 2$  and  $n = 4$ , small arrays (around  $15 \times 15$ ) can be generated totally randomly and some of the generated arrays would comply with the 4-orientable  $n^2$ -window property. Random arrays of dimensions around  $50 \times 50$  can be fixed by replacing the conflicting windows with random content; after a moderate number of such trials for replacement, the array will achieve the  $n^2$ -window property. However, for arrays of higher dimensions (e.g.  $90 \times 90$ ), the process of fixing the array by replacing the conflicting tiles with random content can be very time consuming.

### 3.2.2 Genetic Algorithm for Synthesis of Window Arrays

The considerations from the previous section led to a genetic algorithm which works with maps containing conflicts and improves it continually by mutations that lead to decreasing the conflict count. To initialize the population *individuals* can be generated randomly. Alternatively, they can be created from a smaller window array extended by randomly generated rows and/or columns.

The genetic algorithm can be characterized by these terms:

- For the initial population we use a number of copies of the same array, or various arrays are generated randomly.
- The fitness function of an individual is based on the number of conflicts in the given array  $f(A) = \frac{1}{c(A)+1}$ .
- The fitness threshold, where the algorithm is stopped is set to 1 (the algorithm is looking for conflict-free maps).
- For selecting members for the next generation, rank selection is used.
- Mutation is defined as replacing a window with randomly generated content. The windows are selected randomly; the conflicting windows have a higher probability of being selected for replacement.

In order to improve the algorithm's convergence we made a few modifications to the aforementioned general approach. In the mutation step, the algorithm discards from random selection all windows that are already in the map or that would cause more conflicts than the current count. The other changes to the algorithm described below were carried out in order to further improve the speed of convergence on the proposed client-server configuration.

In order to distribute calculations required to solve the conflicts in randomly generated arrays, we used a *client-server* architecture. The server keeps track of the population, while the clients query the server for computational tasks and send back the results.

The *server* stores the active population and distributes tasks for mutations to the clients. The server itself is state-less in the sense that it does not keep track of the clients and their status. The most important change from the described genetic algorithm is that the server does not separate arrays into several generations, but works with a single population and updates it incrementally. This is necessary because the run-time of the clients might differ dramatically, and clients might post changes into different generations.

As mentioned earlier, each *client* queries the server for a new task after it is started or when it finishes a task. The client receives a single array and tries to solve conflicts in the array by mutation: i.e. by replacing a conflicting (or non-conflicting) window with a better combination of modules. If it successfully finds a different array with the same or smaller number of conflicts, it sends the array back to the server. Otherwise, the client discards the mutated array and requests a new one from the server.

### 3.2.3 Synthesized Window Arrays

We generated a set of binary aperiodic 4-orientable  $n^2$ -window arrays using the client-server architecture describe above. The data set includes rectangular maps of different aspect ratios:

- **1:1** – square marker fields,
- **$\sqrt{2}:1$**  – marker fields suitable for office paper sizes,
- **2:1** – marker fields for rectangular areas,
- **3:1** – marker fields for wide rectangular areas.

Table 3.1 gives the highest resolutions of the window arrays available for the respective aspect ratios in the data set. There are several important things to note about the size of the generated markers.

aspect ratio	available dimension
1:1	$92 \times 92$
$\sqrt{2}:1$	$110 \times 78$
2:1	$122 \times 61$
3:1	$159 \times 53$

Table 3.1: Available sizes of the binary 4-orientable aperiodic  $4^2$ -window arrays.

Although larger markers allow covering larger areas, the distance from the marker while it is still detectable depends largely on the chosen module size. For example, the generated  $92 \times 92$  marker with modules size  $1 \text{ inch} = 2.54 \text{ cm}$  would have an edge  $2.34 \text{ m}$  long. Such marker could theoretically be detectable from distance  $\sim 15 \text{ cm}$ , to fit a  $n \times n$  unique window, to  $\sim 2.5 \text{ m}$ , where the checkerboard nature of the marker starts to disappear making detection difficult. A marker with window-array size  $9 \times 9$  and module size  $25 \text{ cm}$  on the other hand would be detectable with the same camera and resolution in the distance interval  $\sim (1.5 \text{ m}, 20 \text{ m})$ . These values largely depend on the camera’s field of view and resolution. This fact makes the choice of module size and required window array size application specific.

A disadvantage of using large markers where the location inside the marker needs to be detected even from the smallest unique sub-window is the high probability false-positive detection in case of noise or occlusion. With growing marker sizes nearing the theoretical limits, the average hamming distance between individual unique sub-windows is reduced. Smaller markers can be generated to maximize the average Hamming distance between unique  $n \times n$  sub-windows. On the other hand, where the application allows it, larger marker sizes are desirable to fit as many modules into the camera’s view as possible for more reliable location identification.

The final important things to note about synthesized window arrays, is that a single large window array can be separated into several disjunctive smaller sub-marker fields. All the resulting marker fields will have the  $n^2$  unique window property. In this set of marker fields, each of them is uniquely identifiable. For example, given the generated  $92 \times 92$  marker field divided into uniquely identifiable marker fields of size  $9 \times 9$ , would result in 100 unique marker fields.

## 3.3 Detection of Marker Fields and Camera Pose Estimation

My main contribution related to Uniform Marker Fields is the efficient detection algorithm. This work was published in [157] and the description below is the extension of this work. Uniform Marker Field was designed to cover large planar area and is composed of mutually overlapping uniquely identifiable partial markers (see Figure 3.6). The design was inspired by Fractal Marker Fields,

proposed by Herout et al. [59] and by ARTag, the work of Fiala et al. [41]. The former removed the limitation of existing fiduciary markers of using only a single scale to encode information over a given area of a planar surface. Unfortunately, the described detection algorithm is not suitable for mobile processors, mostly caused by the memory and performance requirements of the Hough transformation (PC-lines). ARTag still used conventionally designed fiduciary markers to build a large-scale field or reference model from them. For more detailed description of these approaches, see Section 2.3.1.

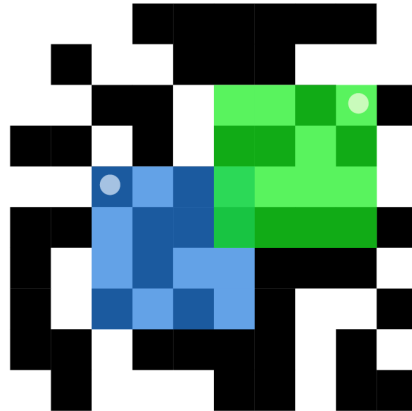


Figure 3.6: A fragment of the marker field. The overlapping markers are unique in all possible rotations. Two markers are highlighted with different colors - the point in the left top corner indicates rotation.

Conventional marker detectors typically rely on first detecting the bounding borders [76, 41] of the markers by finding the contours in a thresholded image and choosing shapes consisting of four straight-line contours. The Uniform Marker Field construction does not distinguish between marker design features intended for general marker *localization* and features for marker *identification*. Checker-board modules serve simultaneously as the localization and identification features. This approach is more space efficient – no valuable area is taken up by thick black borders and quiet zones. The second advantage from camera pose estimation viewpoint is the more uniformly distributed points of interest – the corners between modules. For conventional markers, only the 4 corners are used as 2D-3D correspondences. In an ideal case, a single marker inside a  $4^2$ -window array could provide 25 correspondences. Theoretically this could provide more robustness against occlusion or other outliers and result in higher camera pose estimation precision.

### 3.3.1 Overview

The detection algorithm was designed so that it visits as small a fraction of the image pixels as possible. This is important mostly on mobile platforms, where the memory access is slow and the processor's L2 and L3 (if present) caches are small relative to their desktop counterparts. The algorithm performs the following main steps:

1. **Extraction of edgels** (edge element or connected edge pixels; term borrowed from Martin Hirzer [61]) – typically, the algorithm extracts around one hundred straight edge fragments in the whole image for VGA resolution, but generally the number of edgels depends highly on the used marker and the resolution of the image. Edgels are described by an image point and edge orientation (vector) or by two endpoints.



2. **Determination of two dominant vanishing points** among the edgels. The vanishing points define the horizon (a line connecting the vanishing points).
3. **Finding the grid of checker-board edges** as two groups of regularly repeated lines coincident with each vanishing point. These groups will be referred to as *fans* in the further text.
4. **Extraction of checker-board modules** using the grid and localization of the camera view within the 4-orientable  $n^2$ -window array.

I describe each of these steps in more detail below. The algorithm assumes that a significant portion of the input image is covered by the marker field. In our earlier research [158] we showed that the image can be efficiently searched for areas likely to contain a checker-board pattern thus pre-selected rectangles can be supplied as the input to the recognizer (see Section 4). In high-resolution images, the marker fields can thus be found even if it covers an arbitrary fraction of the camera input.

### 3.3.2 Extraction of Edgels

Yu et al. [182] in their work on Gridding Hough Transform showed that for long straight lines in the image, a relatively sparse grid of horizontal and vertical scanlines can be used to detect the edges. Also, some of the “conventional” marker detectors [76, 41] scan the image in horizontal and vertical scanlines and detect edges (or edgels), which in turn are connected into segments.

To detect an edge point  $\mathbf{p}_0$  on a scanline, our algorithm uses adaptive thresholding based on a moving average window (similarly as in the state-of-the-art marker detectors). The Sobel operator is applied in order to obtain a *rough approximation*  $\mathbf{s}_0$  of the slope of the corresponding checker-board edge line using the gradient direction  $\mathbf{n}_0$ .

In order to improve the accuracy of the slope in each direction of the slope, more edge pixels are searched for. Let  $w$  be an offset parameter (a good offset could be 5 pixels). Edge points  $\mathbf{p}_i$  (i.e. image locations with a high gradient similar to  $\mathbf{n}_0$ ) are searched for in the proximity of points  $\tilde{\mathbf{p}}_i$  on the presumed edge line:

$$\tilde{\mathbf{p}}_{i+1} = \mathbf{p}_0 + iw\hat{\mathbf{s}}_i, \quad (3.8)$$

where  $\hat{\mathbf{s}}_i$  stands for normalized vector  $\mathbf{s}_i$ . If an edge point is indeed found, the slope is updated using the new edgel’s endpoint

$$\mathbf{s}_{i+1} = \mathbf{p}_{i+1} - \mathbf{p}_0, \quad (3.9)$$

and the same procedure is repeated for increasing  $i$  and also for the opposite direction  $i \in \{-1, -2, \dots\}$ . The iteration is stopped when no further edge point is found in the given direction or the edgel’s length is over a given threshold. The threshold helps avoid situations, where one long and strong line in the image would cross multiple scanlines and from each point separate edgel lines would be extracted with the same parameters and similar lengths. A good threshold is  $2l_s$ , where  $l_s$  is the offset between scanlines in pixels.

Only the reliable edgels are stored for further processing: if points  $\mathbf{p}_i$  are not collinear with a sufficient tolerance, or if an insufficient number of edges  $\mathbf{p}_i$  is found leading to short edgel length, the edgel is dropped. When the edgels are reliable, further steps in the algorithm do not require a very high number of them (around 100) to successfully detect the markers. Experiments show that approximately 10 scanlines in horizontal and in vertical direction are sufficient for extracting enough edgels for VGA resolution. See Figure 3.7 for an illustration of the edgels that are extracted from an input image.

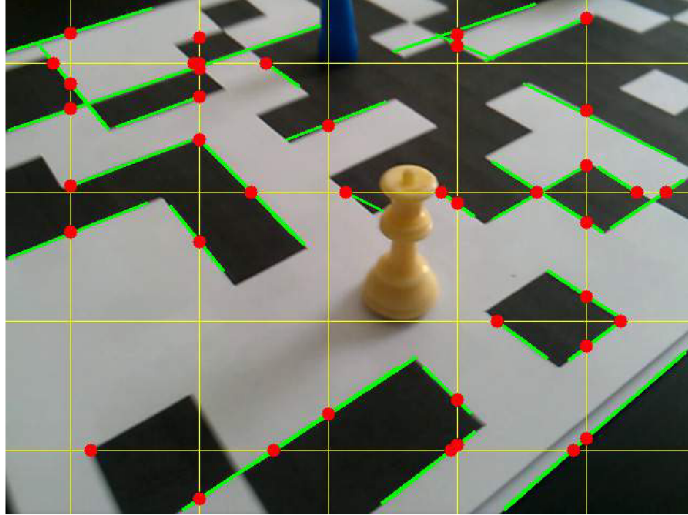


Figure 3.7: *Yellow* – horizontal and vertical scanlines that are processed by the algorithm. In the actual implementation, a higher number of scanlines is used (around 10 in each direction). *Red* – edges  $p_0$  detected on the scanlines. *Green* – edgels detected around the “starting” red dots.

### 3.3.3 Identifying Vanishing Points

After edgels were successfully extracted, the algorithm clusters them into two groups based on their slope. These groups should correspond to two main directions of the edges in the marker. Such grouping seems sufficient for the detection; however, in noisier data, a RANSAC-like selection of groups of edgels coincident with a common vanishing points is done. In order to improve the reliability of clustering, a heuristic is used during clustering, where the angle difference between the two groups’ slope angle should be around  $90^\circ$ . The two selected groups of lines (edgels extended to infinity) are shown in Figure 3.8.

Using the described method, a cluster of lines is obtained without any disturbing outliers. Such a cluster can be used to compute the vanishing point for the cluster fairly precisely. Using homogeneous coordinates for the vanishing point  $\mathbf{v}$  and the cluster of lines  $\mathbf{l}_i$ , all the lines must be coincident with the vanishing point, i.e.

$$\forall i : \mathbf{v} \cdot \mathbf{l}_i = 0. \quad (3.10)$$

The coordinates of the lines in the real projective plane form a 3D vector space without an origin (with an equivalence relation). Points of the real projective plane correspond to hyperplanes passing through the origin, so the vanishing point can be found by fitting a hyperplane through all the observed lines – edgels. The line vectors  $\mathbf{l}_i$  are scaled so that each one’s magnitude corresponds to the edgel length. This way, the longer and more reliable edgels are favored. The hyperplane’s normal is found as the direction of the least variance by eigendecomposition of the correlation matrix

$$C = (\mathbf{l}_0 \dots \mathbf{l}_N)(\mathbf{l}_0 \dots \mathbf{l}_N)^T. \quad (3.11)$$

Since the matrix  $C$  is  $3 \times 3$  and symmetric, the eigenvalues and vectors can be computed very efficiently through QR decomposition [46].

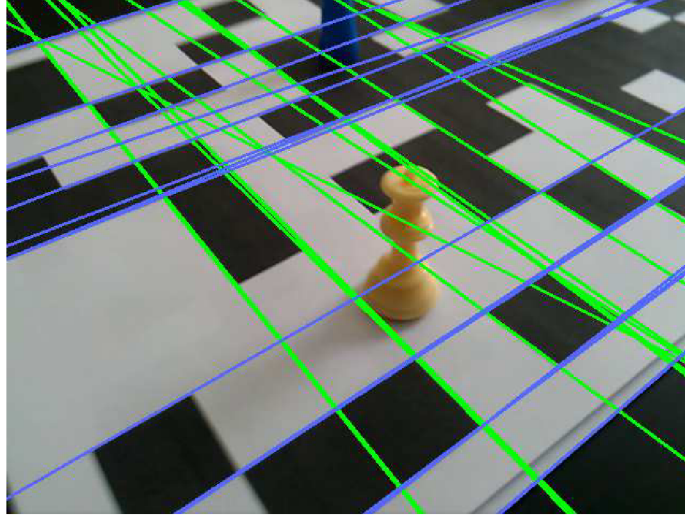


Figure 3.8: *Blue, Green* – Two groups of lines (extended edgels). For each of the groups, a vanishing point is found by using hyperplane fitting with eigendecomposition.

### 3.3.4 Finding Two Fans of Grid Lines

After the two vanishing points  $\mathbf{v}_1, \mathbf{v}_2$  are found for the two groups of edgels, the horizon (i.e. the line that passes through both of the vanishing points) is computed as  $\mathbf{h} = \mathbf{v}_1 \times \mathbf{v}_2$ . The lines in the marker corresponding to the edges of the grid squares at one direction can be computed using the horizon as ( $\hat{\mathbf{x}}$  denotes normalized vector)

$$\mathbf{l}_i = \hat{\mathbf{l}}_{base} + (ki + q) \hat{\mathbf{h}}, \quad (3.12)$$

where  $\mathbf{l}_{base}$  is an arbitrarily chosen base line coincident with the vanishing point, different from the horizon [143]. Parameter  $k$  controls the line density and  $q$  determines the position of the first line. A good simple choice for  $\mathbf{l}_{base}$  is a line through the center of the image (and through the vanishing point).

To find  $k$  and  $q$ , the value of  $(ki + q)$  is calculated for every line (extended edgel) of the input group. These values are clustered, each cluster is assigned an  $i$  and then overall optimal  $k$  and  $q$  are found by linear regression. Blue and green lines in Figure 3.9 are the two fans of lines  $\mathbf{l}_i$  corresponding to the two vanishing points.

The clustering at this stage can be very simple due to the low number of lines. The small number of lines also makes more sophisticated grouping of lines (k-means clustering, Fourier transformation to detect dominant frequency) problematic. For the small number of lines, the  $(ki + q)$  values are sorted and the differences between neighboring values are computed. The most frequent difference over a small threshold is chosen as the slope  $k$ .

### 3.3.5 Extraction of the Checker-Board Modules

Once the two fans of lines defining the edges of the checker-board grid are found, they are both defined by Equation (3.12). Their definitions are differing only in  $\hat{\mathbf{l}}_{base}$ , because each one is related to a different vanishing point, and each has its pair of  $k$  and  $q$ :

$$\mathbf{l}_i^{(1)} = \hat{\mathbf{l}}_{base}^{(1)} + \left(k^{(1)}i + q^{(1)}\right) \hat{\mathbf{h}}, \quad (3.13)$$

$$\mathbf{l}_j^{(2)} = \hat{\mathbf{l}}_{base}^{(2)} + \left(k^{(2)}j + q^{(2)}\right) \hat{\mathbf{h}}. \quad (3.14)$$

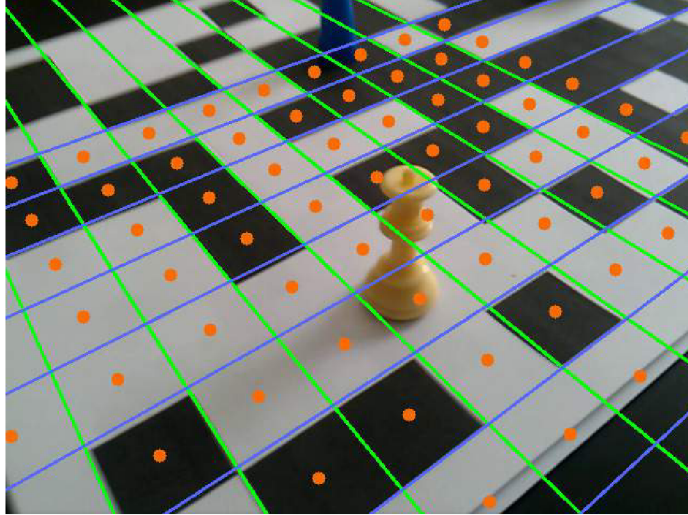


Figure 3.9: *Blue, Green* – lines  $\mathbf{l}_{(i)}^{(1)}$ , and  $\mathbf{l}_{(j)}^{(2)}$ , respectively. *Orange* – points  $\mathbf{x}_{ij}$  sampled from the input image.

Points

$$\mathbf{x}_{ij} = \mathbf{l}_{(i+1/2)}^{(1)} \times \mathbf{l}_{(j+1/2)}^{(2)}, \forall i, j \in \mathbb{N} \quad (3.15)$$

are intersections of lines right between the edge lines: points in the middle of the checker-board square modules. These locations are sampled from the input image. The resulting bitmap (which has very low resolution) is adaptively thresholded. The  $n^2$ -windows are looked up in a hash table to determine the location within the marker field. The sampled points  $\mathbf{x}_{ij}$  are illustrated by Figure 3.10.

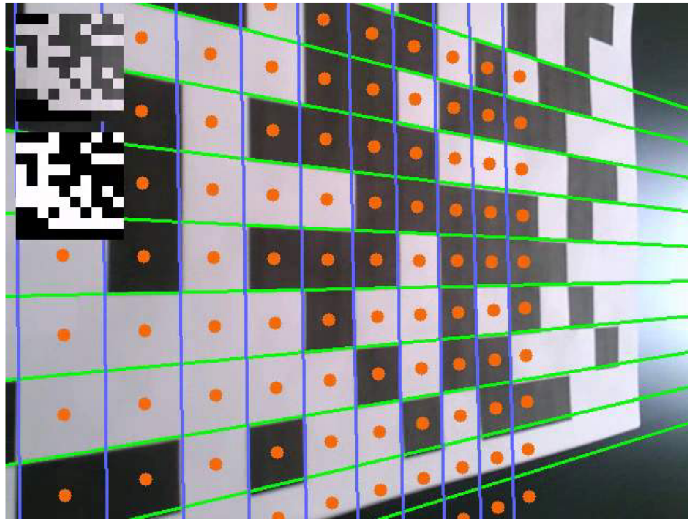


Figure 3.10: *Orange* – points  $\mathbf{x}_{ij}$  sampled from the input image (left top box) then filtered using adaptive threshold to get a the binary map (left bottom box).

### 3.3.6 Location Recovery and Camera Pose Estimation

Once the sampled values have been filtered using an adaptive threshold, each  $4 \times 4$  window's location inside the sampled region is found using a hash function. The global location and orientation of the

extracted region inside the marker field is decided by the most votes. Given the known locations for all on-screen sample points, the 3D positions of the module centers is known. This information can be used to recover the camera pose using standard methods (Section 2.2). In this early work we used an iterative method [89] based on Levenberg-Marquardt optimization as implemented in the OpenCV<sup>1</sup> library.

This algorithmic approach had several flaws. The image needs to be rectified to remove radial distortion and achieve high precision vanishing point estimation. The assumption of minimal camera distortion leads to imprecise camera pose estimation. The image distortion is also clearly visible on Figures 3.7 – 3.10.

A weak trait of this approach is during the third step of the algorithm, approximating  $k$  as the average most frequent difference between groups of lines belonging to a single vanishing point. With motion blur or depth of field blur, the computed  $(ki + q)$  parameters for edgels associated with a single line in the image will have significant deviation. This will cause the algorithm to consistently underestimate the parameter  $k$ , since the gaps between groups of lines would be smaller.

Another weakness is the dependency of the camera pose estimation on the precision of the marker grid parametrization, since the computed module centers are used in this step. If the assumptions of correct camera calibration and precise grid parametrization held, such global solution for the camera pose would be more robust and precise when compared to reliance on noisy corner points. These weaknesses and flaws have been resolved in our follow-up work described in Section 3.4.

### 3.3.7 Experimental Results

For testing purposes of this initial solution, I collected a set of videos acquired by 3 different smartphone cameras at resolution  $640 \times 480$  or  $720 \times 480$  with 24 frames per second, each 20 to 30 seconds long. Smartphones used for collecting the data were:

- **Desire** – HTC Desire GSM ( $720 \times 480$ ),
- **Evo** – HTC Evo 3D ( $720 \times 480$ ),
- **Titan** – HTC Titan ( $640 \times 480$ ).

In order to evaluate the detection precision for different types of movements, I split the dataset into 6 *categories* according to the dominant *movement* manifested in each video:

- **zoom** – zooming in and out on the marker field,
- **horizontal** – horizontal movement over the marker field,
- **rotation** – rotating the camera over,
- **perspective distortion** – videos where the marker field is skewed by perspective projection,
- **general movement** over the marker including several of the above mentioned distortions, and
- **occlusion** – objects covering some parts of the marker field.

Each category contains 6 videos: 2 for each smartphone. In the videos, the marker field is visible in all frames. I used two different marker fields. Both marker fields were printed out black and white on standard office papers. The low-density marker field's resolution was  $14 \times 10$  and the high-density marker field's was  $28 \times 19$ . The total number of videos in this dataset was 36. Example frames from the video dataset are shown in Figure 3.11.

In the first set of experiments I evaluated the success rate of detecting the checker-board marker field and recognizing a location within it. To estimate a baseline of the algorithm's robustness we

---

<sup>1</sup><http://opencv.willowgarage.com>

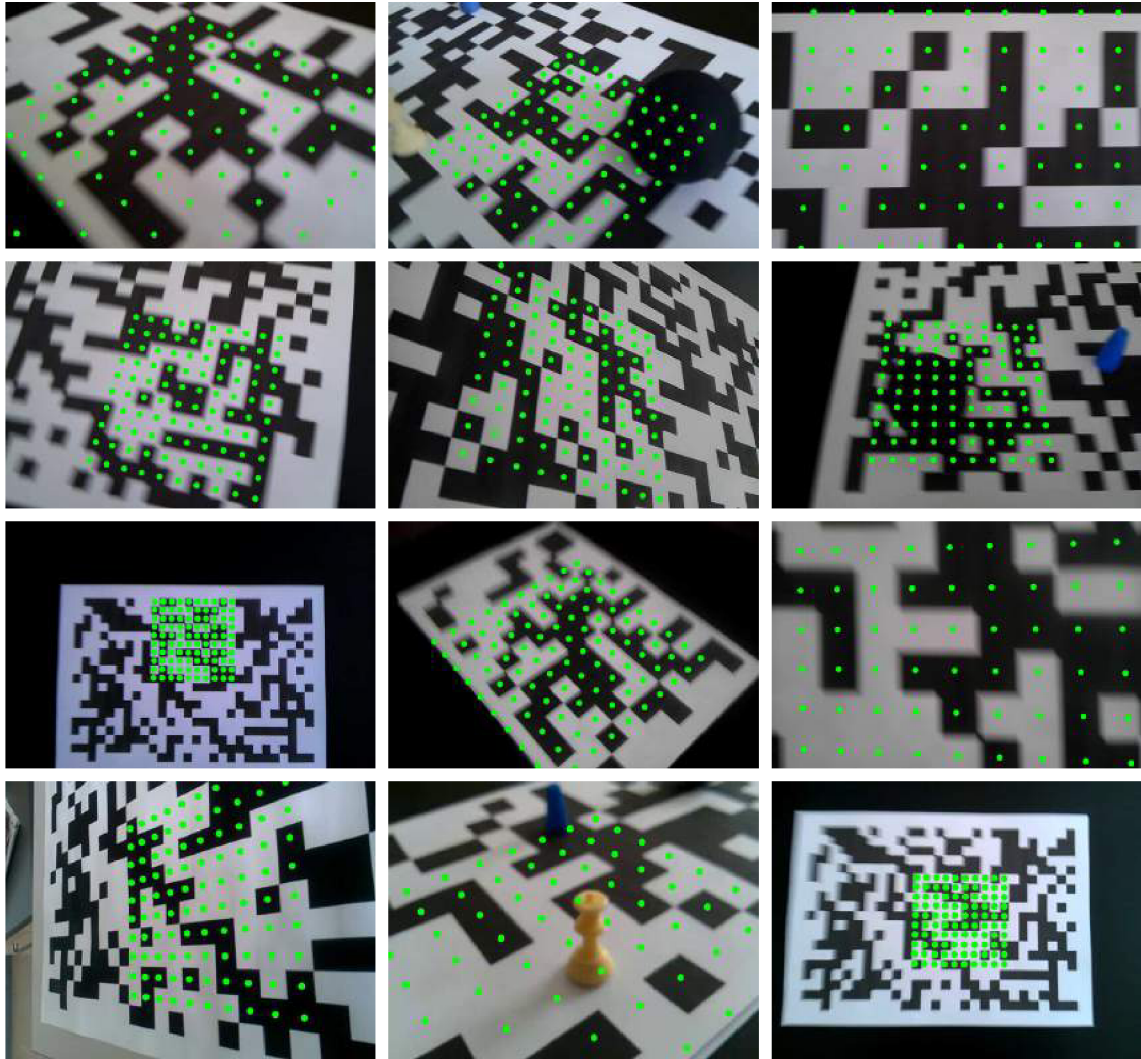


Figure 3.11: Sample images from the dataset. Green dots are the detected module center points  $\mathbf{x}_{ij}$  sampled for localization (Section 3.3).

did not use information from previous frames. Using such information in real applications should improve the precision of the algorithm, as well as improve the processing speed.

Table 3.2 contains the success rate results for the video dataset. *Successfully recognized frame* is a frame, where the detector was able to find the grid, sample the  $\mathbf{x}_{ij}$  points and the resulting binarized bitmap (or its significant part) was successfully localized inside the Uniform Marker Field.

The results show that our algorithm performs well even for very challenging videos (see the dataset) with rapid movement causing directional blur, rotation, and high perspective distortion. The low-density marker performed better in almost all categories. This is caused by the longer edges, hence more precise edgels. In the case of the occlusion category, the higher density marker still contains enough unoccluded fields to enable successful detection, unlike in some frames with the lower density marker. The difference for the horizontal category is caused by a statistical error caused by rapid movement and excessive motion blur in one of the videos. In the case of zoom category, the low density marker is detectable from larger distance, while the high density marker's grid was difficult to retrieve due to dense on-screen edgel distribution.

Category	Low density	High density
Zoom	94.5 %	92.0 %
Horizontal	97.3 %	99.4 %
Rotation	99.9 %	99.0 %
Perspective	99.8 %	99.4 %
General movement	95.3 %	95.0 %
Occlusion	91.9 %	92.5 %

Table 3.2: Success rate for detecting the position in the marker with different categories and marker densities.

	Success rate	Visited pixel percentage
<b>Smartphone</b>		
Evo	95.3 %	5.50 %
Desire	96.3 %	5.79 %
Titan	97.4 %	5.68 %
<b>Marker density</b>		
Low	96.5 %	5.83 %
High	96.1 %	5.59 %

Table 3.3: Success rate and percental evaluation of the visited pixels in the image by our algorithm.

Table 3.3 contains the success rate breakdown for the videos based on the used smartphones and marker density. Smartphone Titan gave the sharpest image and best picture quality, which helped improve the success rate. The videos captured by smartphones Evo and Desire contain an intensive motion blur which caused a drop in the performance. The results also show that the algorithm visited and used for computation only less than 6 % of the total number of pixels in the image. Since one of the slowest operations on mobile processors is randomly accessing large chunks of memory, our algorithm is well suited for smartphones.

Table 3.3 also shows the difference in the visited pixel percentage for marker fields with different densities. Since the edges in the image are longer in the low density markers, the algorithm visits more pixels along these edges. This helps the precision of the edgels and also the detection performance slightly.

In order to get a comprehensive analysis about the computational complexity of the algorithm, I measured the required time of different components of the detection algorithm. Table 3.4 shows the percental distribution of computational time between different components. The most time-consuming part of the algorithm is the camera localization as implemented in the OpenCV library. The computational times were measured on an Intel Core i5 661, 3.3 GHz with a DDR2 memory.

Algorithm part	time	percent
Scanlines	0.21 ms	16 %
Edgel extraction	0.22 ms	16 %
Vanishing points and Grid	0.11 ms	8 %
Module extraction	0.06 ms	5 %
Camera localization	0.74 ms	55 %
<b>Overall</b>	<b>1.34 ms</b>	<b>100 %</b>

Table 3.4: Breakdown of computational time into different parts of the algorithm.

### 3.4 Five Shades of Gray

In the previous sections, I described the base Uniform Marker Fields design and its detection algorithm. I also pointed out several weaknesses of the proposed detection algorithm in Section 3.3.6. The precision of the pencil detection was insufficient for augmented reality applications. Although it gave a global solution for the marker field orientation and position, the detection of the grid was noisy due to outliers and numerical calculations. In this section a generalization to Uniform Marker Fields is presented (Figure 3.12). The description below is based on our publication [60].

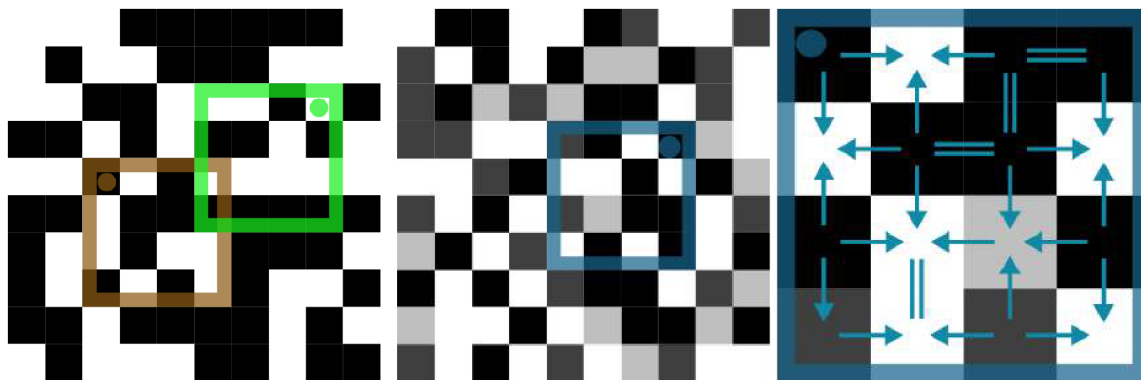


Figure 3.12: Marker Field design patterns. **left:** The basic black and white marker. The highlighted green and brown windows are unique in the marker field, just as all other sub-windows in every rotation. **right:** The marker with several shades of grey. The highlighted blue sub-window is unique in the map considering the edge directions as seen on the extracted region.

#### 3.4.1 Improved Uniform Marker Fields Design

As defined in Section 3.1, *aperiodic 4-orientable binary  $n^2$ -window arrays* [157, 24] are matrices  $A = (a_{ij} \in \{0, 1\})$ , where each square sub-window  $A_{rc}$  of dimensions  $n \times n$  appears only once, including all four rotations. If any of the windows appeared more than once, we would be speaking of a *conflict* – either a mutual conflict between two different windows (possibly rotated) or a self-conflict, where a window is self-similar after rotation. In Section 3.1, we interpreted such a binary array as a black-and-white checkerboard and proposed to use it as a marker field for augmented reality. The unique  $n^2$ -windows largely overlap. Thanks to this overlap, only a small fraction of the marker field must be visible in order to be detected and recognized.

In our follow up work [60], we generalized the Uniform Marker Fields construction to grayscale or color  $k$ -ary marker fields ( $a_{ij} \in \{0, \dots, k-1\}$ , Figure 3.13). Mathematically if the shades of gray or exact colors could be recovered, these would correspond to window-arrays with  $k$  defined as the number of shades used.

Technically, however, in comparison with binary marker fields the absolute grayscale or color values of the grid modules cannot be reliably discerned under varying lighting and camera conditions. That is why we used the *edge gradients between the modules* in a single  $n \times n$  window as the unique window array property for localization within the marker field. Horizontal (3.16) and vertical (3.17) edge gradients are defined as:

$$e_{ij}^{\rightarrow} = a_{i,j+1} - a_{ij}, \quad (3.16)$$

$$e_{ij}^{\downarrow} = a_{i+1,j} - a_{ij}. \quad (3.17)$$



The absolute value of the edge gradient is also hard to recognize reliably and thus only the basic character of the edge is used for recognition:  $g_{ij}^* = \text{sgn } e_{ij}^* \in \{-1, 0, +1\}$ . The  $n^2$ -window used for localization within the marker field then is (Figure 3.13):

$$G_{rc} = (g_{rc}^{\rightarrow}, \dots, g_{(r+n-1, c+n-2)}^{\rightarrow}, g_{rc}^{\downarrow}, \dots, g_{(r+n-2, c+n-1)}^{\downarrow}), \quad (3.18)$$

where  $G_{rc}$  is the unique window at position  $(r, c)$  inside the window array. Given this ternary classification of edges, grayscale markers can be seen as a generalized version of  $k = 3$ -ary  $n^2$ -window arrays, and color marker fields as  $k = 3c$ -ary  $n^2$ -window arrays, where  $c$  is the number of channels in the used color model. Note that the number of shades  $s \geq n^2$  has no effect on the possible variations for edge directions inside a window and has minimal effect on a global scale. For  $s < n^2$ , the number of shades limits the available pool of edge configurations, reducing the size of possible marker fields. The number of shades also has an impact on the detector's performance, with low number of shades causing shorter and less edgels, and high number of shades reducing the ability to differentiate between shades under varying lighting conditions. Based on empirical data, 5 shades for a grayscale marker gave a good compromise.

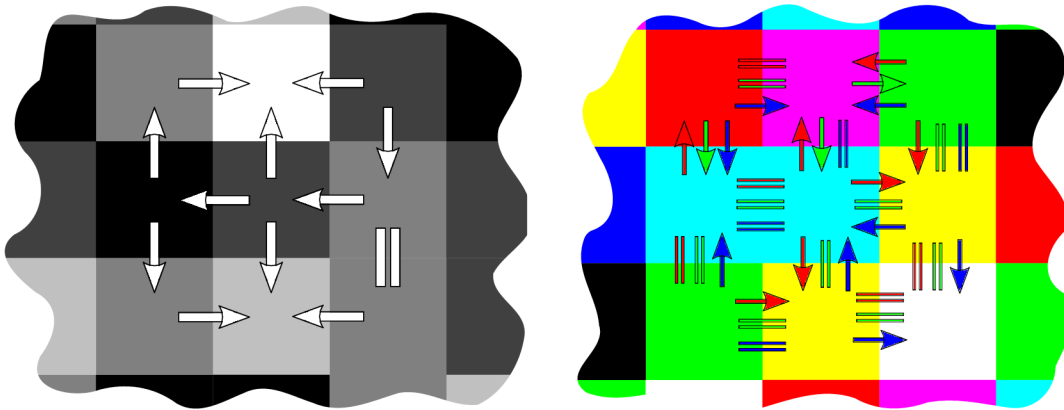


Figure 3.13: A fragment of the marker field. **left:** Five shades of grey. **right:** 8 different colors. Arrows across the edges illustrate the observable gradient which describes an individual line. In color, multiple gradients can be observed at one edge (e.g. RGB).

Synthesis of the marker field is done in a manner similar to the genetic algorithm sketched out in Section 3.2. In this case, the fitness function additionally also reflects the quality of edges between the modules – edges with higher absolute value  $|e_{ij}|$ , representing higher intensity difference between modules are preferred. This property ensures more distinct edges for the grid detector and also makes edge direction detection more robust.

### 3.4.2 Recognition of Planar Greyscale Grids of Squares

This section describes the revised algorithm based on the algorithm presented in Section 3.3 for detection of the grayscale checkerboard-like marker field. As before, the algorithm supposes that the grid of squares is planar and distorted by a perspective projection. The experiments (Section 3.4.3) show that this condition is fulfilled enough in realistic scenes observed by standard cameras. Thanks to this assumption, the algorithm is very efficient: the fraction of visited pixels (the algorithm's “pixel footprint”) within an average input image is very small (Section 3.4.3.2).

### 3.4.2.1 Greyscale Grid Detection

Conventional marker detectors typically rely on first detecting the bounding borders [76, 41] of the markers by finding the contours in a thresholded image and choosing shapes consisting of four straight-line contours. The grayscale or color Uniform Marker Fields, similar to binary Uniform Marker Fields, does not distinguish between marker design features intended for general *marker detection* and features for *marker identification*. Grid modules serve simultaneously as the detection and identification features. The motivation for this approach is to better use the marker field's surface: the localization features are much denser in the field, while still preserving the identification capabilities.

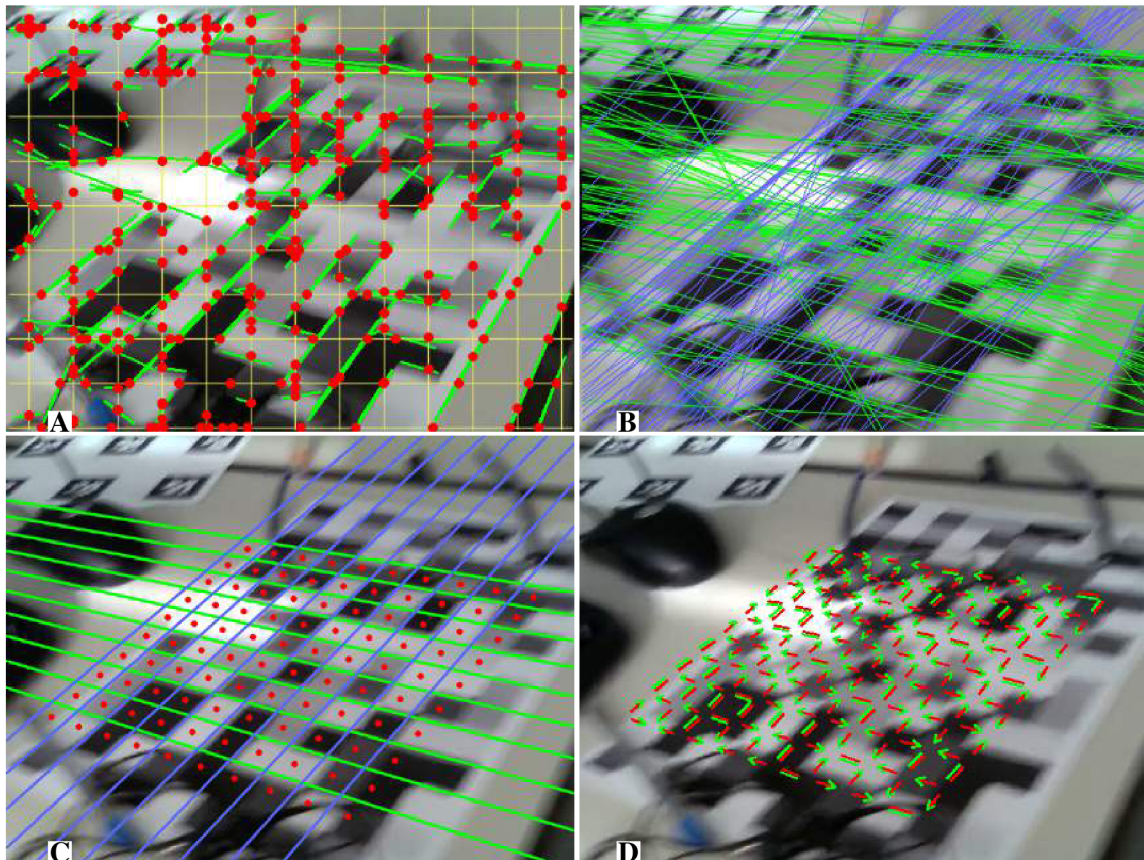


Figure 3.14: Detection of the greyscale grid of squares. **A:** The image is processed in sparse scanlines. On each scanline, edges are detected (Red) and extended to edgels (Green) by iteratively finding further edge pixels in the direction perpendicular to the gradient. **B:** The edgels are grouped into two dominant groups using RANSAC; two vanishing points are computed by hyperplane fitting. **C:** Based on the vanishing points, the optimal grid is fitted to the set of the edgels (orange dots denote the estimated centers of grid modules). **D:** Edges between the modules are classified (Section 3.4.2.2). Note that despite the blur, lighting and occlusion in the image, the camera is localized correctly (Figure 3.17).

The Uniform Marker Fields detection algorithm performs the following three main steps to detect the grid (Figure 3.14). The steps correspond to the steps 1 – 3 described in Section 3.3. I describe the improvements for all steps and discuss the necessary changes to the algorithm due to the changed marker design.

**1. Extraction of edgels** (edge element or edge pixel) – typically, for the improved design the algorithm extracts more straight edge fragments in the whole image than for the binary variant. The exact number depends on a lot of variables: resolution, number of shades, scanline density, blurriness. The image is processed in sparse horizontal and vertical scanlines (Figure 3.14 A). Additionally, when a video input is being processed, the detected edgels are filtered based on the previous detected position of the marker field. In the tests we used a simple rectangular mask to filter out the edges outside the area corresponding to the previously detected marker field. This change is beneficial in case of more marker’s on-screen on other patterns in the image with strong lines.

**2. Determining two dominant vanishing points** among the edgels (Figure 3.14 B). This step mostly remained identical. Two groups are found using RANSAC. The lines  $\mathbf{l}_i$  in each group should be coincident with vanishing point  $\mathbf{v}$ :

$$\forall i : \mathbf{v} \cdot \mathbf{l}_i = 0. \quad (3.19)$$

The vanishing point is found as the smallest eigenvector of the correlation matrix

$$C = (\mathbf{l}_0 \dots \mathbf{l}_N)(\mathbf{l}_0 \dots \mathbf{l}_N)^T. \quad (3.20)$$

Since matrix  $C$  is  $3 \times 3$  and symmetric, eigendecomposition can be computed very efficiently. We added one additional step to filter out outliers inside each group using RANSAC-like approach before the vanishing point calculations. This change helped detection on frames with strong perspective distortion, where the slope distribution inside pencils was wide.

**3. Finding the grid of marker field edges** as two groups (*pencils*) of regularly repeated lines coincident with each vanishing point. As pointed out in Section 3.3.6, the estimation of  $k$  was not ideal in equation

$$\mathbf{l}_i = \hat{\mathbf{l}}_{base} + (ki + q) \hat{\mathbf{h}}, \quad (3.21)$$

where  $\mathbf{l}_{base}$  is an arbitrarily chosen base line through the vanishing point, different from the horizon [143]. Parameter  $k$  controls the line density and  $q$  determines the position of the first line.

Since grayscale markers provide longer and more reliable edgels, a more robust estimation can be used for  $k$ . In order to find  $k$  and  $q$ , the value of  $(ki + q)$  is calculated for every line (extended edgel) of the input group. These values are clustered by mean-shift and median difference between cluster candidates is used as initial estimate of  $k$ . (The mean-shift box kernel size in normalized image coordinates  $[0, 1]$  in our tests was  $w = 0.05$ .) Each cluster is assigned an index  $i$  based on the initial guess and then overall optimal  $k$  and  $q$  are found by linear regression (Figure 3.14 C, blue and green lines).

For simplicity, the algorithm description supposes that a significant portion of the input image is covered by the marker field. Additionally, steps 2 and 3 of the algorithm are conditionally applied on rectangular parts of the image (quarters, ninths); in high-resolution images, the marker field is thus found even if it covers an arbitrary fraction of the camera input.

Due to the design decisions for the generalized Uniform Marker Fields, the proposed algorithm after step 3 diverged significantly from the original algorithm described in Section 3.3. These changes are discussed in more detail below.

### 3.4.2.2 Edge Classification

When only a small fraction of the marker field is visible, it is crucial that the edge gradients (Equations (3.16) and (3.17)) are recognized correctly. Their recognition can be challenging due to motion blur, uneven lighting conditions, etc. (Figure 3.15).

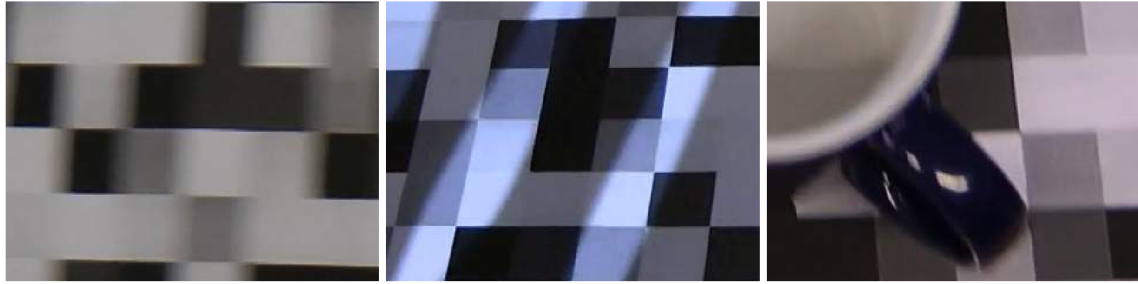


Figure 3.15: Examples of problematic edges within the pictures of the marker field (left to right: motion blur, dynamic lighting, occlusion).

In order to correctly classify an edge given the locations of the neighboring marker field modules, our algorithm samples pixels from the edge’s vicinity. If a small number of samples suffices to decide an edge either way ( $-1, +1$ ; see Section 3.4.1), the decision is made, otherwise more pixels are sampled. If an edge cannot be confirmed, the location between the modules is treated as a place without an edge:  $e_{ij}^* = 0$ . The stopping criterion is given by Wald’s sequential probability ratio test [173], which is proven to be the optimal sequential test for this purpose.

### 3.4.2.3 Localization Within the Marker Field

The sub-window described by edges  $G_{rc}$  is formulated as a vector of scalars in (3.18). This vector can be used as a key to a **hash table**. Values in the table represent locations in the marker field (two discrete coordinates in the terms of grid modules; enumerated orientation  $0^\circ / 90^\circ / 180^\circ / 270^\circ$ ). An absent record in the hash table means a wrongly recognized fragment of the marker field. Hash tables are implemented fairly efficiently in today’s programming languages.

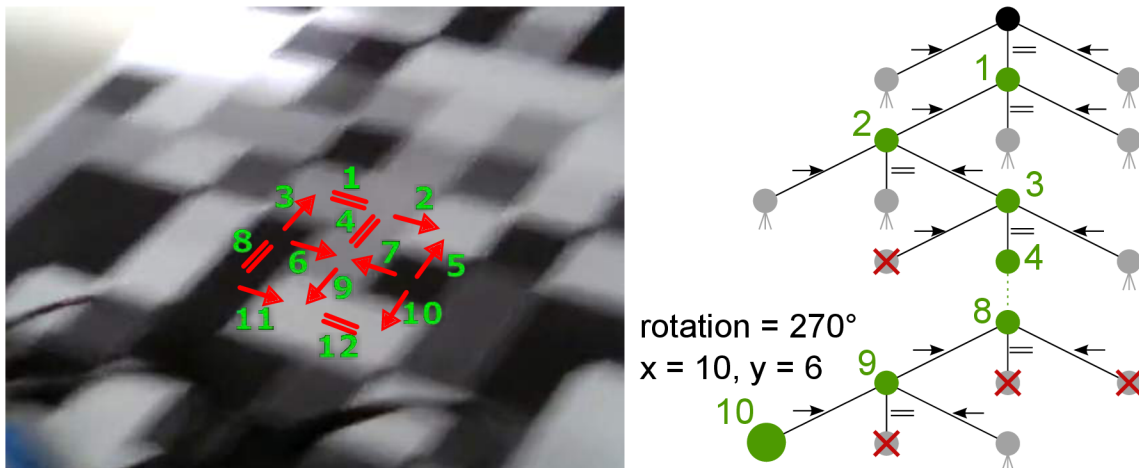


Figure 3.16: The decision tree used for localization in the marker field. **left:** A compact cluster of edges detected in the image. Edges are numbered in a predefined order. **right:** Decision tree – the leaves are either *invalid* or contain a location and orientation inside the marker field.

Instead of using a ready-made hash table, we prefer to create a **decision tree**. When a compact piece of the marker field is detected in an input image, the edges are classified and used for traversing the tree. A central edge in the detected cluster of edges decides the root node, and surrounding edges follow in a predefined order (Figure 3.16). Any cluster of neighboring edges is recognized by the

tree – the leaf node would either define the cluster’s location and orientation within the marker field or reject the cluster of edges as invalid (due to misdetection). Constructing a deeper tree implies that a larger cluster of edges is used for localization within the field. This allows for larger marker field resolutions. By using a larger number of deciding edges, the tree can also be constructed **fault-tolerant** – the tree nodes can tolerate one or more falsely classified edges.

#### 3.4.2.4 Corner Search and Iterative Refinement

For a precise camera pose estimation the algorithm finds all possible corners between the square modules in the marker field (with a sub-pixel precision). The corners of the grid of squares are localized using the detected overall position and grid parametrization. Their precise location in the image is iteratively refined. Based on the marker field’s layout, the algorithm knows each corner’s appearance including its rotation and searches for such a particular pattern. This helps mostly in cases when the image is motion blurred, the marker is not perfectly planar, or noise in the edgel data cause the grid not to fit the edges precisely.



Figure 3.17: Result with grayscale Uniform Marker Field. Marker Fields with clutter, occlusion, varying lighting, strong blur. **left:** Original image – input to the recognizer. **right:** Recognized camera location and augmented scene.

Another way of improving the precision of the camera pose estimation accuracy is to iteratively search for correct corners outside the detected region in the marker field in the image space using back-projection. In the tests we use both of the aforementioned improvements. A demonstration of improvements and robustness can be seen Figure 3.17.

### 3.4.3 Experimental Results

We compared our solution to ALVAR [2] as the most mature available ARToolKit follower (ARTag is no longer publicly available). ALVAR supports arrays of disjointed square markers. The other baseline is the Random Dot Markers (RDM) [166] as an alternative “marker field” solution, where individual localization markers overlap in the field and exhibit robustness against occlusion (Section 2.3.1). We performed identical experiments with CALTag [8] as well, but its results were always worse than ALVAR and it is much slower (written in Matlab), so I omit CALTag’s results from this work.

For comparing our solution with the alternatives, we shot videos of side-by-side markers (Figure 3.18). The marker fields have comparable (as much the same as possible) dimensions and

resolution of the individual markers ( $n^2$ -windows vs ALVAR individual markers vs RDM’s sub-markers) and the movement is simple and well-defined to ensure fairness in the comparison (see Figure 3.21 for example frames from the videos).

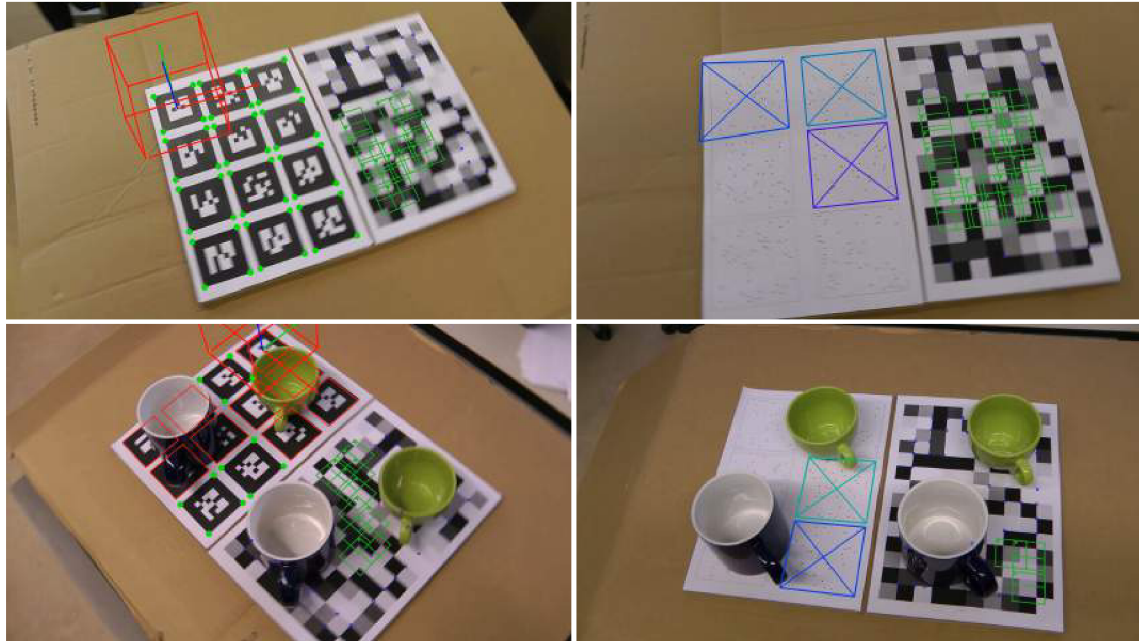


Figure 3.18: Illustrative frames of the side-by-side comparison video. **left:** Greyscale UMF vs. ALVAR. **right:** UMF vs. RDM. **top:** Motion blur tests. **bottom:** Occlusion tests. Videos were recorded in 1080p, capturing different classes of movement: zig-zag movement, upright rotation, rotation with severe perspective distortion, near/far movement, variable lighting conditions with fixed camera and general movement with occlusion.

Method	RDM	ALVAR	UMF
Average position variance:	8.5 cm	3.48 cm	3.28 cm
Average rotation variance:	0.049	0.035	0.024

Table 3.5: The average local variance in position and rotation change using 10 frames for averaging in a 1080p 50FPS video. The rotation variance is expressed as variance of quaternions, since the euler angles are unstable due to the gimbal lock. (Note: RDM gave highly unstable results and the low average variance in rotation is caused mainly by the low detection rate. For the *rotation* test video it gave 0.080 variance.)

### 3.4.3.1 Success Rate and Precision

Figure 3.20 shows the estimated camera pose in graphs for different videos for ALVAR and UMF. In order to evaluate the precision of our algorithm, we used the local variance (in time domain) of the estimated camera pose (position and rotation) – see Table 3.5. Low local variance means that the results of camera localization are smooth. We did not include RDM, since its stability was notably worse (Table 3.6) and ALVAR thus serves as a good reference for precision evaluation. Our method gave smoother results thanks to the good spatial distribution of matched points between 2D and 3D. ALVAR was also unable to find the corners of the individual markers precisely in blurred

images and for partially occluded corners. The number of detected corners used for the camera pose estimation is shown in Figure 3.19.

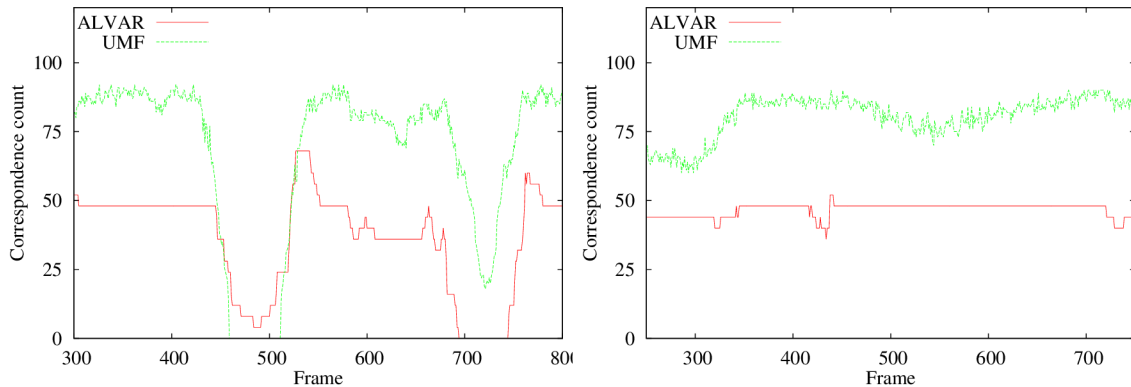


Figure 3.19: Number of correspondences used by ALVAR (red) and UMF (green) for the camera pose estimation. **left:** near-far video, **right:** perspective (refer to Figure 3.20). More points naturally mean a more stable and precise camera pose estimation and better tolerance to wrongly detected points (caused by a motion blur, partial occlusion, etc.).

Apart from the precision I also show in Table 3.6 the success rate for each method in every category of videos. (A frame is successfully detected, when the marker was successfully detected and a camera pose was estimated.) Random Dot Markers had the lowest performance, mostly due to their high sensitivity to motion blur. Even for a fixed camera or rotation with minimal blurring, the RDM detection algorithm gave the worst results. ALVAR and UMF were both very successful and gave very similar results. The only major difference was for zooming and occlusion. Figure 3.18 shows the clear advantage of our continuous marker field over ALVAR. ALVAR only detected two completely visible markers plus one, which had one edge slightly occluded. On the contrary, our method was able to detect sub-markers and corner points even between the cups.

Method	RDM	ALVAR	UMF
Lighting	89.7	100.0	100.0
Perspective	42.7	100.0	100.0
Near/Far	75.8	91.3	93.4, 94.6
Rotate	94.7	100.0	100.0
Zig-Zag	29.6	98.3	97.5, 97.4
Occlusion	38.5	93.0	94.0, 96.5
<b>Overall</b>	<b>61.8</b>	<b>97.1</b>	<b>97.8</b>

Table 3.6: Marker field detection success rates in %. For Uniform Marker Fields, rates from comparison videos with RDM and ALVAR are given separately, if different. Success rate is the percental ratio of video frames where the different markers were successfully detected.

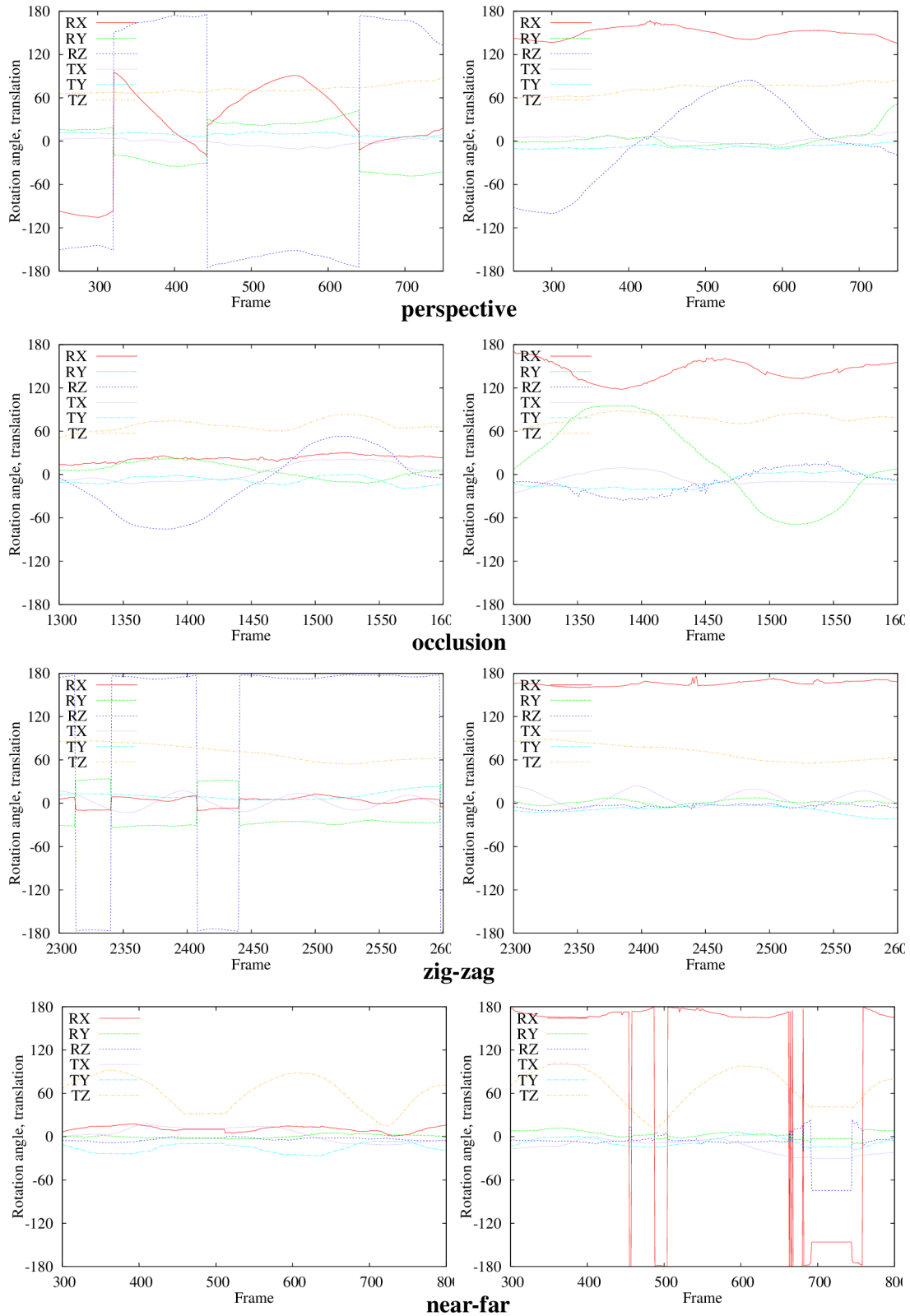


Figure 3.20: Graphs of estimated camera rotation (RX, RY, RZ) and translation (TX, TY, TZ) for representative videos. Smooth curves mean an error-free pose estimation; noise and “dents” in the curves indicate inaccuracies. 1<sup>st</sup> column UMF, 2<sup>nd</sup> column ALVAR. From top to bottom: perspective, occlusion, zig-zag, near-far. Major steps in the graph are caused by the gimbal lock and an angular over/underflow at  $-\pi$  and  $\pi$ .



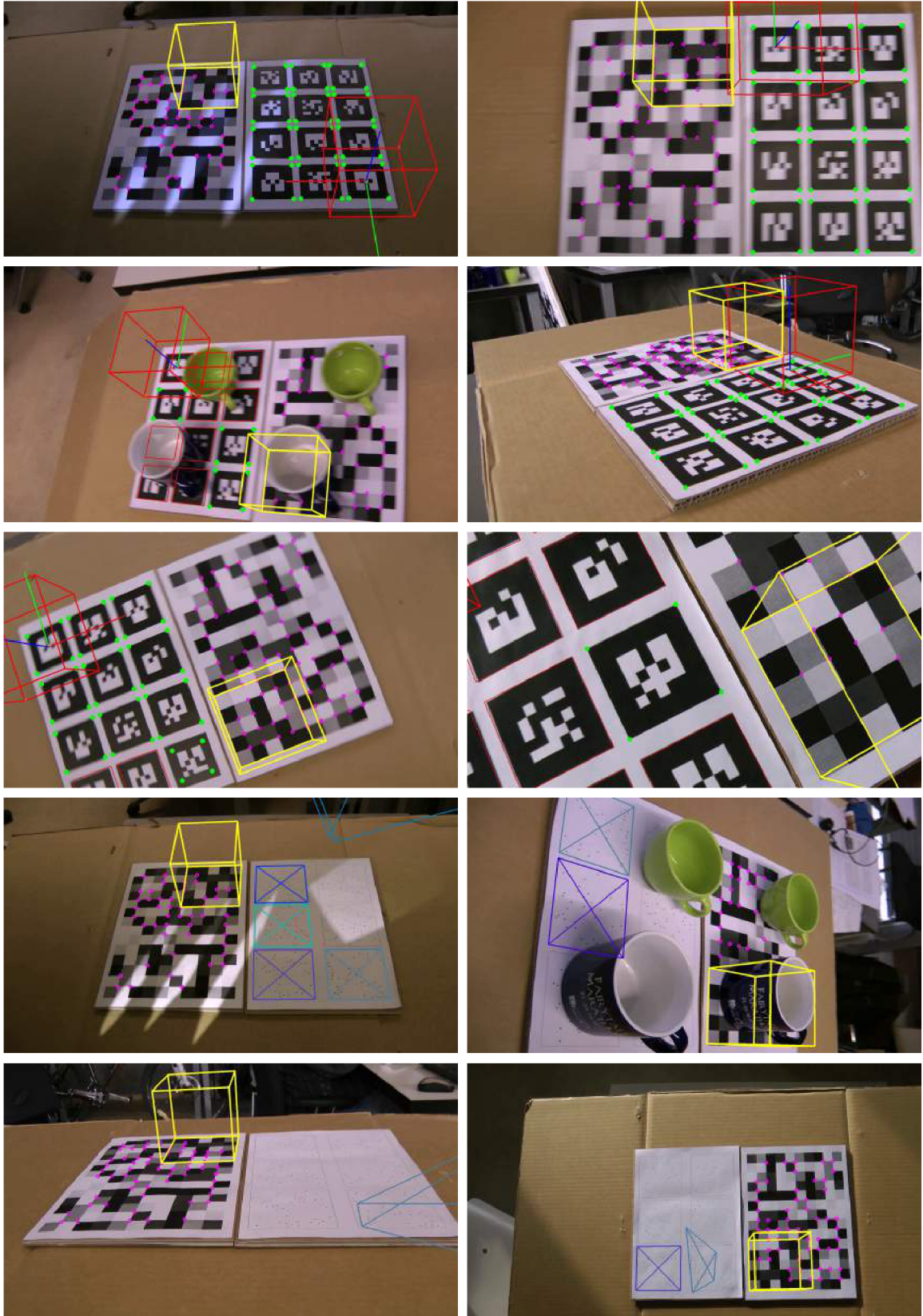


Figure 3.21: Sample images from the dataset. Purple dots are the detected corner points for UMF used for camera pose estimation.

### 3.4.3.2 Pixel Footprint and Computation Complexity

Table 3.7 shows the speed of the three tested algorithms and the breakdown of speed of our marker detection algorithm for videos with  $1920 \times 1080$  resolution. Our algorithm was more than  $3 \times$  faster than ALVAR and visited on average about 5.3 % of all pixel points. A small memory footprint is an important property for ultra-mobile processors where the memory accesses are slow due to limited caching.

<b>RDM</b>	<b>ALVAR</b>	<b>UMF</b>	(edge	grid	match	cam	sref)
164.4	30.1	8.8	(3.8	1.1	0.3	0.7	2.9)

Table 3.7: Breakdown of speed in milliseconds for 1080p videos using a mid-range Intel(R) Core(TM) i5 CPU 661 (3.33GHz) CPU. **edge**: edge detection in scanlines; **grid**: reconstructing the grid using RANSAC and vanishing point detection; **match**: edge direction detection and position decision making; **cam**: camera pose estimation based on the found matches; **sref**: processing in subwindows and position refinement by iterative search for more corner points.

### 3.4.4 Realistic Dimensions of the Marker Field

We used a cluster of computers ( $\sim 1000$  nodes) to synthesize the marker fields with highest possible resolutions. Table 3.8 shows several reasonable configurations of the marker fields. Marked in bold are the most conservative variants –  $3 \times 3$  sub-window, 3 colors. (*Note*  $\star$ : Substantially larger fields could be generated if necessary, but for practical reasons we capped the generation at this resolution.) Square marker fields are suitable for printing on paper. 16:9 marker fields are fine for panoramic canvases and monitors. Stripes are a good solution for scenes with large lateral movements. For example, stripes are good in case of localization inside buildings with long corridors or long moving shots in front of a greenscreen (Section 6).

Lower number of colors is better for assuring sufficient edge contrast. During our research we used 3 – 5 shades for grayscale (Sections 3.4.1 and Chapter 5) and greenscreen (Chapter 6) markers. Higher numbers can be afforded when more variable shades can be permitted: higher quality camera, good lighting conditions, large resolution, etc.










aspect ratio	window edges	# of colors/shades	MF dimensions	
1:1		3	<b>40×40</b>	
		4	110×110	
		5	175×175	
		3	175×175	
		4	250×250 *	
		5	250×250 *	
			3	250×250 *
			4	250×250 *
			5	250×250 *
16:9		3	<b>48×27</b>	
		4	144×81	
		5	192×108	
		3	160×90	
		4	320×180 *	
		5	320×180 *	
			3	320×180 *
			4	320×180 *
			5	320×180 *
stripe		3	<b>600×6, 150×15</b>	
		4	3000×6, 1200×15	
		5	3000×6, 1500×15 *	
		3	4000×8, 1200×20	
		4	8000×8, 2000×20 *	
		5	8000×8, 2000×20 *	
			3	8000×8, 2000×20 *
			4	8000×8, 2000×20 *
			5	8000×8, 2000×20 *

Table 3.8: Synthesized marker fields of reasonable parameters. The window edges shows the configuration of used edges for the window-array property. For each aspect ratio 3 different configurations are used: edges inside a  $3 \times 3$  window (12 edges), edges inside a  $4 \times 4$  window in a circle pattern (16 edges), and all edges inside a  $4 \times 4$  window. For practical reasons we capped the size of the marker fields, but larger fields could be generated if necessary (labeled as \*).

### 3.5 Follow-up and Related Research

The previous sections demonstrated that Uniform Marker Fields can be detected faster, more reliably and with similar or better precision as alternative marker based solutions. The simple and clean design has the potential to be used in many applications.

In this section I would like to summarize follow-up works – applications and improvements – based on Uniform Marker Fields. Follow-up works, where I contributed significantly, are discussed in more detail in Chapters 5 and 6. In this section I briefly describe publications, where I was not involved or my involvement was minimal. For more details refer to the individual works. These works demonstrate the extendability, reuseability and viability of Uniform Marker Fields as a marker field solution.

#### 3.5.1 Generalizations and Adaptations of UMF

A common assumption, when dealing with fiduciary markers, is their planarity. Even 3D markers often consist of several planar surfaces. Approximating more complex 3D forms for camera pose estimation is more difficult, since they have more degrees of freedom. Kriz [71] in his diploma thesis explored the possibilities of mapping Uniform Marker Fields on a cylinder (Figure 3.22). He approximated the ellipses formed by rings of the cylinder by a set of paraboles coincident with two points on the horizon. He was able to successfully detect grayscale Uniform Marker Fields in average in 70 *ms* with 96 % success rate. These results are promising and show a possible future work direction.

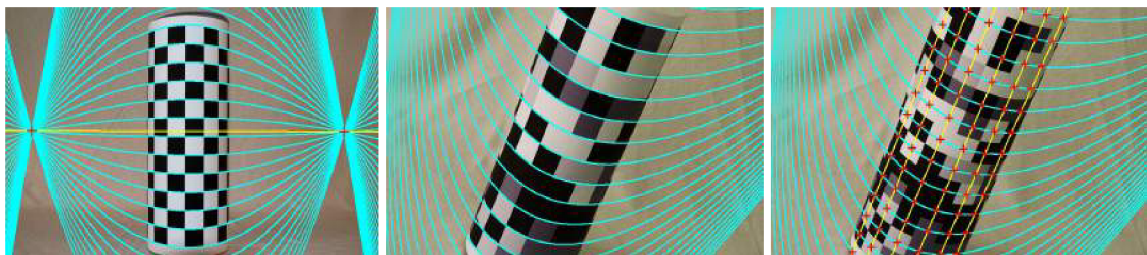


Figure 3.22: Detection algorithm for Uniform Marker Fields mapped on a cylinder [71]. Left: approximation of ellipses forming the rings on a cylinder with parabols coincident with two common points on the horizon. Center: detected pencil of parabols. Right: detected module centers on the cylinder.

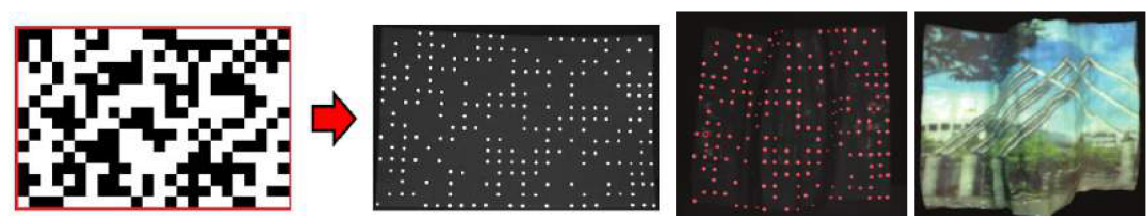


Figure 3.23: Modification of binary Uniform Marker Fields for deformable surfaces [47]. Left: the modification of the Uniform Marker Fields. Right: an example of the detected surface with the resulting mapped texture.

Deformable fiduciary markers or structured light can be used for surface reconstruction. In combination with a projector, deformable markers can be used to create a variant of Augmented

Reality system, where real objects are given virtual texture. Fujimoto et al. [47] used a modified variant of binary Uniform Marker Fields (Section 3.1) to create geometrically correct projection-based texture mapping on deformable surfaces. The deformable surface was overlaid with a thin  $0.2mm$  aluminium metal mesh, with a point based variant of Uniform Marker Fields (Figure 3.23) encoded on it visible under IR light. To identify the location of each marker point detected on the deformed surface, they used a simple voting system. It is important to note that they did not use the deformed marker for surface reconstruction, but only for texture mapping. Their current implementation did not work in real time – they were able to detect the deformed pattern on average in  $0.41 s$ .

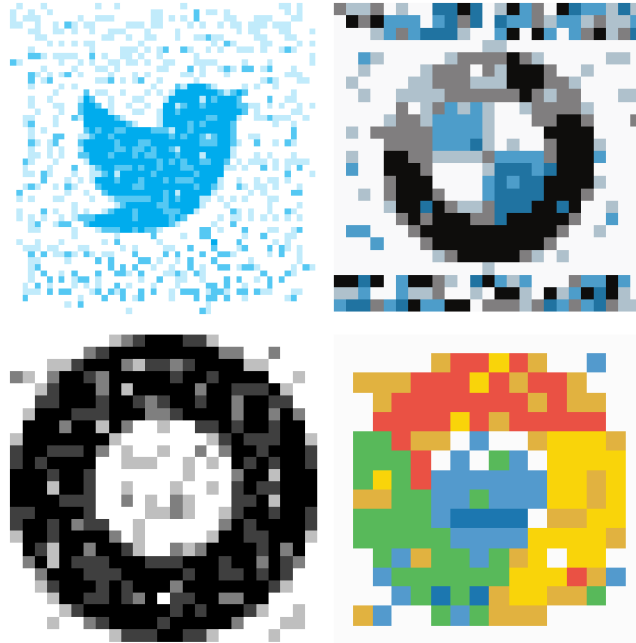


Figure 3.24: Generalized Uniform Marker Fields with image templates [128].

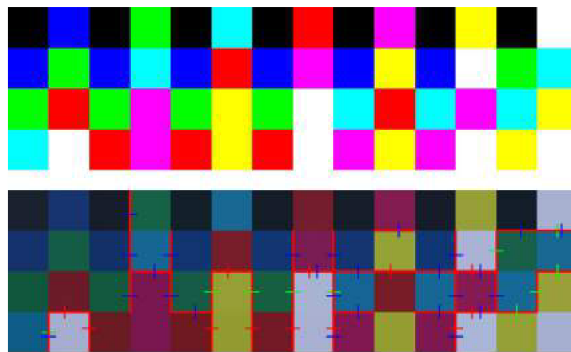


Figure 3.25: Empirical evaluation of color stability by Betko [17] for Uniform Marker Fields. The highlighted edges show false edge detections.

The generalization of binary Uniform Marker Fields to use colorful markers was further developed by Páldy et al. [128]. In his student paper he investigated the possibilities to optimize the generated Uniform Marker Fields using a cost function (Figure 3.24). He extended the marker design by generalized concept of window definitions and introduced a cost function to generate

optimal marker fields. The cost function involved maximizing edge difference for reliable lines, matching a template image, optimize color palette, etc.

A more detection rate-oriented investigation of Uniform Marker Fields with RGB colors was done by Betko [17]. He performed empirical tests on color and edge stability in RGB color channels between reference colors and camera shots of printed colors. Unfortunately, his tests primarily concerned mostly RGB and only partly YUV color channels. His results show that encoding Uniform Marker Fields into RGB color channels leads to unreliable detection. His investigation led me to explore alternative options of mapping Uniform Marker Fields into color information for greenscreen markers (Chapter 6), instead of using color channels directly.

### 3.5.2 Indoor Localization by UMF

We created a dataset of images to measure the precision of the Uniform Marker Fields detector in collaboration with M. Zachariáš et al. [184] as part of our research concerning indoor navigation. We marked 6 different view points relative to a projected grayscale Marker Field with  $14.3\text{cm}$  module size. We triangulated the reference point relative to 3 fixed points on the ground. The precision of our ground truth measurement approach had  $0.193\text{ cm}$  standard deviation.

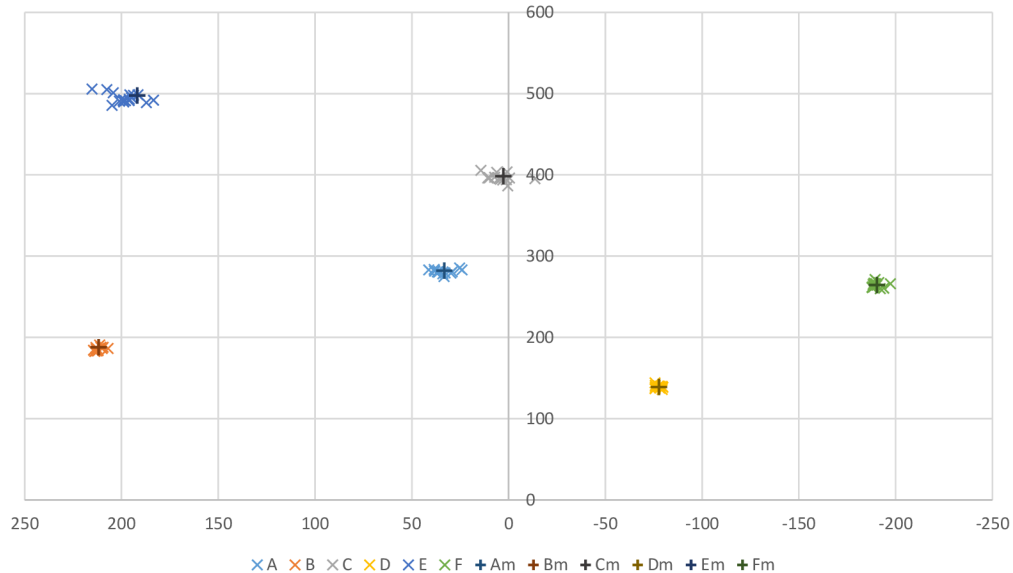


Figure 3.26: Detected positions from the (15+ photos for each test point) -  $A, B, C, D, E, F$  points. The + points are the illustrative reference point positions -  $A_m, B_m, C_m, D_m, E_m, F_m$ .

We took images from the 6 reference points with a smartphone camera (Nokia Lumia 930) at  $1920 \times 1080$  px resolution. For each reference point we took 15+ photos of the scene at different angles and 3 different heights from the ground. Robotic arms have limited range and would not have covered the tested area. Instead, we used a plummet to achieve relatively high precision camera positioning. Another factor complicating the exact measurement is the camera rotation and determining the exact center of projection. These complications cause a small, but not dismissible noise to the detected positions. We also calibrated the smartphone camera for maximal precision.

Figure 3.26 shows the results. The ground truth points in Figure 3.26 are for illustration purposes only. The absolute distance of the detected points from the ground truth might contain a bias. One of the sources of bias is the possibility of slight axis misalignment (rotation) between the reference system of our ground truth position relative to the fixed objects in the scene and the space of the

marker. Another source of misalignment could be the camera calibration and/or mis-detections. To achieve reliable and unbiased results, we compared the relative distances between pairs of detected positions.

The overall standard deviation of the distances between pairs of detected positions from the ground truth was 6.73 cm and the median 3.21 cm. It's important to note that the precision varies based on the distance from the fiduciary marker plane. With growing distance, the limited camera resolution and possibly imperfect camera calibration cause loss of camera pose precision. This effect is also visible in Figure 3.26. The measurement jitter is visibly increased for point *E* which observes the marker field from a large distance.

This concludes the design, testing and evaluation of the base algorithm for Uniform Marker Fields. In the following sections, I demonstrate the viability of Uniform Marker Fields for device-to-device task migration (Chapter 5) and for camera pose estimation as part of a greenscreen (Chapter 6). The next section deals with fast grid pattern localization that can also be used to efficiently localize candidate positions of Uniform Marker Fields in high resolution images.





## Chapter 4

# Fast Detection of Grids and Matrix Codes

The previous chapter described the design and detection for Uniform Marker Fields marker solution. One important assumption during the design of UMF and its detection algorithm is that it covers a significant portion of the input image. For large scenes, this assumption is unrealistic. In this chapter, I propose an efficient search for candidate positions for regular grid patterns. While I demonstrate this approach for detecting QR codes in high resolution images, the algorithm is not limited to any specific marker or Matrix Code.

Fiduciary markers (ARToolKit, ARTag, Fractal Marker Fields, etc.) and visual patterns for encoding information (QR code, Data Matrix, Aztec Code, etc.) often exhibit distinctive checkerboard structure. In the case of data codes, this serves multiple purposes. Firstly they are a straightforward way to encode binary information in 2D and secondly they are easily noticeable by users and machines. The “manhattanic”, mostly black-and-white structures stand out in most scenes. There is great potential in using this definite structure to find regions of interest in the image containing markers and data codes.

Traditional fiduciary markers rely on strong edges and segmentation to localize the markers. This has two major disadvantages: they lack robustness against occlusion and the segmentation process is computationally costly. The advantage of these marker solutions is that their size is relatively small and they can represent multiple separate virtual objects in the scene (for example a virtual canvas and a virtual pencil used to draw on the canvas).

Uniform Marker Fields described in Section 3 rely on two clusters of long edgels to localize the marker. This approach has the advantages of being robust against occlusion and it is possible to cover large areas in the scene. A disadvantage of this (global) approach is that the localization method has no direct support for multiple markers in the scene. This problem can be partially solved by processing the edgels in sub-regions of the image and discard parts, where no suitable pattern was found.

We presented an alternative, more robust approach to finding checkerboard structures in collaboration with A. Herout and M. Dubska [158]. In our research, we focused on QR-codes. My main contribution was the efficient algorithm to find possible occurrences of checkerboard structures and testing. The description below follows closely our publication.

## 4.1 Motivation

QR codes [3] are a very popular case of matrix codes (or 2D barcodes). They are receiving an increasing popularity among smartphone users and are becoming the standard when it comes to short data migration into their devices. The QR codes themselves or similar matrix codes (Aztec, DataMatrix, etc.) are being routinely used by different industries to carry meta-information attached to real-world objects. These matrix codes are designed so that they would be detected in images where they cover a significant portion of the image and where they are not deformed by perspective projection. However, their detection in high-resolution images of real-world complex scenes is desirable. These views would involve rotation, perspective deformation, and other distortions of the codes.



Figure 4.1: Examples of QR codes in high resolution images distorted by perspective transformation.

When the QR code covers a significant portion of the input image ( $\approx 25\%$  or more) and only one code appears in the image, the situation is simplified and algorithms exist which solve this efficiently [4, 36]. However, in complex scenes with more than one QR code skewed by perspective projection and rotation (see Figure 4.1 for an example), the process of detection and recognition of the codes is different. Alfthan [4] observes that for “just” detecting the code, a lower-quality image and other means of processing can be used – compared to recognition of the code. This may lead to two distinct algorithms – an algorithm for detection of the codes in a complex high-resolution image, and an algorithm for recognition of individual codes in the identified sub-images.

Some researchers keep relying on the *Finder Patterns* embedded in each QR code (however absent in other codes, such as the DataMatrix) [18, 151, 112] to localize the markers in high-resolution images. Others detect the code as a whole – by using texture analysis [62], morphological operators [7] and other techniques [68]. Most of these approaches are computationally very demanding (they would use the Canny edge detector, several passes by a morphological operator, convolution with large kernels, and similar techniques). Even the algorithms targeting real-time operation are not quite usable on images with high resolution. For example, a recent solution by Belussi and Hirata [18] takes 50-150 ms to process a  $640 \times 480$  image in one pass and several passes are needed.

I propose in the following sections our method for QR code detection in complex scenes based on Histograms of Oriented Gradients (HOG) [31] and segmentation of image parts based on HOG. Our solution is computationally very efficient – only a fraction of image pixels needs to be visited in order to detect the QR codes in a high-resolution image. Its byproduct are the evaluated HOGs for the individual detected QR codes; they can be used by a published algorithm by Dubská et al. [36] for matrix code recognition, sparing additional computational resources.

## 4.2 QR code Detection

The QR code was designed in such a way, so that it can be easily localized by finding the predefined structures at its three main corners (see Figure 4.2). There have been many attempts to speed up and improve the detection of the QR codes the way it was intended using the *Finder Patterns* (FIP).

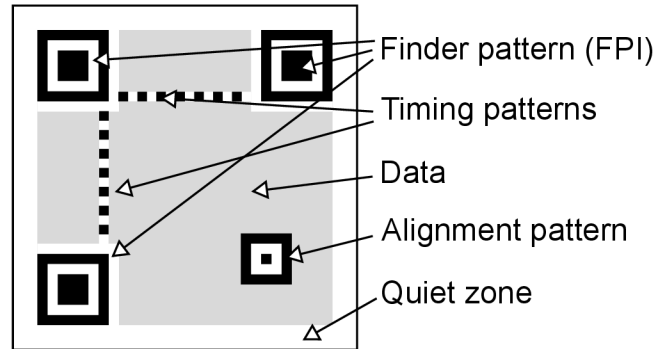


Figure 4.2: The structure of the QR code.

Ohbuchi et al. [112] used scanlines on a preprocessed image to find FIPs and then applied an inverse perspective transformation for image reorganization to extract the binary information enclosed in the QR code. Sun et al. [151] used a similar approach, but for locating the FIPs they used the Canny edge detector and looked for square contours.

Choosing the correct threshold to binarize the image and find FIPs depends among others on the properties of the camera and illumination. Liu et al. [87] used multilevel threshold to deal with varying lighting conditions for QR code detection.

Given the regular structures of QR codes and matrix codes in general, there has been research on using the Hough transformation for matrix code or barcode detection and localization. Muniz et al. [104] presented a robust method for decoding linear barcodes using the Hough transform. Wang and Zou [174] used the Hough transform to locate the four corners of the 2D bar code region and used the second derivative image in combination with bidirectional centripetal run-length to decode the bar code. Parikh and Jancke [121] also used the Hough transform. They combined it with adaptive threshold and texture analysis to locate the correct position of 2D high capacity color barcodes and to decode them.

Dubská et al. [36] introduced an approach, that efficiently searches for straight lines that coincide with two dominant vanishing points by using a variant of Hough transform (PC-lines) and a line parameterization which is a point-to-line mapping. They assume that the image contains a single QR code, which covers a large portion of the image. This method was used during our research for experiments and evaluations.

The matrix codes (including the QR codes) have a very specific texture characteristic. Thus the problem of localization of the QR code is similar to image segmentation based on the texture features and then selection of a segment with requested characteristics. Methods of texture based segmentation vary in used textural descriptors such as the Local Binary Patterns [113], Gabor features [175] or Markov random field statistic [52].

Ouaviani et al. [119] presented an image processing framework exploiting the regular structure of 2D barcodes. First they used the histogram of gradients to detect regions of interest in the image. To get the precise area of the code, they segmented out the region containing the code based on an average gray-level of pixels in a small neighborhood using region growing and found the smallest 4-sided polygons surrounding the segmented code.

### 4.3 Rapid Detection of Regions with a Possible QR code

During our research, we focused on relaxing the constraint of having exactly one 2D barcode in the image and consequently allowing that barcodes cover only a small part of the image (see Figure 4.3 for an example). We proposed an algorithm that can efficiently localize and extract candidate positions for the 2D barcodes even in high-resolution images. In order to achieve this, we exploited the property of 2D barcodes of having a regular distribution of edge gradients to get probable positions for full QR code detection.



Figure 4.3: The original 15MPix image. Multiple codes are present in the image; they are subject to perspective projection; dominant edges and text appear in the image.

#### 4.3.1 Processing in Tiles

A naive way of finding the 2D barcodes without using special properties of specific barcode types in a high resolution image at every reasonable scale would be to scan through the image at each scale with a sliding window of a given size and run the detection algorithm (such as the one described in [36]) at each position. This would require too much computational time.

There are some common properties of 2D barcodes, however, that could be potentially exploited to help localizing them. Two most important of these are the regular distribution of edge lines and a characteristic distribution of the gradient directions.

We proposed separating the image into a regular grid of tiles. Each tile could serve as an indicator for a probability of being part of a 2D barcode. Let  $\mathcal{T}(u, v)$  be a tile at index  $u, v \in \mathbb{N}_0$ . The tile then corresponds to a square image region with top-left corner  $(uw, vw)$ , where  $w$  is the width of the tile (all tiles share the same size). Figure 4.4 shows an example of such a grid. The overlaid green intensity represents the probability of the tile being a part of a QR code.

##### 4.3.1.1 Extraction of the Histogram of Oriented Gradients

To get an indicator for QR code presence, the algorithm first extracts the Histogram of Oriented Gradients (HOG) for each tile. For each pixel, the algorithm computes the size of the gradient: both its magnitude and orientation. Only gradients whose magnitude exceeds a given conservative threshold are processed. The threshold is set fairly low so that a reasonable amount of blur does

not spoil the detection. The stored edge pixels are used to create the HOG for the tile. We used a histogram with a low number of bins (e.g. 12 bins); higher histogram resolution is futile due to the limitations of gradient direction precision computed from a small neighborhood of pixels. The resulting histogram is a vector for each tile:  $\mathbf{h} = \mathbf{H}_n(\mathcal{T}(u, v))$ , where  $n$  is the number of bins. See Figure 4.4 for HOG representation. The highlighted red and blue bins of the HOGs are the detected two dominant edge orientations.

To speed up the process, the histograms can be approximated by taking only pixels along scanlines vertically and horizontally separated by a given number of pixels. By processing only the scanlines, only a small percentage of input image pixels have to be processed for each tile, but still have a precise-enough approximation of the HOG.

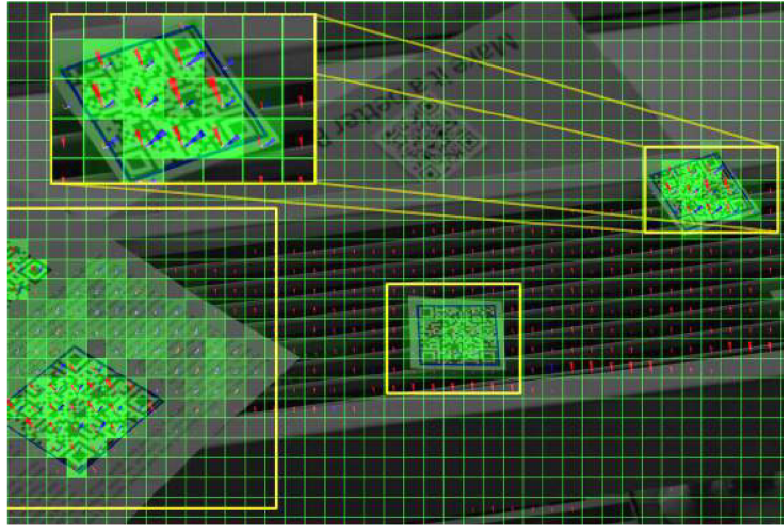


Figure 4.4: The grid with histograms of oriented gradients.

#### 4.3.1.2 Feature Vector Extraction

In addition to the gradient histogram, a feature vector is computed for each tile. It contains the normalized histogram and four additional values: corresponding angles of the two main gradient directions in the histogram, the number of edge pixels per unit area and an estimation of the probability score of a chessboard like structure in the tile based on the histogram.

Let then  $\mathbf{v} = (p, \alpha_1, \alpha_2, \frac{N_e}{A}, \mathbf{h}_{norm})$  be the feature vector for a tile with normalized histogram  $\mathbf{h}_{norm}$ , pixel area  $A$ , and the number of edge pixels over a threshold  $N_e$ . The probability score is computed in the following way:

$$p = \left( 1 - \frac{||\alpha_1 - \alpha_2| - \frac{\pi}{2}|}{\frac{\pi}{2}} \right) \frac{2 \min(\mathbf{h}_a, \mathbf{h}_b)}{\mathbf{h}_a + \mathbf{h}_b}, \quad (4.1)$$

where  $\mathbf{h}_a$  and  $\mathbf{h}_b$  are the values in the histogram corresponding to the two dominant edge orientations. The first term ensures that the two dominant directions are approximately  $\frac{\pi}{2}$  apart; the second term prefers equally high peaks in the histogram.

The two main gradient directions detected from the HOG  $\alpha_1$  and  $\alpha_2$  meet an additional constraint that  $|\alpha_1 - \frac{\pi}{2}| \ll |\alpha_2 - \frac{\pi}{2}|$  or  $\alpha_1 < \alpha_2$ . The order of the angles is important for the segmentation.

### 4.3.1.3 Segmentation of the Tiles

This vector of features can be used for segmentation of the tiles. For segmentation we used a simple 4-neighborhood blob coloring [139], which requires two passes for each level in the hierarchy of tiles.

We used only the first angle  $\alpha_1$ , the probability score  $p$  and the edge density  $\frac{N_e}{A}$  for segmentation for performance reasons. The segmentation is done on a relatively small number of elements, so it does not introduce any major computational overhead.

The result of the segmentation (Figure 4.5) is a set of connected tiles  $\{S_1, S_2, \dots, S_k\}, k \in \mathbb{N}$ . If  $A(S_k)$  is the area of the axis-aligned bounding box in  $pixel^2$  and then let the probability score that the segment is a possible position of a QR code be:

$$P(S_i) = \frac{1}{A(S_i)} \sum_{\mathcal{T} \in S_i} p \frac{N_e}{A}, \quad i \in \{0, \dots, k\} \quad (4.2)$$

where  $p$  and  $\frac{N_e}{A}$  are values from the feature vector for tile  $\mathcal{T}$ .

### 4.3.1.4 Segment Classification

Based on the computed score  $P(S_i)$  and the shape of a set of tiles  $S_i$ , we use a binary classifier to determine whether the area covered by a given segment should be further processed to find QR codes:

$$C(S_i) = \begin{cases} 1 & \text{if } P(S_i) \geq T \text{ (is a possible position)} \\ 0 & \text{otherwise (not a probable position)} \end{cases}, \quad (4.3)$$

where  $T$  is an experimentally acquired threshold. The regions classified as probable positions of a 2D barcode in our example in Figure 4.4 are represented by thick yellow rectangles.

An example of the segmentation is shown in Figure 4.5. The edge length of the tiles in this case were  $w = 120$  pixels for a  $15MPix$  image. Two of four QR codes were precisely localized. The segment corresponding to the third, bigger one on the left side of the image contains also some parts of the text, since at this level of tiling it has properties similar to the QR code itself. The fact that the QR code candidate rectangle does not tightly fit the actual code edges, does not prevent it from being detectable by the detection/recognition algorithm. The fourth blurred QR code in the background did not get localized due to the small number of edge pixels and the near uniform distribution of the gradient directions.

## 4.3.2 Multiscale Processing

The size of the tiles can have a major effect on the success of finding correct candidate positions. Choosing too large region size might cause overlooking small matrix codes. For example, when a small QR code surrounded by large noise covers only a small part of four neighboring tiles, the probability score of a regular structure in the four tiles would be very small. On the other hand, choosing too small tile size  $w$  would cover only a small part of a large QR code and the segmentation would fail to group the tiles covered by the QR code.

We proposed using a quad-tree of tiles  $\mathcal{T}_l(u, v)$ , where  $l$  is the level. The HOG and edge extraction have to be computed only at the lowest level  $l = 1$ . The histogram of the tiles at a higher level in the hierarchy can be computed simply by accumulating the corresponding bins of the histograms of the child tiles:

$$\mathbf{H}_n(\mathcal{T}_l(u, v)) = \sum_{i=0}^1 \sum_{j=0}^1 \mathbf{H}_n(\mathcal{T}_{l-1}(2u + i, 2v + j)). \quad (4.4)$$



Figure 4.5: The segments used for finding the QR code. Red color component encodes the edge count per tile (inverted). Green color component represents the angle of the first dominant direction ( $\alpha_1$ ). Blue color component indicates the absence of the QR code based on  $p$  (refer to Equation (4.1)).

The segmentation and classification of the segments can be done at each level (or alternatively, a hierarchical segmentation can be used). Since the gradients and edge pixels are not recomputed for higher levels and the number of segments is relatively low, so this does not mean a significant computational overhead.

The whole algorithm for detection of QR codes in a high-resolution image is depicted in Algorithm 1.

---

**Algorithm 1** QR code detection in high-resolution images

---

**Input:** Image  $I$

**Output:** Detected QR codes

- 1: compute  $\mathbf{H}_n(\mathcal{T}_1(u, v))$  by edge extraction
  - 2: compute  $\mathbf{H}_n(\mathcal{T}_l(u, v)), l \in \{2, \dots, l_{max}\}$  from lower-level histograms (4.4)
  - 3: **for all**  $l \in \{1, \dots, l_{max}\}$  **do**
  - 4:   compute feature vectors  $\mathbf{v}_l(u, v)$  from the histograms
  - 5:   compute the segments  $\mathcal{S} = \{S_1, S_2, \dots, S_k\}, k \in \mathbb{N}$
  - 6:   **for all**  $S_i \in \mathcal{S}$  **do**
  - 7:     compute segment probability  $P(S_i)$  (4.2)
  - 8:     **if**  $C(S_i) == 1$  **then**
  - 9:       **run** QR code detection algorithm
  - 10:    **end if**
  - 11:   **end for**
  - 12: **end for**
- 

## 4.4 Experimental Results

In order to evaluate the performance, we collected a dataset of challenging real-life images. Since no standard dataset was publicly available for evaluation of QR code detection algorithms, we acquired

the images ourselves<sup>1</sup>. The images in the dataset were taken both with a cell-phone camera and a professional high-definition camera. The resolution of the images scales from 1 MPix to 15 MPix. The cell-phone camera had a maximal resolution of 5 MPix, so larger images were all taken with the professional camera.

The images in the dataset contain several different QR codes. All images contain at least one QR code, the maximal number of codes in a scene was 14. The percentual image coverage of the codes varies from 2 % to 70 %. The position and size of the codes for all images was annotated. Additionally we also annotated some further properties/flags for the codes: rotation, perspective deformation, varying lighting conditions and blur.

We divided the evaluation into two parts. In the first part we evaluated only the performance of the candidate position search using our hierarchical grid. In the second part we evaluated the performance of the whole algorithm including the 2D barcode detection method described in [36].

#### 4.4.1 Candidate Position Search

In order to evaluate the efficiency and precision of the candidate search with our hierarchical grid, we used *false positive* and *false negative* metrics. They are defined in this case as follows:

$$\#FP = \frac{\#correct\ candidates}{\#all\ candidate\ positions}, \quad (4.5)$$

$$\#FN = \frac{\#missed\ positions}{\#all\ annotated\ positions}. \quad (4.6)$$

A *correct candidate* is a candidate position, where an annotated QR code lies completely inside the candidate image region and covers at least 30 % of it. We also experimented with expanding the candidate image regions by 0 %, 20 %, 40 % and 60 %, since the segmented image region might underestimate the QR code's size.

grow by	FP	FN	DT
0 %	58.9 %	12.2 %	74 %
20 %	54.6 %	9.4 %	79 %
40 %	52.9 %	7.4 %	80 %
60 %	53.4 %	9.4 %	78 %

Table 4.1: Performance evaluation of the candidate search algorithm. FP: false positive percentage; FN: false negative percentage; DT: the percentage of QR code successfully detected in all images using [36].

Table 4.1 shows the results. Growing the candidate areas by 40 % proved to be the most robust option. This proved our hypothesis, that the segmentation underestimates the QR code size. By growing the candidate regions even further causes loss of performance, since the covered area by the QR codes became too small for the QR code detection algorithm [36]. For this challenging data set, results show acceptable false positive rate and low false negative rate.

#### 4.4.2 Detector Evaluation

In this section, I describe the evaluation of the overall precision of our method. We extended the method for evaluation proposed by Dubská et al. [36] to include information also on the localization

<sup>1</sup>Downloadable from [www.fit.vutbr.cz/research/groups/graph/download/qrcodes/](http://www.fit.vutbr.cz/research/groups/graph/download/qrcodes/)



performance. Since the QR code decoders are differently sensitive to the rotation, perspective deformation, blur and irregular illumination, the images are thus sorted into six categories:

- plain** the code is upright,
- rot** the code is rotated,
- pers** the code is upright but skewed by perspective,
- rot+pers** general orientation of the code,
- blur** general orientation of the blurred code,
- shadow** general orientation of the code with irregular illumination.

The algorithm is composed of localization, detection, and decoding of the codes. If the code is localized, the binary bitmap of the code's blocks (binary matrix) is extracted and compared to the ground-truth bitmap. Since the QR code is capable of error repair and a few percent of the code can be missed, correct detection is counted when at least 95 % (99 %, 100 %) of the code is correctly extracted. Finally, the QR code is decoded by a publicly available ZBar<sup>2</sup> library and compared to the original encoded string.

<b>type (count)</b>	<b>L</b>	<b>DT (100, 99, 95)</b>	<b>DC</b>	<b>ZB</b>
plain (53)	93.7	90.6 (89.6, 95.8, 100.0)	88.7	90.6
rot (54)	98.2	88.9 (93.8, 97.9, 100.0)	85.1	98.2
pers (19)	100.0	94.7 (55.6, 61.1, 88.9)	63.2	57.9
rot+pers (50)	96.0	88.0 (77.3, 88.6, 95.5)	78.0	60.0
shadow (26)	92.3	69.2 (33.3, 66.7, 94.4)	46.1	46.1
blur (51)	80.4	52.9 (66.7, 74.1, 92.6)	41.2	67.9
<b>overall (251)</b>	<b>92.8</b>	<b>80.5 (76.7, 86.1, 96.5)</b>	<b>70.1</b>	<b>74.8</b>

Table 4.2: The results are shown for candidate position enlarged by 40 % of the corresponding segment size. The values in the three columns relate to the number of all QR codes in the category. L: successfully localized; DT: successfully detected (the brackets show the percentual correctness of the extracted matrices); DC: successfully decoded using zbar; ZB: success rate achieved by zbar library independently of our algorithm.

Table 4.2 shows the success rate for each step of the algorithm. All the values (except the ones in brackets) relate to the overall number of codes in the category. The first column shows the number of correctly localized barcode positions using the first part of the algorithm; the second column the percentage of the barcodes for which a binary matrix was successfully extracted. The brackets contain the information about the precision of the extracted binary maps compared to the annotated ground truth data. The third column shows the percentage of the QR codes that were successfully decoded out of all the code instances in the dataset.

<sup>2</sup>zbar.sourceforge.net - we reused the QR code decoding part from the already extracted matrix

For example for the rotated QR codes 98.1 % of the codes were correctly localized. For 88.9 % percent of all annotated QR codes, the binary matrix was successfully extracted. 93.8 % of these were identical to the annotated data, 97.9 % of them were over 99 % correct and all the rest was over 95 % accuracy. Lastly 85 % of the QR codes in this category were correctly decoded.

The results show that the blur caused by out-of-focus shots and movement are the most disturbing phenomena for 2D barcode detection by our algorithms. Another problematic case are 2D barcodes with shadows across them, since our detection algorithm did not take into account the much weaker gradients on shadowed areas. This could be solved by separating the candidate area into smaller parts and taking an equal number of best edge pixels from each part. Another disturbing area in the case of shadows is the penumbra, where the adaptive thresholding fails to determine the binary information of the field.

One other area where the matrix code is not extracted precisely are the perspectively distorted QR codes. This might be caused either by the non-planarity of surfaces and the small resolution of the  $\mathcal{TS}$  space used during Hough transformation [36], since the maxima are not equally distributed along the vanishing line. To solve this issue, a re-fitting step could be added to detected QR code field centers similarly as was described by Parvu et al. [111].

We compared the performance and speed of our method with an open-source project for QR code detection: the ZBar library. It uses linear scans with adaptive thresholding to localize FIPs in the QR codes. We chose this implementation because it is popular, frequently maintained, it uses a completely different and theoretically fast approach, and also because it supports detecting multiple QR codes in one image.

In Table 4.2 the ZB column shows the success rates for the ZBar implementation for different QR code categories. Because ZBar does not use edge extraction, it is more successful decoding blurred images. On the other hand it does not cope well with perspectively distorted images in comparison to our method. The difference for detecting rotated QR codes might be caused by slight blur or small size of the codes, causing our algorithm to fail to determine the edge orientations precisely. This limitation might be solved with an extra sharpening step.

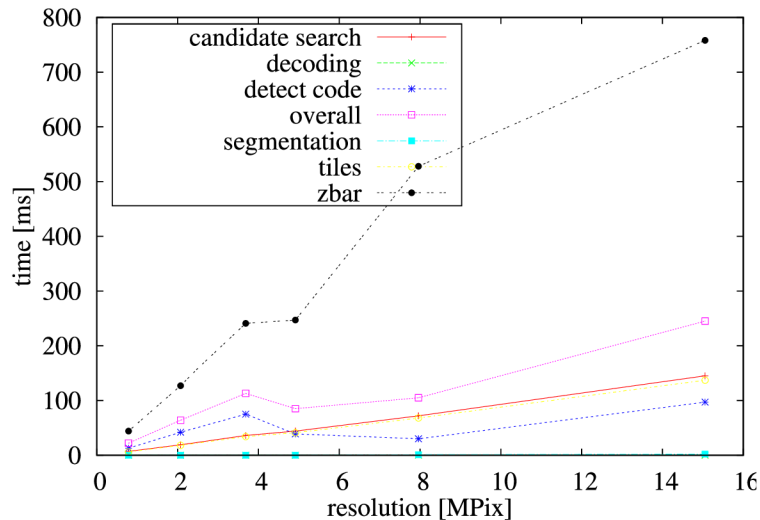


Figure 4.6: The required time for processing in ms. The graph also shows average required times for different parts of the algorithm: the overall time consists of the candidate search, code detection and binary matrix decoding; the code detection then from segmentation and the creation of the tile hierarchy.

### 4.4.3 Speed Evaluation

Figure 4.6 shows the average required time for the algorithm. It does not contain the system overhead for loading and saving the images. The tests were carried out on a mid-range Intel<sup>®</sup> Core<sup>™</sup> i5-2410M, 2.3 GHz processor using a single core. The graphs show that there is a linear relation between the image resolution and candidate search. The overall time, however, varies more thanks to the different amount of detected candidate positions. The drop at 5 MPix resolution can be explained by the difference between professional cameras and cameras integrated into mobile phones. Since most mobile phones apply sharpening filters to the images, the number of edge gradients grow accordingly. This may cause a higher number of candidate positions, making the detection/recognition part of the algorithm slower.

The main purpose of using the grid of tiles is to reduce the number of positions to search for the QR codes in the whole image. As seen in Figure 4.6, the code detection itself takes for high-resolution images in average less than half of the required time of the whole algorithm. As mentioned earlier, to reduce speed and computational requirements we also approximated the HOG creation for the grid by subsampling the scanlines. This speed-up is largely dependent on the distance of the scanlines. We used 10 pixel distance between consecutive scanlines. This means the algorithm had to process only approximately 1/5 of the image pixels plus the neighbors of the pixels where a gradient was detected to get the gradient direction.

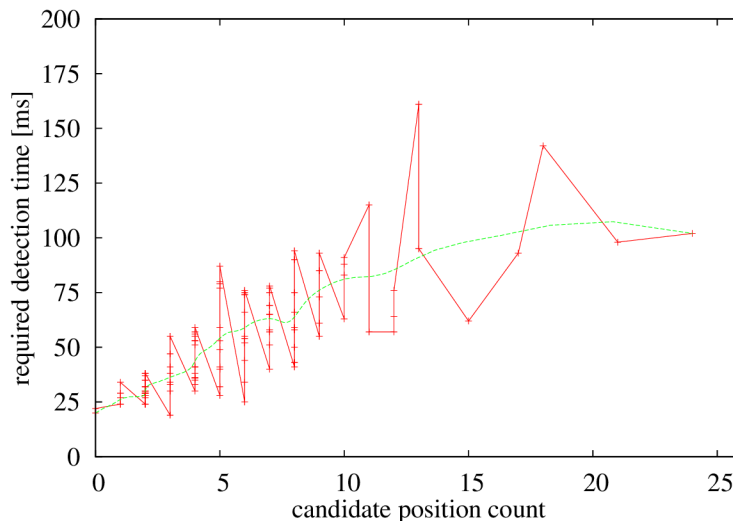


Figure 4.7: The required time for detecting QR codes in relation to the number of detected candidates in milliseconds for 1080p images.

From Figure 4.6 is also clear that the major slowing factors are the creation of the tile grid hierarchy and the number of candidate positions the detector is forced to process (see Figure 4.7).

The grid creation part basically consists of the gradient detection and feature vector calculations for each tile. The segmentation is only done on the grid level and it does not introduce a large computational overhead. However, the quality of the segmentation and the classification of the segments has a direct effect on the number of candidate positions and consecutively on the speed of the detection part.

We also tested the required time for resolution used by common HD cameras – 1080p. Figure 4.7 shows the results depending on the number of candidates positions that were found. The average time required to process one image was 54 ms and over 95 % of the images were processed under

100 *ms*. The outliers were caused mainly by extremely disturbing background texture and very high-resolution QR codes, where the number of candidates were relatively large.

Since we are able to detect for a large range of scales, the achieved 250 *ms* in average for 15 MPix images is sufficient for localization for a slow moving object equipped with a high-resolution camera. In order to get truly real-time speed, GPGPU solutions could be used for segmentation, classification and Hough transform.

Figure 4.6 shows also the speed of the detection with the ZBar library. Similar to the evaluation of our method, these results do not include the overhead for loading and saving images. The results show that the ZBar library is approximately 4 times slower for high resolution images with a comparable detection rate.

## 4.5 Conclusions

I described an algorithm for predetection of QR codes (and similar matrix codes, such as Aztec, DataMatrix, etc.). The algorithm is capable of detecting locations with probable occurrences of matrix codes in a high-resolution image. These locations can be further processed by an algorithm for detection and recognition of QR codes. The predetection algorithm and the detection/recognition one can share some computed information, further reducing the computational cost.

The algorithm was evaluated on a dataset of high-resolution images with one or more QR codes present in the scene. The codes are viewed from different point and at different scales; the dataset thus well represents real-life scenes. Our algorithm successfully localized 95 % of all QR codes. The required run-time was under 250 *ms* in average for the tested images (up to 15 MPix large).

The algorithm was fairly efficient and it notably exceeded the results published recently (e.g. [18]). The algorithm is very suitable for parallelization in a SIMD environment, and it should be fairly efficient when executed on recent GPUs. To evaluate the performance of the algorithm, we collected and annotated a dataset of real-life images, suitable for comparison of algorithms. Such a public dataset was missing.

In a follow-up work, Klimek and Vamossy [80] published an implementation written using Microsoft .NET library and demonstrated slightly better results on their own dataset. Our QR code dataset has been already used for evaluation in [48], [162], [161] and others.

## Chapter 5

# On-screen Markers

In the previous Chapter 3, I introduced the Uniform Marker Fields, described methods for its synthesis and especially its real-time and computationally cheap detection. It turned out that this technology enables certain domain-specific augmented reality applications. This chapter presents one of them, where the markers are placed onto a computer screen so that it can be detected and recognized by an ultramobile device (typically a smartphone). This work was done in collaboration with other colleagues (primarily Rudolf Kajan) who provided the user interaction expertise and the use-case itself, my role was to adapt and optimize the technology of the marker fields and to design and implement the mobile technological solution.

We used Uniform Marker Fields as an overlay on a large displays' content to achieve fast screen-to-screen task migration initially with binary markers [72]. This work involved research and development of an improved prototype using grayscale Uniform Marker Fields (Section 5.1). We took this research further and revised it into a full augmented reality experience for continuous inter-device interaction technique [73] (Section 5.2). The text of this chapter is based on two articles published in in a conference proceedings and in a journal [72, 73].

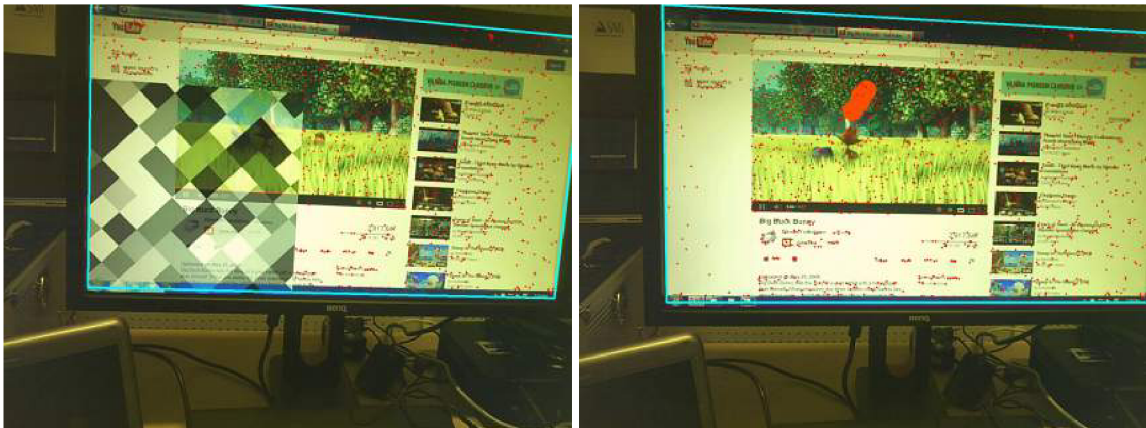


Figure 5.1: Uniform Marker Fields aiding eye-tracker localization [124]. **Left:** frame where marker is shown and was detected. **Right:** consecutive frame with hidden marker tracking feature points (red dots).

With the appearance of large and cheap high-resolution network-connected displays, and smartphones becoming a widespread personal accessory, the ubiquity of screen real-estate naturally drew the interest of many researchers to examine the possibilities for interaction between these devices. This research, though, is not limited only to smartphones – large display combination, but includes

any camera-equipped personal device like: eye-trackers, head-mounted see-through displays, binoculars, etc. These interactive systems can also be perceived as a form of augmented reality system: the mobile device localizes itself in 3D relative to the large display and uses this information to remotely interact and/or augment the users' view with virtual objects and extra information in real time.

Similar investigation into on-screen markers was done by Pavelková [124], who combined on-screen Uniform Marker Fields with an eye-tracker and natural feature tracking. During this research, my only contribution was the Uniform Marker Fields detector. I only mention it briefly, since this research direction has a great potential. She proposed a system for real-time tracking of the computer screen and estimating area of user's visual attention on the screen with the use of eye-tracking data from a head-mounted eye-tracker aided by a fragment of Uniform Marker Fields at monitor corners to establish reference key-frames. Between key-frames the proposed system used natural feature tracking (Figure 5.1). In order to make the marker as unobtrusive as possible, gaze information is used to display the marker as far away as possible from the area of user's visual attention.

## 5.1 Screen-to-Screen task migration using Uniform Marker Fields

An important element of computer-supported cooperative work is information sharing and task migration among users' devices, whether they are desktops, mobile or ultra-mobile devices. Recent study by Bales et al. [10] focused on methods and content of web information re-access and sharing among personal devices. It showed that cross-device reaccess and content sharing is often very spontaneous and unplanned and that currently native applications play an important role in how users share content. Unfortunately, contemporary solutions for content sharing and information reaccess are mainly document-centric and rely on complicated infrastructure, thus creating barriers for users trying to share information and collaborate [75].

For an on-spot dynamic sharing and collaboration, it is necessary to exchange not only the data itself (e.g. a text document or an address), but to transfer the information in context. This usually requires the transfer of a complete application state (e.g. place on a map with a route along with any comments or annotations) rather than just fragments of information. What is more, this process usually involves information transfer between different platforms - whether they are public displays, smart meeting rooms, desktops or mobile devices.

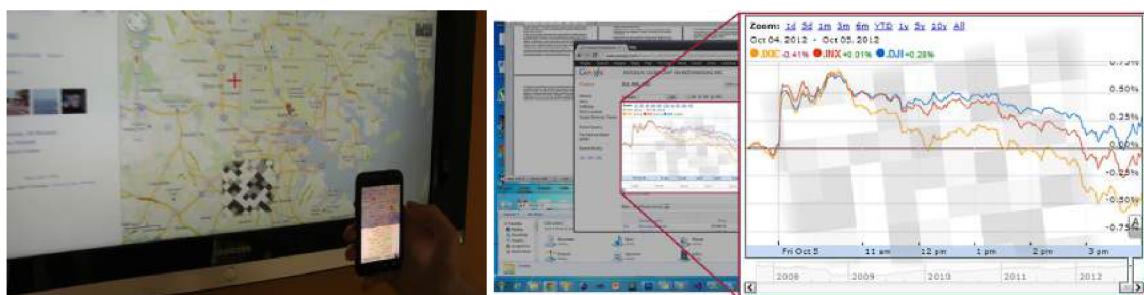


Figure 5.2: In our work, we aimed at exchange of information between a large screen (a desktop computer, a public kiosk, etc.) and a mobile device. This information exchange should be visual and intuitive: based on the metaphor of “video recording” with the mobile camera. We achieved this by inserting a cutout from a Uniform Marker Field into the monitor screen that would be reliably detectable and could accurately establish the location within the screen.

Dearman et al. [32] state that for most users, the migration from a non-mobile platform to a mobile platform is more frustrating than any other means of follow-up. The main source of this frustration is the fact that users are often forced to use many creative, although very time-consuming, methods to enable content and task migration among their devices, because of the lack of support from the software. Among most commonly observed means of content reaccess were [32]: leaving browser tabs open on the mobile device; using handwritten or printed information carried between the devices and inputted on the second device; utilizing shared bookmark systems in order to access and share data between devices; unplanned reaccesses and information sharing are frequently executed by entering search queries into another device.

As Bales et al. [10] state, users would often use features that were made for different purposes as methods to find information later. Many tools, such as Context Clipboard, Evernote, and Dropbox, have attempted to address this problem by enabling easy capture and reaccess, such as saving a link to find it later [53]. Although these tools are seamless and easy to use, they still require planning ahead on the part of the user.

In our work, we were looking for a way to make this task migration or content sharing process instant and intuitive. In particular, the explored method is completely mobile-centric, i.e. no user interaction needs to be done on the side of the content provider screen. That made our approach especially feasible for acquiring information from public displays.

Our approach was to insert a piece of fiduciary information into the screen's contents in order to reliably establish the homography between the screen and the observation of the mobile device's camera. We were using grayscale Uniform Marker Fields (Section 3.4) as the fiduciary marker. The marker field can be reliably detected with moderate computational resources (available on low-end mobile devices). The marker fields are suitable for this task especially because a small fraction of the marker field can be masked out of the complete field and used for the localization.

### 5.1.1 Previous Work and Motivation

For a vast majority of applications, the initial assumption is still that users interact with just a single computing device throughout the day. The practical consequence of this assumption is the lack of collaboration among devices and lack of user-centric activities that may span multiple devices as well as multiple applications.

While there are initial steps in this direction [13], they must support a wider variety of activities and fully recognize the members of a user's device collection. Researchers have proposed supporting activities that span multiple applications [29].

Pierce et al. [126] introduced an infrastructure based on instant messaging which provides mechanisms for applications to send information, events and commands among devices. A system for document redirection based on SIFT features which uses a mobile camera to achieve document drag and drop functionality in a physical environment was presented by Liu et al. [86]. Shoot & Copy [19] is an interaction technique for transferring information from a public display onto a personal mobile phone by using its built-in camera. In similar manner, Point and Shoot allows users to take a photo of an object in order to interact with it [12].

Chang and Li proposed DeepShot [25] – a framework for capturing work state which uses natural visual features (SURF [16]) and tracks them. Their approach is a simplified sibling of PTAM (Parallel Tracking and Mapping [79]). However, despite various techniques to balance the features' density in the camera view, it is impossible to ensure the presence of enough visual features in the whole view. In the case of observing a computer screen, the problem is even more difficult, because unlike the real world, the monitor screen tends to contain surfaces of exactly constant color, backlit by the monitor lamp and thus avoiding any external lighting which would help distinguish unique locations.

A recent step towards direct information transfer from a desktop screen to an ultramobile device are the VR Codes by Woo et al. [177]. In this solution, a digital payload is encoded into solid gray surfaces on the screen by a time multiplex. The encoding requires a significant computational effort on the desktop monitor side, and assumes a particular design of the camera (rolling shutter) on the mobile side. This method could be more promising for the desired purpose of task migration if it allowed encoding the information into arbitrary images and into dynamic content.

In our research, we wanted to go further and provide users with a lightweight solution for information transfer, which is able to work with different types of information and contexts, respects the need for privacy and supports additional metadata generated through interaction which is useful for future interactions on other devices.

There are four basic scenarios when we considered content and task migration:

*Sharing content with an (ultra)mobile device* which is one of the most desired and at the same time most frustrating scenarios. Very often “going mobile” means significant reduction of comfort, pace of work and accuracy in the favor of accomplishing tasks on the road. Usually this scenario involves explicit planning and preparation, for example synchronization of documents through specialized tools and services, which is time consuming.

*Sharing content with a desktop computer.* Besides consuming information (reading books, content from web pages, games), users also often generate multimedia content on the go – they take photos, record video sequences, create notes and bookmarks. As in the previous scenario, specialized synchronization services are commonly used, even though they are document-centric and they are not able to capture and handle the application state.

*Sharing content between (ultra-)mobile devices.* Only few solutions exist that allow for sharing of content among (ultra-)mobile devices. Most commonly seen scenario is a content exchange between devices belonging to different users, content and state transfer between mobile devices of one user (e.g. task migration from smart phone to a tablet) is rare [25].

*Sharing content between desktop computers through an (ultra-)mobile device.* Despite wide availability of cloud-based platforms (SkyDrive, Dropbox, Google Drive, etc.) and traditional (FTP, email) file sharing solutions, people still tend to rely on USB drives [32] to transfer content from one (desktop) computer to another.

### 5.1.1.1 Objectives of Our Research

The main objective for design and implementation of our task collaboration system was to transfer tasks and information among a large variety of devices while minimizing configuration time and being as intuitive as possible. An inspiration and the leading metaphor was video recording on mobile platforms and augmented reality applications in general, where users just point their device’s camera at the object of interest and immediately begin to record (capture) it or interact with it. This means that besides transferring a simple document or URL to another device, also complete application state and related metadata are migrated to the requester device.

We have designed our task collaboration system with the following goals in mind:

- Fast deployment without changes to existing network infrastructure.
- Avoid requirements for a shared server or other central hub.
- Users do not have to manually configure network settings or configure mobile devices.
- Support of variety of interaction scenarios in both online and offline environments.



- Because communication protocols and ways in which messages are delivered are always changing, the system must be flexible enough to support these changes and allow for an easy modification of the existing protocols and addition of new protocols.
- Implemented security must ensure that users have only access to certain parts of the system and do not access restricted parts of system and applications.
- Integration of available standard computing components to support effective collaboration by combining the most suitable set of existing resources available on nearby devices.

## 5.1.2 Proposed System Architecture

With the video recording metaphor in mind, we have designed a highly responsive system (see Section 5.1.4.4) which allows for intuitive task migration without the need for manual application state inspection or copying of “raw” pixels without any additional semantic information (as done in Deep Shot [25]). The task migration process from the system architecture’s point of view is a two-way communication between a *content provider* and a *content requester* device (see Figure 5.3).

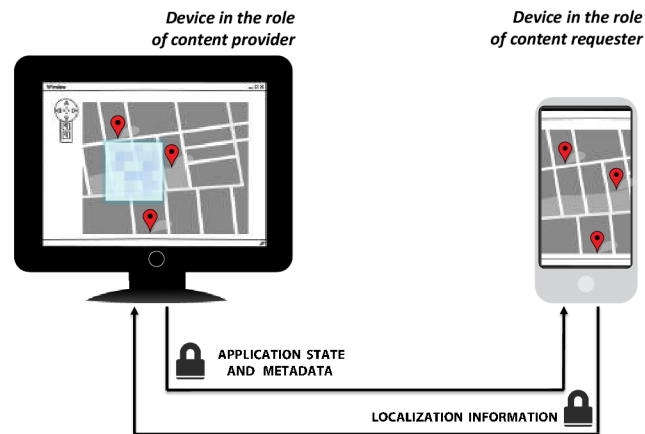


Figure 5.3: The task migration process between a content requester and a content provider device. The content provider device creates an unobtrusive marker field overlay which enables fast and accurate within-screen view localization of the requester device. This localization information is used to select either full application state or to migrate selected content to the requester device.

The *content provider* device is the device with the application state that needs to be transferred to the other device or platform based on the user’s current context. A device in this role is able to share the state of its applications with authenticated clients – *content requesters*. The *content provider* device provides the state of its applications by either querying individual applications for their current work state (URL and internal settings for web applications, the document along with current page number for document viewers, streamed multimedia content, etc.) or provides general services, e.g. providing high quality screenshots of a selected screen area or text from a selected area via optical character recognition.

*Content requesters* are responsible for communication initiation with the target provider device, for selection of the screen region or application of interest and selection of requested/offered content based on the user’s intent. In a typical scenario, *content requesters* are mobile or ultra-mobile devices (smart phone, tablet, PDA).

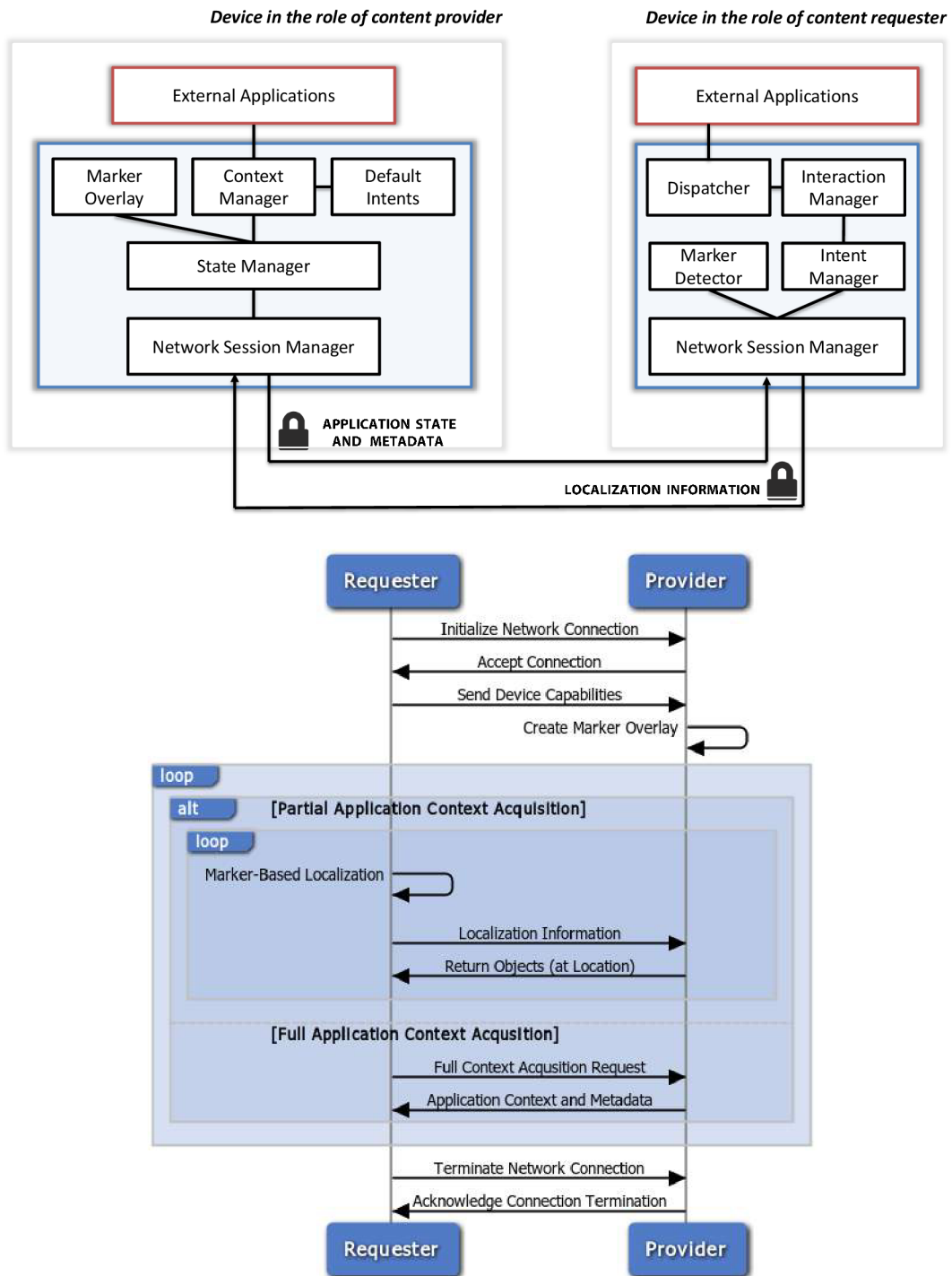


Figure 5.4: **Top:** System overview. Content requester communicates wirelessly with a content provider. The system on both sides consists of a stack of functional blocks (blue rectangle) whose purpose is to ensure information sharing between the built-in or third-party applications (red rectangles). The system thus provides infrastructure for general communication between different devices' applications. **Bottom:** Communication between requester and provider devices during the task migration process. Our system supports full and also partial state migration where only selected state elements are transferred to a requester device.

### 5.1.2.1 System Components and Their Functionality

Communication between requester and provider devices during the task migration process is shown in Figure 5.4. Prior to the task migration, a network connection between the content requester and the provider must be established. The *network session manager* module, which is present on both devices, is responsible for network connection initiation to a remote *content provider* (e.g. public display, laptop). At the moment, the communication is implemented through a WiFi connection, due to its availability on a broad range of devices. The target device is located either via network discovery, by manually entering an IP address (or selecting IP address from history) or the user scans a specific code associated with target device (e.g. on-screen or printed visual marker / matrix code).

After the connection is successfully established, the *requester* device asks for authorization and sends its device capabilities (e.g. camera resolution and internal camera parameters). Afterwards, the *content provider* creates and displays an unobtrusive on-screen localization marker adjusted for the given *requester*. The marker overlay covers the whole screen until the camera's position is detected for the first time (less than a second). Afterwards, the overlay is automatically adjusted, so that it covers just (a part of) the area captured by camera. The part of the marker which is observable is based on camera's position and distance from the *content provider* device. The *marker field detector* on the *requester* device is used for fast and accurate client-side on-screen marker decoding and sending of within-screen localization information to the target device. This approach minimizes the amount of transferred data between the devices, because only the detected 2D position coordinates are sent back to the *content provider*. During the interaction, the marker field is automatically adjusted (its opacity, color contrast and/or brightness are changed in time) based on the detection rate and quality. The transferred and processed content is much smaller compared to the feature-based solutions where either the feature vectors or the whole camera stream are sent to the target device or to an intermediate server for processing and camera localization.

Based on the obtained camera-localization information, the *provider's context manager* queries individual applications and gathers their status. In order to obtain application status from web applications, we have implemented an extension for Google Chrome browser which is able to forward the application state requests from our system and return the gathered state information for further processing. If the selected application is unable to provide its state and metadata, only general intents are available. General intents include high-quality screenshots, and text and phone numbers recognition for the selected part of the screen.

The acquired application state is sent to the *intent manager* on the *requester device* which translates these JSON-encoded messages to intents, directly usable on the requester platform (e.g. on the Android platform the system creates Android intents from JSON messages). Afterwards, the *state manager* provides the user with a visual feedback and updates the GUI, based on the available actions for the selected content. The options include resuming work on a *requester device* – continuing work with a reconstructed web application state on a current device, editing text in an available text editor, manipulation and viewing of images, audio/video playback, etc.

### 5.1.2.2 Marker Field's Design and Detection

In our approach we tried to minimize the required time to localize the client relative to the content provider with high stability. We achieved this using grayscale Uniform Marker Fields [157, 60] (Section 3.4). Marker Fields are efficiently detectable fiduciary markers made up of uniquely identifiable sub-markers. Uniform Marker Fields provide large enough pattern to cover the whole content provider's display. The position identification in the marker field relies on the relative intensity differences between neighboring fields and even a small fragment of the marker field is

sufficient for reliable localization. Thanks to these properties Uniform Marker Fields are easily and unobtrusively “mixable” into the real content showing only a fragment of the marker field with high transparency.

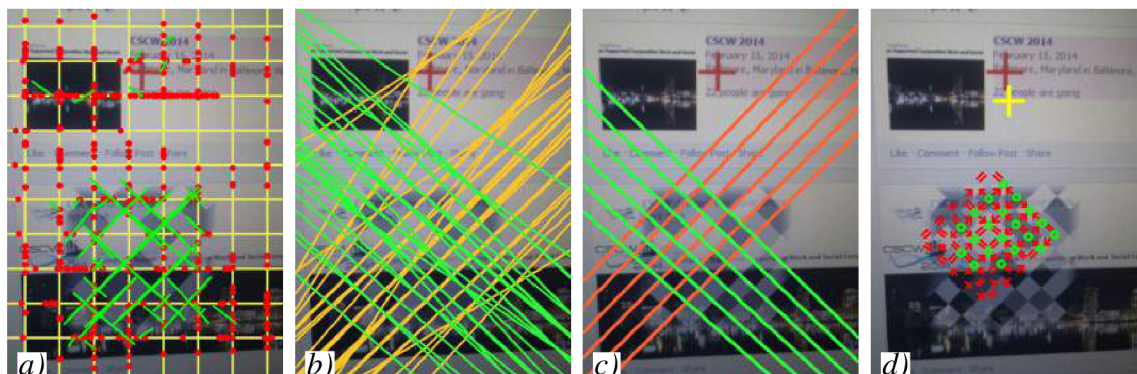


Figure 5.5: Marker detection steps: *a)*: along sparse scanlines (yellow) edges are detected using adaptive threshold (red dots) and expanded into edgels (green); *b)*: lines filtered based on camera rotation and separated into two groups; *c)*: the detected grid; *d)*: based on the extracted edge directions (red arrows and equal signs) the correctly identified positions (green dots) are used to calculate the cross-hairs (yellow) position inside the marker – this position is sent to the content provider to readjust the target position on the display (red cross; pink background – the highlighted element based on the position using the Chrome plugin).

Showing the whole marker on the whole display would be highly obtrusive. Instead, only a small part of marker is shown, which is still reliably detectable by our detection algorithm (see below). We tested constant transparency or pulsing between transparency levels (25 – 75% during performance evaluations) to achieve high detection rates and make the marker less obtrusive.

In order to minimize the outliers caused by the most commonly occurring horizontal and vertical lines in display content (window borders, menus, vertical panels), we rotated the whole marker by angle  $\alpha$  ( $\alpha = \frac{\pi}{4}$  in our tests). To avoid introducing additional long edgels into the content provider’s display, we also used two types of border mask: a sinus function border mask, resembling the jagged edges of postal stamps (see Figure 5.6); and a gaussian blurred mask (see Figure 5.2).

The detailed detection algorithm of grayscale Uniform Marker Fields for augmented reality applications was described in Sections 3.3 and 3.4.1. The main steps of the algorithm for this use-case are demonstrated in Figure 5.5. There were no major changes necessary to the detection algorithm. Improvements and smaller changes are described below.

An improvement over the base algorithm is during the edgel extraction step. The advantage of using modern smartphones is the availability of orientation sensors. The content provider’s orientation is assumed to be mounted on the wall without any rotation. The extracted edgels in the mobile device’s view can be consequently filtered based on its orientation acquired from its built-in accelerometer or gyroscope and the marker orientation ( $\alpha$ ) on the content provider.

Another minor change from the original algorithm was in the edge classification (Section 3.4.2.2). To make the edge classification more robust against transparency, we checked more sample points than the stopping criterion by Wald’s sequential probability test [173] for redundancy.

The other minor changes were implementation specific for mobile devices. The algorithm also does not compute the full homography, since the detected grid and marker position is sufficient to compute for arbitrary image pixel the position inside the marker field. Given the decoded marker position,  $\hat{l}_0$  in each pencil represents either a set of rows starting at index  $l_{0r}$ , or columns starting

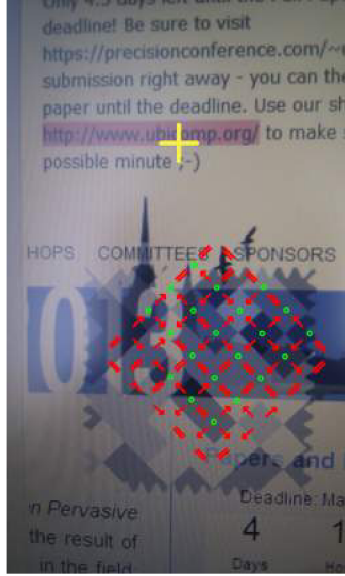


Figure 5.6: A successfully detected Uniform Marker Field as seen by the requester device’s camera with a crosshair shown to aid the user with targeting. The purple highlight is the content provider’s reaction to targeted content based on continuous interaction.

at  $l_{0c}$ . The position  $(m_r; m_c)$  in the marker field of an arbitrary point  $\mathbf{p}$  in the camera image can be determined by solving:

$$\mathbf{p} \times \mathbf{v}_r = k\hat{\mathbf{l}}_{0r} + (m_r - l_{0r})\hat{\mathbf{h}}, \quad (5.1)$$

$$\mathbf{p} \times \mathbf{v}_c = k\hat{\mathbf{l}}_{0c} + (m_c - l_{0c})\hat{\mathbf{h}}, \quad (5.2)$$

where  $\mathbf{v}_r$  and  $\mathbf{v}_c$  are the determined vanishing point for the pencils representing rows and columns, and  $\hat{\mathbf{h}}$  is the horizon.

In order to minimize the noise caused by a shaking hand holding the requester device and to correct slight inaccuracies of the detector, we also use a 2D Kalman Filter before sending the data to the content provider for further processing.

### 5.1.3 Implemented Solution – Chrome and Android

As a proof of concept and as the testing prototype for user testing and exact experimental evaluation, we created a pilot version of the whole system. It consisted of:

- The content provider background service for Microsoft Windows,
- Google Chrome extension as the application-side provider module,
- Android application as the client.

My contributions in this prototype was the client side communication and Uniform Marker Fields detection.

#### 5.1.3.1 Content Provider Chrome Plugin

In order to be able to access and retrieve the full application state of an online application or user-selected parts of a web page, the *content provider* needs to access the loaded content in a web

browser. We have developed an extension for the Chrome browser which acts as a communication bridge between our application on the *content provider* device and web applications / pages running inside the browser (see Figure 5.7).

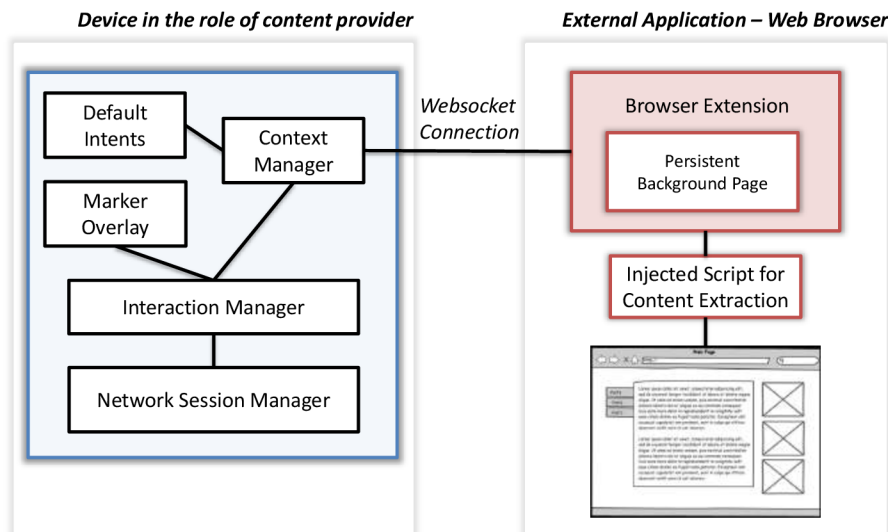


Figure 5.7: The content provider is able to retrieve the application state of a web application running in a browser via a browser extension. The implemented browser extension is able to extract the state of web applications or alter it through code injected into the opened web application.

When the user selects contents of a web page (blocks of text, images, videos or links to other web-based resources) or a complete online application state (e.g., a trip planned in Google Maps) for a migration, the extension finds the active web application and through code injection inserts a script into it. This script is able to directly manipulate with the page’s Document Object Model (DOM) and extract the required parts of the online application/web page and send them back to the *content provider*. This script executes in the context of a page that’s been loaded into the browser, making it a part of a web application, not as part of the extension. Besides extraction, the script is also able to inject information into web applications (forms), thus allowing for continuation of task. Afterwards, the *content provider* forwards them to the *requester device*, where the application state is reconstructed, thus allowing the user to continue with a task on the other device.

Besides retrieving the application state, the browser extension is able to directly manipulate with web page’s DOM and provide visual feedback for user’s actions. When the user is selecting a part of a web page for migration with an ultra-mobile device, a virtual cursor can be created above the marker. This way the user can clearly see on the provider screen what content is selected for migration (see Figure 5.6 and Figure 5.8 for example).

### 5.1.3.2 Content Requester Android Application

For the implementation of a mobile *content requester* prototype we chose the Android platform because of its availability on a broad range of mobile and ultra-mobile devices. For the initial design of our application, the main goal was a clean and minimalistic design of control elements, which will support the video recording metaphor.

After starting the application, the user is welcomed with the option to choose between connecting to a previously used *provider* device, using network discovery to discover available *provider* devices

## Attend CSCW

Baltimore. Bustling colonial port. Birthplace of our nation's metropolis. Ever a welcome refuge for free thinkers.

Baltimore is the port city for the mighty Chesapeake Bay and its cuisine. "Charm City" is renowned for its patchwork of districts. From our conference venue in vibrant Harbor East, it is a : Federal Hill, restaurants of Little Italy, museums of Mt. Vernon, the oldest saloon in the country at Fell's Point. The many ways together by the seven-mile long waterfront promenade of the

Serving as the center point for the Mid-Atlantic region of the country, Baltimore can be reached from almost anywhere. It is served by four major international

Figure 5.8: The content provider directly manipulates the on-screen content and provides a visual feedback for the user's actions. For content migration assistance either a virtual cursor can be displayed above the marker (yellow crosshair), or selected element can be highlighted (area with a pink background).

or by scanning a specific code associated with target device (e.g. on-screen or printed visual marker/matrix code).

The main user interface can be seen in Figure 5.9 left and middle. The top of the screen is filled with the upper part of the camera preview. This way the bottom part of the camera stream containing the marker stays hidden on the requester side. The bottom part of the UI shows the preview of the information acquired from the content provider (text on the left, map or images in the center). During the interaction between the *content provider* and the *requester* device, the application stores the history of the accessed content, which allows the user to migrate multiple data, and choose between them afterwards. The user can optionally migrate the whole state of the viewed application (button on the bottom).

The application uses the video stream from the camera to identify the position and orientation of the *content requester* relative to the *content provider*. For best performance we use double buffering of the video stream. The marker field detection algorithm was implemented in native code through Android NDK toolset that allows implementation application parts using native-code languages such as C and C++.

The detection algorithm computes the position inside the marker and also the position of a virtual cursor (see Figure 5.8). These coordinates are sent to the *content provider*, which uses them to extract semantic content and move the visible marker field fragment on the *content provider* device, so that it is still in the bottom part of the video stream and rarely registered by the user.

### 5.1.4 Experiments: Empirical Tests and User Study

We have performed a user test to find out the most acceptable shape of the marker field and the parameters of mixing it into the desktop screen image. The technical evaluation involves tests of reliability of detection of the marker field on different screen contents and under different viewing angles. Finally, computational performance is evaluated.

#### 5.1.4.1 Marker Unobtrusiveness

We conducted an initial user study to observe how would people use our prototype. Our main goal was to find out how obtrusive was the usage of marker fields for task migration for participants and whether this approach is feasible also for inexperienced users.

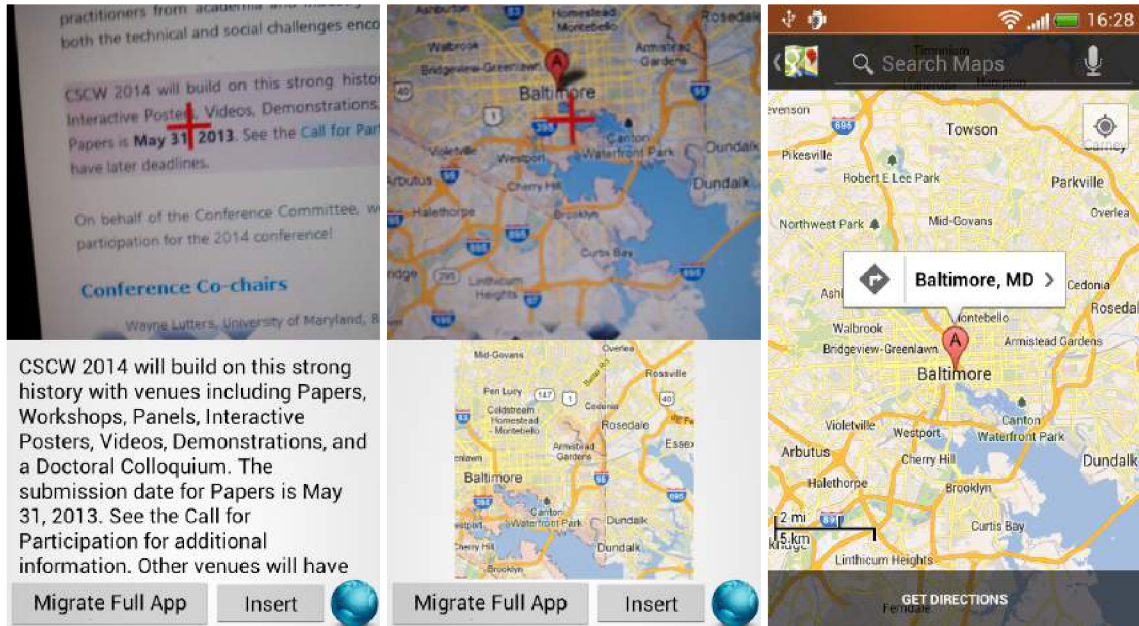


Figure 5.9: Content requester application. Left and Middle: Partial application state acquisition. Only selected parts of state (block of text and high resolution image) are migrated to a requester device. Right: Full application state (with metadata and internal variables) is migrated to a requester device.

The study we conducted consisted of 25 participants. In the beginning, participants had to fill out a questionnaire. This questionnaire asked them about their technical expertise, their knowledge regarding mobile phones, as well as some demographic statements. The average age of attendees was 28 years, the youngest participant was 21 and the oldest was 42. Ten participants were from non-technical professions.

All attendees used at least one ultra-mobile device (e.g., smart phone, tablet, PDA) and one desktop computer or laptop on a daily basis. The average device count per participant was 3.18. 20% of our participants used the camera on an ultra-mobile device for taking pictures on a daily basis, 36% several times in a week, 23% at least once in a month and 21% less than once in a month. 9% of our participants used the camera on an ultra-mobile device for video recording on a daily basis, 16% several times in a week, 38% at least once in a month and 37% less than once in a month. 72% of participants used multiple devices for content reaccess on a daily basis. Table 5.1 shows reported usage of most frequently used methods of content reaccess by participants.

Method of content reaccess	Reported usage
File synchronization service	88%
Search queries	76%
Flash disks or external drives	60%
Shared bookmark systems	56%
Leave browser tabs open as a reminder	40%
Handwritten or printed notes	16%

Table 5.1: Reported usage of most frequently used methods of content reaccess.



After filling in the questionnaire, we introduced our system and four basic task migration scenarios: cartographic map state transfer, acquisition of textual information from a web page, acquisition of an image from long online article with a photo gallery and resuming writing of a text on a laptop.

We provided the participants with an Android smartphone and a laptop with our system; the laptop also contained an application which allowed fast change of marker parameters from presets. Participants were asked to rate marker presets based on perceived obtrusiveness on a five point Likert scale (1–least obtrusive, 5–most obtrusive). In order to be able to compare feedback from participants, we have created ten marker presets divided into four categories:

1. Marker with constant opacity (20%, 40% or 60%), without blurred background.
2. Marker with constant opacity (20%, 40% or 60%), with blurred marker background (gaussian blur with four pixel radius).
3. Marker with variable opacity, pulsing between 20% – 40% or 20% – 60%, without blurred background.
4. Marker with variable opacity, pulsing between 20% – 40% or 20% – 60%, with blurred background (gaussian blur with four pixel radius).

Figure 5.10 shows the user-reported average obtrusiveness for individual marker presets. The average rating across all presets was 2.65. Markers with high opacity were perceived as most obtrusive (average rating 3.54), while markers with 20% and 40% opacity had similar, significantly lower, average rating (1.67 and 2.38). Among our participants, the presence of blurred background or periodic changes in marker opacity had only minimal influence on perceived obtrusiveness. Application of blur on the background decreases the amount of natural edges present in the image and allows for the marker field edges to prevail. Similarly, the pulsing intensity of the marker allows for periodic appearances of highly opaque form of the marker field, which can be tracked afterwards or at least can provide localization information in discrete time frames. The fact that the users tend to tolerate these modes of mixing, offers truly reliable on-screen localization with acceptable levels of obtrusiveness.

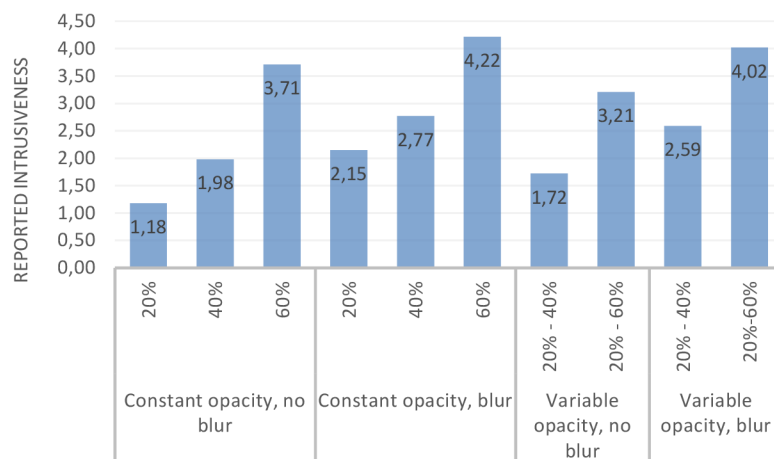


Figure 5.10: Reported average obtrusiveness for marker presets. Percents under each bar are the marker opacity; percental ranges are the extremes of pulsing opacity.

We also conducted a test of our system with simulated network latency, in order to find out the maximum time the system should need until a content is returned to the user's device. The average time after which participants perceived system as poorly responding (obtrusiveness rating 3-5 on a 5-point Likert scale) was 4.5 seconds. This slightly contradicts the findings of Boring et al. [19], where ten seconds response time was acceptable by most users.

In general, our system was perceived very positively, with 86% of participants stating that it would definitely help them with content reaccess. 72% of participants would use it to obtain information from public displays. In this case, the biggest concern were privacy issues – fear that publicly available system could access private information stored in mobile devices due to security flaws or identify individual users and track their actions.

#### 5.1.4.2 Marker Field Detection Reliability

We tested the reliability of our marker field detection algorithm, with the marker mixed into natural screen contents. This is important for smooth user experience. We created a setup consisting of one device in the role of content provider and one device in the role of content requester. As a content provider device we used a laptop computer with a high-resolution ( $1920 \times 1080$ , 17 inch) display, and an Android 4.0.4 smartphone Samsung Galaxy S2 for the role of information requester device. The requester device was attached to a base perpendicular to the floor, in a fixed height, focused at a chosen part of the screen (see Figure 5.11). The experiment was conducted in a room with artificial (fluorescent) lighting. Devices were connected through a WiFi connection.



Figure 5.11: Parameters of marker detection reliability setup – distance between mobile device and laptop, range of screen angles used during tests and height range of a mobile device in order to be focused on the same point on screen.

The requester device was held at a distance of 10, 20 and 30 cm and a pitch angle of  $75^\circ$ ,  $90^\circ$ ,  $105^\circ$ , and  $120^\circ$ . On the content provider device a fullscreen web application containing text, several smaller images, and a map was displayed during the experiment. The experiments were conducted with both constant opacity (50%) and pulsing (25 – 75%) marker fragment. The pulsing period for the pulsing marker was set to 1.5 seconds. The resolution of the videos was  $1280 \times 720$  and the framerate was 20 FPS. The size of the fragment from the marker field mixed into the real content was  $180 \times 180$  pixels covering 1.5% of the display.

Figure 5.12 contains the evaluation of the reliability of our detection algorithm for different pitch angles for the content provider device. This figure (and equivalent subsequent ones) show, what is the probability of missing a given number of acquired frames between two successful detections.

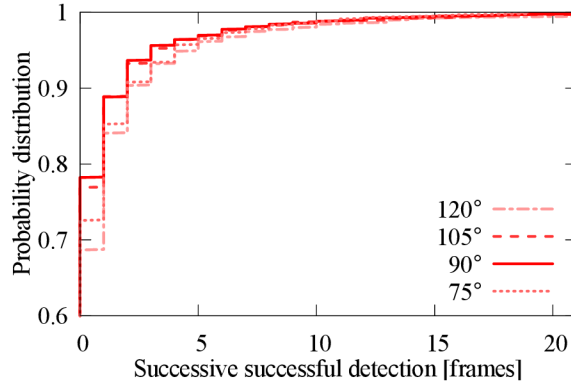


Figure 5.12: Breakdown of the probability distribution of the number of frames between consecutive successful detections for different *angles* between the content provider and client.

The results show, that we were able to detect reliably the content requester’s position and viewing angle within 5 frames with 95 % probability on average over all angles and marker types. For angles  $120^\circ$  and  $75^\circ$  the colors shown on the content provider were already shifted leading to slightly worse results. The framerate of the smartphone camera is 20FPS, which means that the probability of missing the localization for a whole second of time is close to zero.

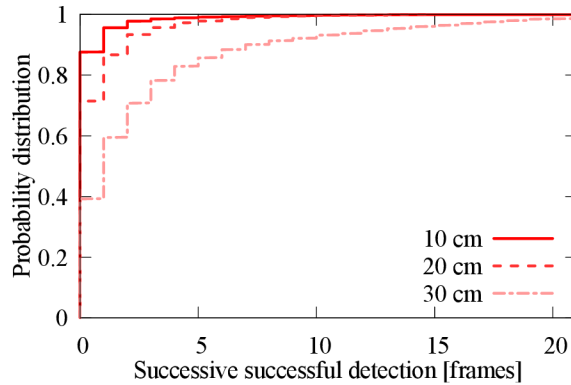


Figure 5.13: Breakdown of the probability distribution of the number of frames between consecutive successful detections for different *distances* between the content provider and client.

During the experiments, we used a fragment of the marker field with constant size. From  $10\text{cm}$  distance the fragment took up  $350 \times 350$  (13%), from  $20\text{cm}$   $230 \times 230$  (6%) and from  $30\text{cm}$  about  $120 \times 120$  pixels (1.5% of the field of view) in the  $720p$  camera stream. With smaller marker fragment in the camera image the number of potential outliers for the grid detection grow, leading to less reliable detection (see Figure 5.13). In our application this issue was solved by information exchange between the content requester and content provider enabling the content requester to set the fragment size based on the distance.

Figure 5.14 shows that the pulsing marker field was detectable more reliably. Thanks to changing opacity, the marker fragment was detected even on highly challenging background, and viewing angles compared to constant opacity.

Finally, we also experimented with different border masks: sinus curve, blurred and non-blurred rectangle. We did not find any measurable difference in reliability between different options. During experiments we used the blurred rectangle, since it was the most aesthetically pleasing for users.

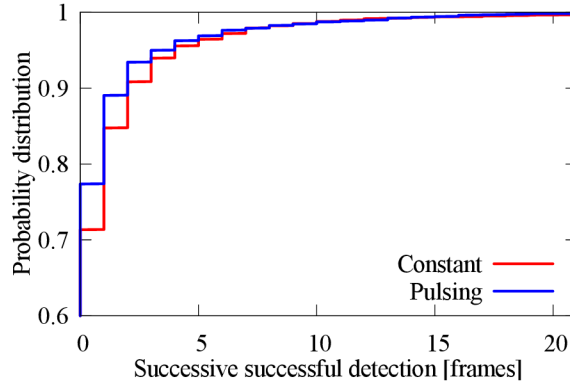


Figure 5.14: Comparison of the probability distribution of the number of frames between consecutive successful detections between constant opacity and pulsing markers.

### 5.1.4.3 Accuracy of Location Finding

For accuracy measurements we used the same videos as in the previous section. The mobile device was fixed during measurements with the visible fix-sized marker segment moving around on the screen. Table 5.2 shows the standard deviation of the determined position on the content provider’s display in pixels. We did not use Kalman filter for these measurements.

[pixels]	75°	90°	105°	120°
10cm constant	11.0	10.0	10.0	27.6
10cm pulsing	6.3	7.6	8.9	11.8
20cm constant	27.7	19.4	21.9	26.4
20cm pulsing	17.0	24.1	24.3	27.0
30cm constant	34.4	27.2	22.6	23.4
30cm pulsing	25.0	27.2	23.9	23.0

Table 5.2: Standard deviation of the detected positions in content provider’s coordinate system in pixels.

The accuracy of the algorithm without corner detection and full homography calculations is relatively low. On the other hand, an unstabilized hand-held mobile device would cause even larger variance in position. As a solution we used a Kalman filter, modelling position and speed of the detected position (measurement variance set to  $\sigma^2 = 400px$ ). The accuracy was sufficient to select blocks of text, map regions, images or menu entries.

### 5.1.4.4 Speed Performance

In order to be able to compare our solution to alternative frameworks (e.g. DeepShot), we have tested four target applications: maps from Google Maps, photos from Picasa, long articles with images from CNN.com and short textual information from Twitter. For each application, 10 information requests were sent and processed. The setup for this experiment was identical to the previous test.

Table 5.3 summarizes the breakdown of the time consumed in different phases of the processing. The marker field decoding process runs independently from state acquisition process.

From the marker field decoding part the edgel extraction and edge classification required on average  $\sim 75\%$  of the time on a mid-range smartphone (Samsung Galaxy S2 1.2GHz ARMv7

Activity	% Time spent
Network transfers (WiFi)	84 %
Provider-side message processing	3 %
State acquisition via plugin	13 %

Table 5.3: Timing breakdown of the mobile client.

processor released in 2011) even with the very low “pixel footprint” of the detection algorithm (14.9% with a 25 pixels between horizontal and vertical scanlines). The overall average time required by our baseline implementation for mobile platforms – excluding the system overhead to acquire the image – was 24.5 *ms* ( $\sim 40$  FPS) for  $800 \times 480$  resolution, extracting on average  $\sim 84$  filtered edgels based on the device’s orientation. We used standard web-site content as the background during these evaluations with the marker constantly visible in the field-of-view. Results might differ considerably for different smartphones or display content.

The results show a significant speed increase when compared to task migration solutions based on visual features - authors of the DeepShot [25] task migration framework report 7.7 seconds (SD 0.3 seconds) for processing the request. Our approach allows for real-time information feedback for a selected screen area. A big advantage of our system is the utilization of a video stream, which allows for fast localization from the following frame, if the localization fails on the current one.

## 5.1.5 Conclusion

I described a solution for instant document reaccess and task migration, based on visual communication between an ultra-mobile device and a content provider screen. We explored the approach based on mixing a fiduciary marker field into the screen – in a way that would be least obtrusive, while offering good detection performance even on low-end mobile devices.

Our system allowed for direct task migration and document reaccess – without any direct interaction with the content provider system, i.e. straight from the mobile device. This made our solution suitable for collaboration on public displays as well as one’s private desktop monitor. At the same time, the solution was instant in the communication and intuitive (being based on the metaphor of video recording).

We created a prototype implementation of the whole solution using Google Chrome (as a plugin) and Android devices. This prototype was examined within a user study and by a set of performance evaluation experiments. The results indicated that it substantially outperformed the existing solutions: the detection and recognition of the marker field was done in real time by a mid-level cellphone; the recognition was reliable even for different observation angles and for cluttered screen content. Our solution operated on a video stream with all the benefits: if one camera frame failed for a reason, the mobile client program determined the location from a subsequent valid one.

A possible future direction is to extend the computer vision solution on the ultra-mobile device by natural features tracking in order to minimize the required time for the fiducial to be present on the screen. The fiducial could be displayed for a limited period of time, only to acquire the reliable information about the observed screen part. As long as the natural features will be sufficient for tracking the location, the fiducial should not be displayed until necessary again. This approach was also followed by Pavelková [124] for eye-tracker – display combination, mentioned at the beginning of this chapter.

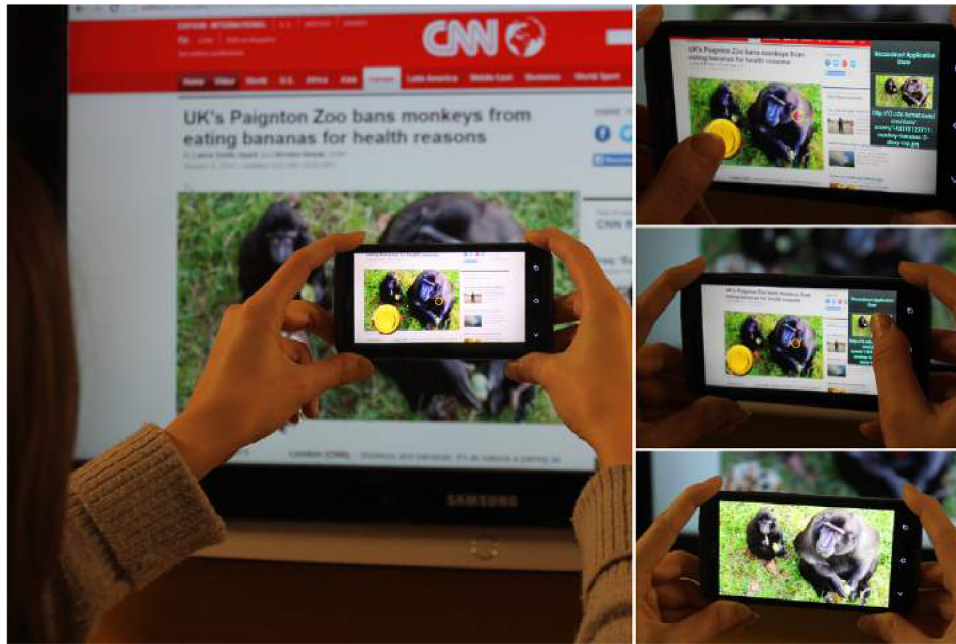


Figure 5.15: Content transfer steps. (left) the user aims at the desired display (which acts as the content provider). (upper right) After the initialization phase, the content is seen in the live video on the mobile device. (center right) The user starts moving the requester device (usually a mobile phone, tablet or a PDA). As long as the device is pointed at the original display, selected parts of the on-screen application are recognized (images, maps, block of text or individual words). (lower right) Whenever user indicates so, the selected content is transferred to the requester device either for immediate processing (editing, opening in a predefined application, sharing or saving) or saved in history for later review and processing.

## 5.2 Continuous Task Migration using Natural Features

In our follow-up work in collaboration with Rudolf Kajan, Adam Herout and Alena Pavelková we improved on the system presented in Section 5.1. We created a full augmented reality experience on the mobile phone centered more around interaction. My main contributions were the camera pose estimation on the content requester device and its testing. Figure 5.15 shows briefly how content is transferred using the improved task migration technique. The description below is based on our publication [73].

The main objective for the design and implementation of our task migration system, similar to previous sections, was to transfer tasks and information among a large variety of devices without any special hardware or changes to existing infrastructure, while being as intuitive as possible. At the same time, we argued for *continuous* interaction, instead of discrete selection. This way, the feedback is provided to the user throughout the whole interaction and the user is able to manipulate with numerous selected objects in one continuous session, instead of breaking down the session into two phases – acquisition and processing of each individual object.

In order to allow mobile use, the requester device continuously tracks itself with respect to interactive displays in its surrounding using its built-in camera and computes its spatial relationship between itself and each identified display. Our interaction method used for state acquisition and transfer is based on rich foundations of *interaction through video streams* [20, 21, 14], *bimanual interaction* [23, 82, 58], *interaction at a distance* [125, 69, 105] and *mobile augmented reality*.

*Augmented reality* is often used in the literature on mobile devices equipped with displays to create virtual pointers and help users access remote content. Pears et al. [125] used a camera phone for pointing on large screens. Sweep [11] used optical flow analysis to enable continuous relative control of a remote pointer on a large screen. The Boom Chameleon [164] is a spatially aware display. Peephole Displays [181] introduced two-handed interaction technique which combined pen input with a spatially aware displays.

Our inspiration and the leading metaphor was video recording on mobile platforms and augmented reality applications in general. In these scenarios users usually just point their device's camera at the object of interest and immediately begin to record (capture) it or interact with it. Our interaction technique provides the user *at every moment* with relevant tasks and content-migration options for the selected application and its content. Our approach thus emphasizes spontaneous and unplanned content access with minimal user input, while being responsive.

The backend system of our improved method is identical to the system described in Section 5.1 (see Figure 5.4). The most changes were made in the interaction technique on the content requester side and camera pose estimation.

### 5.2.1 Marker Tracking and Natural Key-points Based Detection

Our solution utilized a combination of natural features based detection and marker tracking in order to reliably establish the homography between the screen and the observation of the mobile device's camera. This allows us to employ a fast and precise *continuous* interaction even on low-end mobile devices.

During the initialization phase and in case of fast camera movement, we employed natural features based detection. Detecting keypoints and extracting features on the mobile phone would be too costly on some low-end devices. Instead, the features are computed and matched on the content provider. Similar approach was taken in [25] and [14]. The difference is, that our solution does not stream the video, as it would generate high network traffic (see experiments). Instead, we use natural features detection as a fallback method, and send frames only in large intervals.

A major disadvantage of pure natural features based methods is that they rely on rich features being present on the target display. This assumption is rarely met in the highly manhattanic world of desktop and web applications. As a solution, we utilize a *virtual cursor* using the Vuforia library on the content requester side combined with a small natural image target on the content provider. The natural image target is used to compute the required offset on the content provider caused by the camera movement. The computed relative correction is sent to the content provider. This is our primary method for camera movement tracking.

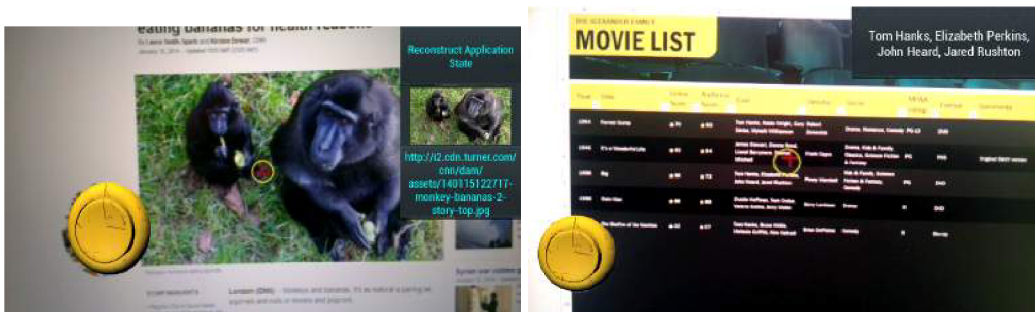


Figure 5.16: The proposed interface of the content requester application with virtual button overlaid over the natural image target for content recording.

The image target also serves a secondary objective as a reference position to draw the augmented UI elements of the application. These elements give visual feedback to the users, so they move within acceptable distance from the content provider. The augmented layer also hides the obtrusive marker on the client side (Figure 5.16).

If multiple users are simultaneously interacting with a single provider, their primary mean of localization is natural features based detection of a target. In the case that multiple targets overlap, clients automatically fall back to natural keypoints tracking on the target display. This approach ensures that the interaction will not be interrupted even if multiple users are migrating the same elements at the same time.

## 5.2.2 Content Requester Android Application

For the implementation of a mobile *content requester* prototype we chose the Android platform because of its availability on a broad range of mobile and ultramobile devices. For the initial design of our application, the main goal was a clean and minimalistic design of control elements, which will support the video recording metaphor.

After starting the application, the user is welcomed with the option to choose between connecting to a previously used *provider* device, using network discovery to discover available *provider* devices or by scanning a specific code associated with target device.

After connection, the detection algorithm computes the position of the *requester* device with respect to the current *provider* and optionally position of a virtual cursor (see Figure 5.16) which helps the user to identify the exact spot the user is pointing at.

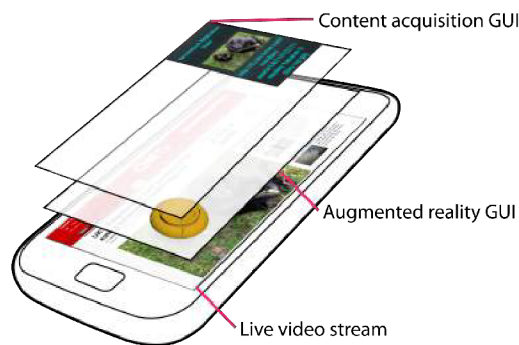


Figure 5.17: The user interface of the requester device consists of three layers - live video stream from the device's camera, 3D GUI based on augmented reality and content acquisition GUI dynamically updated with respect to the current content selection.

The main user interface can be seen in Figure 5.17. The interface consists of three layers. Live video stream from device's camera is on the lowest layer. On top of the video stream is an augmented reality 3D GUI. The main advantage of the augmented reality interface element is that it naturally guides the user to the appropriate distance and angle with respect to the content provider by the means of size and rotation change. This interface is used to mark the content for migration – similarly to a video camera's *record* button. It is used by user's non-dominant hand. The topmost layer consists of 2D elements which are dynamically changed based on the content which the user is currently selecting for migration and display preview of recorded items – blocks of text, images, hyperlinks. If any of this content is touched by the user's (dominant) hand, options for further processing are displayed. These options are dependent on the selected content and include editing, sharing on social networks, saving to the device or launching an appropriate android Intent. During



the interaction between the *content provider* and the *requester* device, the application stores the history of the accessed content and allows the user to access it later.

### 5.2.3 Experiments

We tested both the reliability of our feature detection-based algorithm and the tracking performance of the Vuforia library as a part of the user tests. We created a setup consisting of one device in the role of content provider (17 inch laptop computer with  $1920 \times 1080$  resolution display and an Intel® Core™ i7 processor running at  $2.2\text{ GHz}$ ) and one device in the role of content requester (Samsung Galaxy S2 smartphone).

#### 5.2.3.1 Tracking Reliability

In order to evaluate the reliability of the feature detection based algorithm used during initiation, the requester device was attached to a base perpendicular to the floor, at a fixed height, focused at a chosen part of the screen (see Figure 5.18). The experiment was conducted in a room with artificial (fluorescent) lighting. The devices were connected through a WiFi connection.



Figure 5.18: Parameters of reliability setup for initialization – distance between mobile device and laptop, range of screen angles used during tests and height range of a mobile device in order to be focused on the same point on the screen.

The requester device was held at a distance of 10, 20 and 30 cm and a pitch angle of  $75^\circ$ ,  $90^\circ$ ,  $105^\circ$ , and  $120^\circ$ . On the content provider device a fullscreen web application containing text, several smaller images, and a map was displayed during the experiment. The resolution of the images sent to the content provider to compute the initial homography was  $320 \times 240$ .

Figure 5.19 contains the evaluation of the reliability of our natural feature based detection for different pitch angles for the content provider device. For each distance settings, 20 images were taken – 5 per each angle. The results show that the natural features based detection was highly reliable. For angles  $120^\circ$  and  $75^\circ$  the colors shown on the content provider were highly shifted changing the visible key-point features causing slightly worse results. This issue is caused mostly by the used display during testing, and would harm any computer vision based technique.

During user testing, the participants were given several tasks to migrate data. During the tasks, we recorded the tracking status of our system. The detection algorithm ran at 20 frames per second.

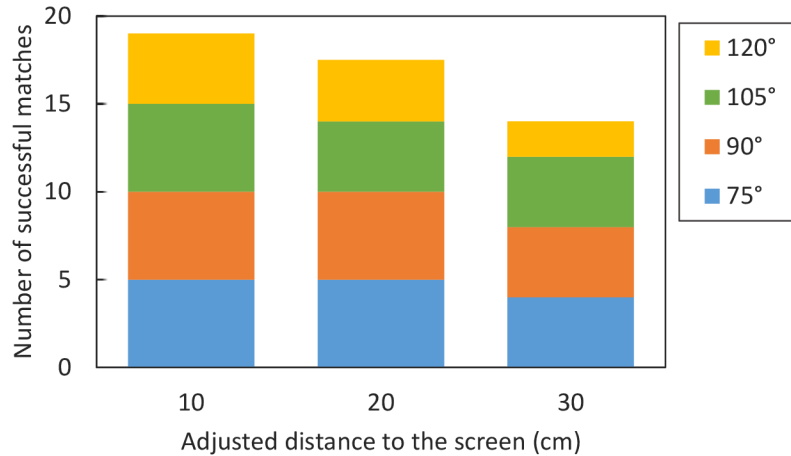


Figure 5.19: The results of natural feature detection reliability for different pitch angles. For each distance settings 20 images were taken - 5 per each angle.

Figure 5.20 shows the probability of successfully localizing the content requester relative to the content provider. The blue line shows the probability of successfully tracking for a given amount of time given that we successfully tracked the previous frame. After the tracking was lost, a full frame was transferred to the content provider, in order to be used for natural features based detection (hence the step around the 1s mark). The delay interval of 1 second for full frame sending was chosen not to overload the network connection. The results show that after 4s the cursor tracking algorithm was able to restore tracking with 99% probability. This causes a short but noticeable delay for the user after the tracking is lost and needs to be restored. Despite this delay, the overall performance of the natural feature detection is good enough to provide the users with *continuous interaction*, and is an area which we are planning to improve.

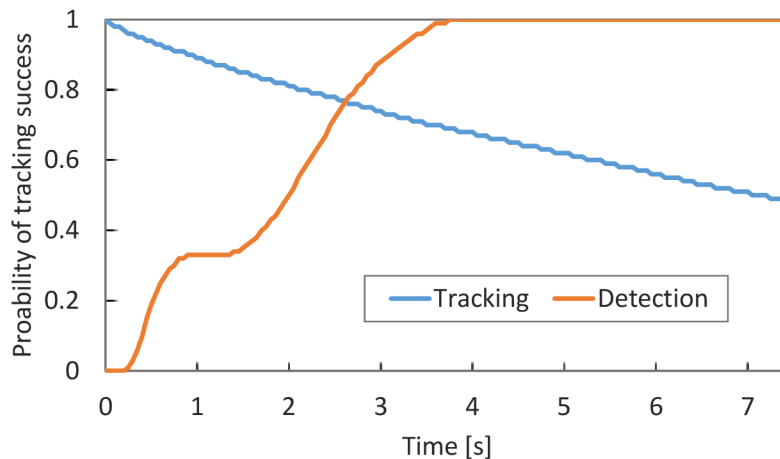


Figure 5.20: Breakdown of the probability distribution for the tracking phase (blue) – the probability of tracking continuously for a given time; and the detection phase (brown) – the probability distribution in time of successfully restoring cursor tracking.

### 5.2.3.2 Speed Performance

In order to be able to compare our solution with alternative frameworks (e.g. Touch Projector or DeepShot), we tested four target applications: maps from Google Maps, photos from Picasa, long articles with images from CNN.com and short textual information from Twitter. For each application, 10 information requests were sent and processed. The setup for this experiment was identical to the previous test. All tests were done using the hardware from the setup for the reliability testing.

Table 5.4 summarizes the breakdown of the time consumed by the initiation phase of the interaction for a single frame. The majority of the time (59.2%) was consumed by network transfer of the reference image. This gives 1.3 FPS for the natural features based position estimation part. In the setup experiment we sent reference images in 1s intervals to avoid flooding the network.

Activity	Time spent
Client side processing	11 ms
Network transfers (WiFi)	442 ms
Provider-side processing	289 ms
State acquisition via plugin	4 ms
<b>Sum</b>	<b>746 ms</b>

Table 5.4: Timing breakdown of the initialization phase. Client side processing covers camera image retrieval and resizing operation. Provider-side processing includes image reconstruction, acquiring screenshot and homography calculations.

Once the tracker was initialized, it was able to track the cursor with full 20 FPS speed provided by the camera on the tested smartphone. After the user decided to migrate content from the content provider, the required time to transfer information was 19 ms on average including network communication (approximately 73%).

The results show a significant speed increase when compared to task migration solutions based on visual features – the authors of the DeepShot [25] task migration framework report 7.7 seconds (SD 0.3 seconds) for processing the request – and allows for real-time information feedback for a selected screen area. An advantage of our system is the utilization of video stream, which enables continuous interaction instead of discreet selection.

### 5.2.3.3 Bandwidth usage

Table 5.5 summarizes the required bandwidth of our system measured during the user tests. The last column contains the theoretical minimum required bandwidth if we used natural features only. In this scenario we assume that the content requester detects and extracts at least 20 binary feature vectors of 512 bits (common for state-of-the-art binary feature descriptors). These feature vectors are then sent to the content provider for homography computations. We also assume a speed of at least 15 FPS for continuous detection.

The results show that our system requires on average  $2.5\times$  less bandwidth than the theoretical minimum bandwidth used up by a pure natural features-based approach. However, 88.4% of the time during interactions (cursor tracking) our system requires just 0.5 kB/s bandwidth, which is approximately  $35\times$  less than a natural features based approach. Our system needs more bandwidth only in the initialization phase and in the case when the cursor tracking is lost during the interaction. In the future, this part could be replaced by computing features on the content requester side.

	<b>Median</b>	<b>Peak</b>	<b>Average</b>	<b>Natural features</b>
Bandwidth	546.5 B/s	82.4 kB/s	7.8 kB/s	19.2 kB/s

Table 5.5: Bandwidth usage of our system used for interaction between content provider and content requester.

#### 5.2.3.4 Content Selection Accuracy

In order to measure accuracy of content selection with our system, we have used targeting tasks based on ISO 9241-9 standard [92]. We used a rectangular target instead of a distinct target point. We asked participants to try to navigate pointer into the rectangular area, while being as fast as possible.

The task started after the connection between requester and provider devices was established and the tracking subsystem was fully initialized. Afterwards users were notified about the trial's start and moved the virtual cursor inside the area filled with text or images. The task ended once the cursor was inside the area and user touched the content acquisition button with the non-dominant hand. We measured the time and the virtual cursor's coordinates throughout trials.

The trials were performed for three different sizes of the target area, corresponding to the sizes of standardized web elements. During the testing trials, timestamp and virtual cursor position was recorded for every received position information. From these recorded data, the throughput was computed. Throughput, in bits per second, is a composite measure derived from both the speed and accuracy in responses [92]. The results are shown in Figure 5.21.

Another information obtained from these data is average target re-entry. This information estimates how many times has the virtual cursor left and re-entered the target area after it entered it the first time. As can be seen in Figure 5.21 right, target re-entry strongly depends on the size of the target area. The reason for majority of target re-entries in small target areas was the natural hand tremble.

We have also computed additional accuracy measures as defined in [92] - movement error, movement offset and movement variability. Average movement error across all trials was 20.78 pixels, average movement offset, computed from absolute values of all movement offsets, was 15.63 pixels and movement variability was 20.09 pixels. Movement direction change occurred  $3.37 \times$  per trial and orthogonal direction change  $8.37 \times$  per trial.

When compared to commonly used pointing devices, our system had a lower throughput (TP), but also lower error rate (ER) for primary migration targets - images, text paragraphs, links. In [92] the reported values were: joystick TP 1.8 bps ER 9%, touchpad TP 2.9 bps ER 7%, trackball TP 3.0 bps ER 8.6%, mouse TP 4.9 bps ER 9.4%.

These results show that our system is comparable to commonly used pointing devices and usable even by inexperienced users. As possible future research, the throughput and error rate could be improved. As a possible solution to compensate for natural hand tremble (which is the main source of lower TP and higher ER), we could employ a smooth estimate of cursor's position and add the option to (semi-)automatically zoom for a better selection of content from remote providers.

#### 5.2.4 Conclusion

We created an improved prototype using feature-point tracking, which allows for task and content migration from web applications to a mobile client. This prototype was examined within a user study and by a set of performance evaluation experiments. The results indicate that it substantially

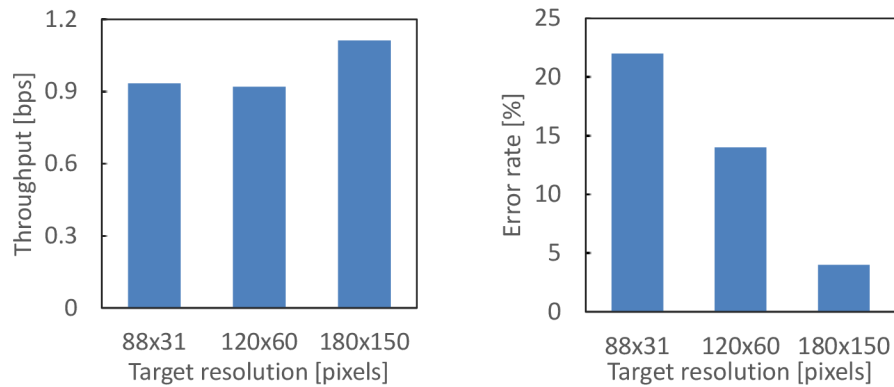


Figure 5.21: Left: Average throughput acquired during experiments where the user was moving the virtual cursor from starting point to the target area. In the graph, the relation between the throughput and the sizes of the target area is shown, which were chosen according to the sizes of standard web elements. Right: Average error rate depending on the size of the target area.

outperforms the existing solutions: the localization and task migration is done in real time on a mid-level cellphone; the localization is reliable even for different observation angles and distances. Our solution operates on a video stream for continuous interaction.

Future research directions include increasing interaction distance, thus allowing for continuous interaction with remote and unreachable displays. This could be made possible by employing automatic and semi-automatic zoom functionality for selected content. Another possible area for improvements is making the tracked cursor as unobtrusive as possible for the other users or minimizing the required time for the fiducial to be present on the screen.



## Chapter 6

# Poor Man's Virtual Camera

This chapter describes another important and distinct usecase of the Uniform Marker Fields which constitute the core of my work. In this case, I, with my colleagues, proposed to use the marker fields in the film-making domain for a structured greenscreen canvas.

Since the early years of film-making, artists have wanted to create artificial worlds to bring their ideas onto the film screen. Modern digital virtual production is still a new technology that requires several steps from different research areas: matting or background subtraction and visual tracking from computer vision, alternatively hardware based tracking from robotics and realistic rendering from computer graphics. Each of these steps is computationally extremely expensive. We proposed a cheap, instant and reliable matchmoving solution for chromakeying-based film effects [35] enabling real-time mobile solution for virtual production for preview purposes and as a fast, simple and cheap solution for low quality production. We pursued this idea further and created a fully functional mobile implementation [156]. My contribution in this research was the adaptation of Uniform Marker Fields detection algorithm as part of the chromakeying process. I proposed a color mapping between grayscale and greenscreen marker fields and matting method using GPU shaders for the functional prototype. My contributions also include the implemented real-time solution and its experimental evaluation. The subsequent description follows closely our publications.

### 6.1 Motivation

One of the early methods to mix the artificial and virtual worlds was *matting* – an effect for composition of several images into one frame. Several approaches for matting were developed in time: e.g. glass paintings, double exposure or using optical printer [169]. Nowadays, the most commonly used method is *chroma keying*, where a specifically selected color (often green or blue) is set to transparent and substituted [44]. Computer vision developed various methods for removal of background, whether with constant color [149], as used in filmography, or with an arbitrary image [152, 28].

During rendering of a virtual world to replace the blue or green canvas, the exact movements of the camera must be determined. This information can be obtained by camera rigs programmed to follow a pre-defined track (e.g. Cyclops or Milo<sup>1</sup>, TechnoDolly<sup>2</sup>), using sensors mounted to the camera (e.g. Insight VCS<sup>3</sup>), by tracking simple artificial markers placed on the canvas [83] or natural

---

<sup>1</sup><http://www.mrmoco.com>

<sup>2</sup><http://www.supertechno.com/product/technodolly.html>

<sup>3</sup><http://www.naturalpoint.com/optitrack/products/insight-vcs/>

image features in the captured real world [33].

Recent advances in mixed reality filmmaking technology include the concept of *virtual camera* – the position of the real camera is detected and used to simulate the virtual camera in the digital world<sup>3,4</sup>. Coupled with a classic camera (also referred to as a *Simulcam*) it forms a tool for superposition of the real and digital world and capturing the scene in one moment. These concepts represent the best available technology today; however, they require high financial and technical resources.

Real-time replacement of a greenscreen with another scene (synthetic or real) can be seen as a kind of an augmented reality system [9], for example, the live TV production [160]. The problem of camera pose estimation can be solved by using markers [172, 76], natural features as points or edges [140], textures [168] or by using other sensors such as GPS and gyroscope on a handheld camera [135]. Commercial virtual cameras are mostly based on motion capture [100, 45], e.g. Insight VCS<sup>3</sup> or ILM virtual camera<sup>4</sup>.

The previously mentioned solutions require complex and costly setup of infrared cameras, additional tracking extensions for the main cameras and external servers. They provide real-time visualization only for the virtual scene and not the augmented result. In the field of augmented reality, there has been prevalent research to create an integrated solution with the help of commodity hardware [97]. Effort has also been put into taking advantage of the growing computational power of hand-held devices for ultra-mobile augmented reality systems [51]. However, Ventura et al. [170] have shown that systems based on feature-points and point-clouds are still realizable only with the aid of additional server-side computations.

Our aim in this research was to eliminate such limitations. The required image processing algorithms are computationally expensive and common mobile hardware still lack the performance to support them in full. There has been extensive research to optimize the computational efficiency of these algorithms. In this work we took advantage of Halide language and compiler proposed by Ragan-Kelley et al.[131, 132]. The main idea behind Halide is to separate the scheduling of the performed operations and the algorithm itself. The main advantage over other domain-specific languages is its transparent support for data-parallel computational units such as AVX, SSE, OpenCL, and – most importantly for our work – NEON<sup>5</sup>. Finding the best schedule for a given algorithm, however, is not trivial and requires deep knowledge of the underlying hardware architecture.

After the precise position of the camera was determined, the eliminated background of the real scene is replaced by the virtual scene. Many graphics algorithms were invented during recent years to achieve a high degree of realism of the virtual worlds: faithful animation of fluids, hair, clothes or properties of light distribution, etc. The composition of the real and virtual world is also not a problem for today’s film production systems. However, this is only possible *after* shooting the given scene and the filmmakers depend on their own imagination during the film shot and its preparation. *Storyboard* is an old and very widely used method for pre-visualizing film sequences using illustrations and images [55, 146]. It represents a great tool for understanding the scene layout. The disadvantages are large time requirements for creating a storyboard for a whole film and complicated incorporation of the real actors and objects into the illustrations.

The bases of our proposed system was built upon grayscale and colorful Uniform Marker Fields (Section 3.4). The advantages of the used marker fields compared to other marker based solutions [172, 76] in a virtual production setting are mainly robustness against occlusion and low contrast. The marker field covers the matting canvas (as a whole or a selected fraction). Then, during the shooting, the camera position is established and a preview of the mixed scene is rendered in real time. Our algorithm is computationally extremely efficient. In contrast with state of the

---

<sup>4</sup><http://www.ilm.com/>

<sup>5</sup><http://www.arm.com/products/processors/technologies/neon.php>



art camera grids our solution can thus run on common mobile devices, providing almost unlimited freedom in movement and a very low-cost virtual camera or simulcam solution.

This solution is unprecedentedly cheap – it is available for a wide range of filmmakers, including amateurs. In general, this relatively simple technique unleashes new creative and innovative approaches, so far unseen especially in indie filmmaking. For simplicity, in the text we are using the term “greenscreen”, but the same applies to any other color (blue, purple, etc.).

This research has been a result of collaboration between Dubská, M., Herout, A., Zachariáš, M. and myself. My main contributions in this research are the following:

- Proposed color mapping for AR use and color selection for the greenscreen marker fields.
- Proposed automatic color calibration for matting by sampling the marker field.
- Mobile prototype implementation of the 3D preview application.
- Real-time performance even on mobile platforms using multi-platform optimization (using Halide [131]).
- Evaluation and testing of the prototype system.

In this section I will focus on these parts of the research. For more detailed information about the proposed virtual camera settings and storyboarding concept, refer to [35] and [156].

## 6.2 Greenscreen Marker Field

In a chromakeying setting, the proposed algorithm first computes the chromakeying mask to segment out the background containing a fiduciary marker. For high quality calibrated cameras the raw data is sufficient to detect the difference between shades of green used by the marker. For videos acquired through low quality or smartphone cameras, the algorithm also maps the different shades of green into a single grayscale image to achieve higher contrast (see Figure 6.1A). The mask and the mapping computations are described in Section 6.3.1.

The fiducials used in our research were based on Uniform Marker Fields described in Section 3.4. The synthesis of such fields is highly time consuming and its size is limited by the size of the uniquely identifiable sub-windows  $n^2$  (see Section 3.4.4).

The detection algorithm, after the shades have been mapped into grayscale representation, follows the detection algorithm described in detail in Section 3.4.2. The process is demonstrated in Figure 6.1.

A benefit of a chromakeying setting is the straightforward foreground-background segmentation. The chromakeying mask was used to filter out foreground edgels during the edgel extraction step (Figure 6.1B). For recovering the edge directions (Figure 6.1E) we used the mapped grayscale intensity value (Figure 6.1A). Based on these edge directions the unique sub-windows can be identified using a decision tree, as in the original algorithm (Figure 6.2).

To improve the precision of our prototype application we employed sub-pixel precise corner search (Figure 6.1F). After a successful detection, these points are tracked using the Kanade-Lucas-Tomasi tracker. This is a change from the pure track-by-detection approach used in previous applications. For camera pose estimation we use LHM [89] initialized by EPnP [84]. The pose ambiguity for planar targets is solved by the method described by Schweighofer and Pinz [145].

The limitation of our approach is that the Uniform Marker Field has to be visible during the whole shooting to be able to track the camera pose. In the future, this can be partially solved by merging visual tracking with motion tracking acquired from hardware sensors. Contemporary

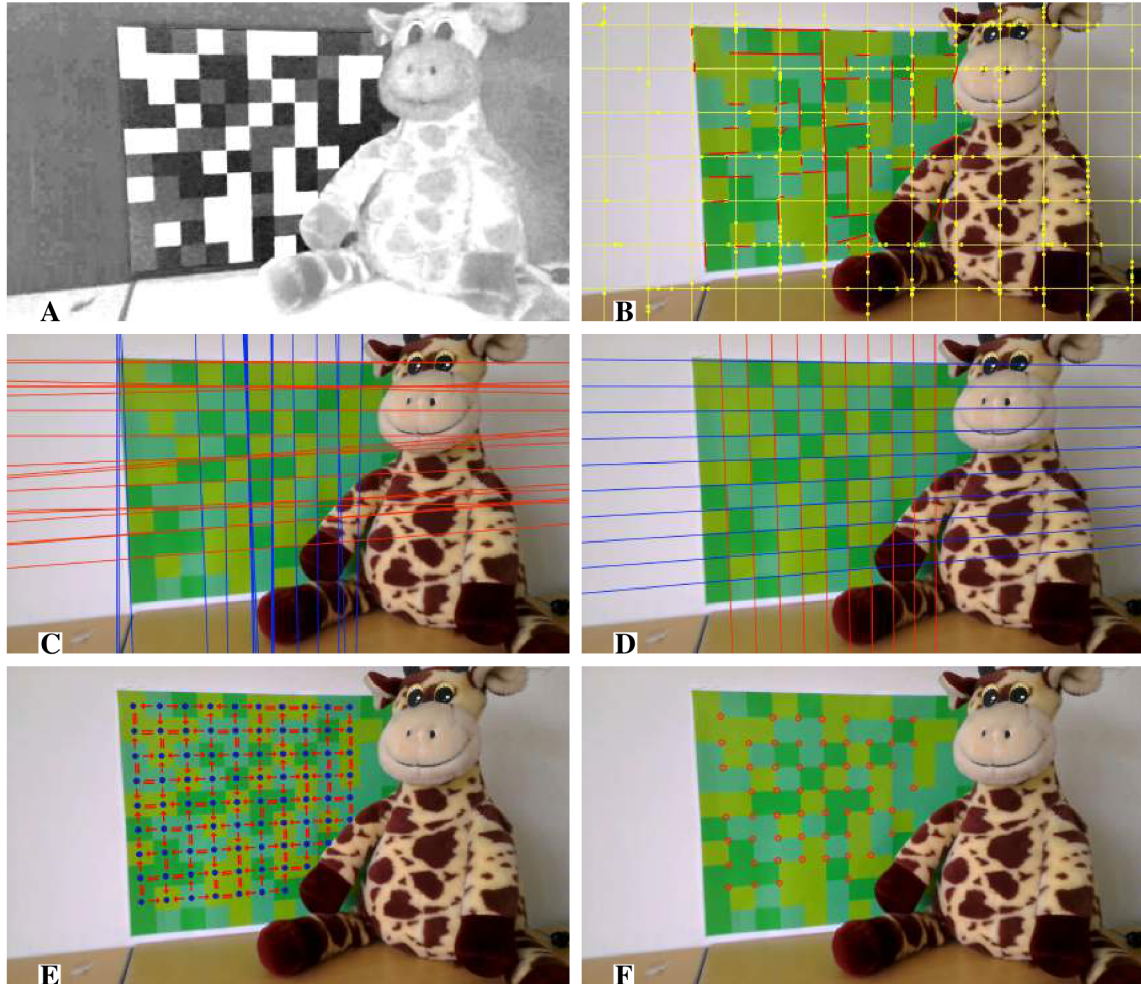


Figure 6.1: Detection of the grid of squares composed of suitable shades of green. **A:** The  $YC_bC_r$  image is mapped to grayscale for the detection algorithm. **B:** The grayscale image is processed in very sparse scanlines (for better visualization we use the source image). On each scanline, edges are detected (yellow points) and extended to edgels (red lines). **C:** The edgels are grouped into two dominant groups using RANSAC; two vanishing points are computed by hyperplane fitting. **D:** Based on the vanishing points, the optimal grid is fitted to the set of the edgels. **E:** Edges between the modules are classified. **F:** The annotated corner points are used for tracking and computing the 3D camera pose.

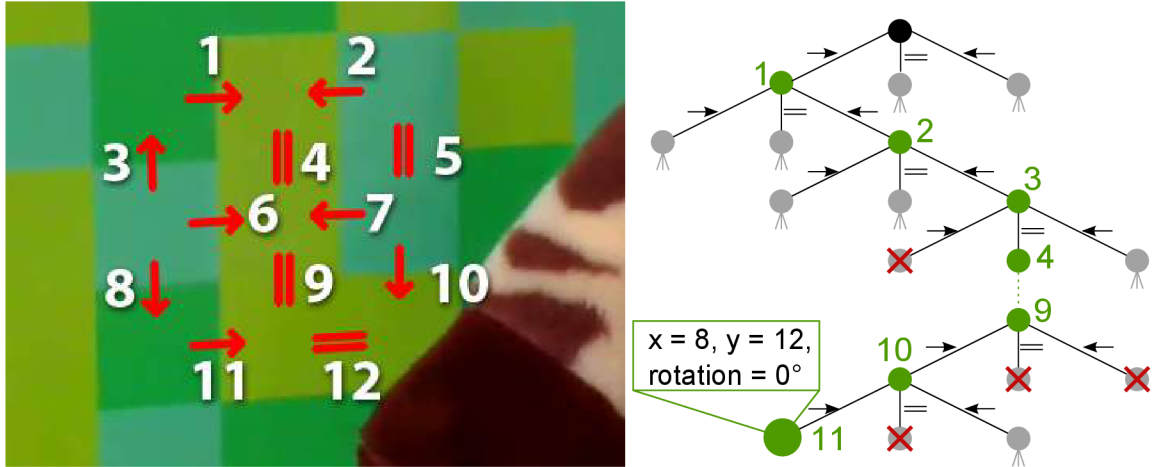


Figure 6.2: Localization within the Marker Field. The RGB decision tree is simplified so that the decisions are made based on a single value for each edge, not on all RGB components. **top:** The order in which the edges are visited. **bottom:** The decision tree. Leaves are either invalid or contain the location and orientation within the marker field.

smartphones already include high-precision IMU units. For cameras connected to PCs, an external sensor could be used (e.g. [45]).

The described camera pose estimation using Uniform Marker Fields in future work is also extendable to use multiple markers, since the image is processed in sub-windows. Alternatively the fast grid detection can be used from Chapter 4.

The grayscale marker design allows for very large marker fields even with 3 shades of gray (see Section 3.4.4). When using multiple marker fields, ideally, the similarity between the marker fields is minimal. To achieve this and to guarantee the uniqueness of detected sub-markers across all used marker fields, a single large marker field is separated into smaller marker fields. To be able to compute the camera pose from multiple marker fields, an initial scan of the scene is necessary to create a model of the scene containing the relative position and rotation of the fields.

### 6.3 Matchmoving With Shades of Green

Most existing fiducial marker designs use different marker features (typically edges) for localization of the marker and for recognition of its identity. Therefore, in order to assure reliable detection, the edges must exhibit a large contrast and none of them can be missed. On the contrary, Uniform Marker Fields use the same edges for detection of the marker and for its localization and recognition. That allows for the edges to be of low contrast, because even when a high fraction of the edges are missed, the marker field can still be detected and recognized. Low contrast edges are necessary for a marker to amalgamate into a sufficiently homogeneous green (or blue or other) surface for the background subtraction.

#### 6.3.1 Selection of Suitable Shades of Green

The marker field modules' color must be a compromise between usage of as-similar-as-possible colors for the chroma keying and colors different enough to detect the edges. The selection also depends on the selected chroma keying algorithm. Using, for example, the keying equation of first choice  $A = G - \max(B, R)$  [178], it seems suitable to vary  $R$  and  $B$  color components so that

$\max(B, R)$  stays the same (i.e. value  $A$  is constant for the whole marker field). However, more advanced algorithms are able to also deal with changing of the intensity of the green color (with constant hue). For example a method described by Jack [67] uses the  $YC_bC_r$  color model to achieve high quality matting robust against changing intensities (shadows). This is important due to the low dynamic range of videos from smart-phone cameras.

Contemporary mobile device cameras provide raw data in this  $YC_bC_r$  color space (or a variant of it). Choosing this colorspace to encode gradient direction between modules, initially means no information loss for the matting process due to conversion, and saves computational time. Encoding the marker into the  $C_bC_r$  channels provides more robustness against intensity changes (shadows) and white balance. Finally, higher quality real-time matting is possible. For matting, in our experiments we are using the method based on [67].

The detection algorithm does not rely on high-quality matting to filter out outliers in the foreground. Instead, we used a simplified approach, where  $(1 - C_b)(1 - C_r)$  is over a conservative threshold ( $C_b, C_r \in (-1, 1)$ ).

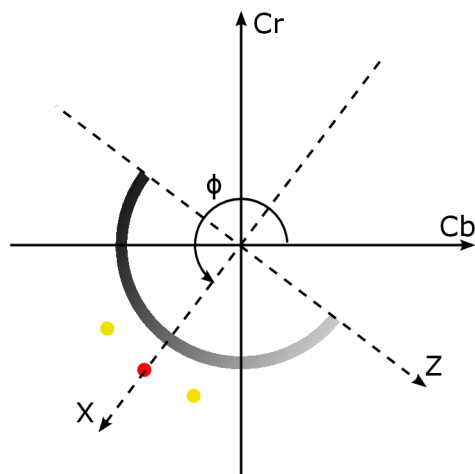


Figure 6.3: The normalized  $C_bC_r$  mapping to  $XZ$  space. The red dot represents the main keying color. The red and yellow dots are used for the UMF fields. The half arc demonstrates the mapping of the  $XZ$  space to grayscale for the detector.

### 6.3.2 Mapping

For the detection algorithm we encoded the edge direction between modules of the UMF into the  $C_bC_r$  channels. A good mapping is:

$$I_m(x, y) = \text{atan2} \frac{X(x, y)}{Z(x, y)}, \quad (6.1)$$

where  $I_m$  is the mapped image and  $X, Z$  are the rotated  $C_b, C_r$  channels respectively by the  $\phi$  angle of the average key color in the  $C_bC_r$  space (Figure 6.3). As an example for the choice of colors, in our experiments we used 3 hues with identical  $Y$  channel with  $20^\circ$  difference on the  $C_bC_r$  plane (yellow dots in Figure 6.3). Such a choice of the mapping is robust against changing intensities and to some extent also against image saturation levels on different smartphones. An alternative simple choice for mapping could be:

$$I_m(x, y) = 2(C_b - C_r) + 1, \quad (6.2)$$

with  $C_b, C_r \in (-1, 1)$ . We used the latter mapping in our experiments.

The detection algorithm then uses the resulting  $I_m$  mapped image as a grayscale image to detect UMF in further processing. Given the chosen mapping function, motion blur would only cause blurring in the mapped grayscale image. The UMF detection algorithm has proven to be highly robust against such distortion (Section 3.4.3).

### 6.3.3 Matting for Visualization

For camera pose reconstruction, a low-quality mask creation is sufficient to guide the detection algorithm to discard foreground pixels. For the user interface a high quality matting is done on the GPU, freeing resources for image processing on the CPU. Due to white balancing and other automatic image capture controls (generally present in commodity smartphones), having a predefined set of key colors is insufficient. We proposed to progressively optimize the exact key colors (by using GMM [137] in the experiments) once the marker was successfully detected in the image. Its layout can be used to sample image regions belonging to different shades in the UMF to predict the exact shades of green.

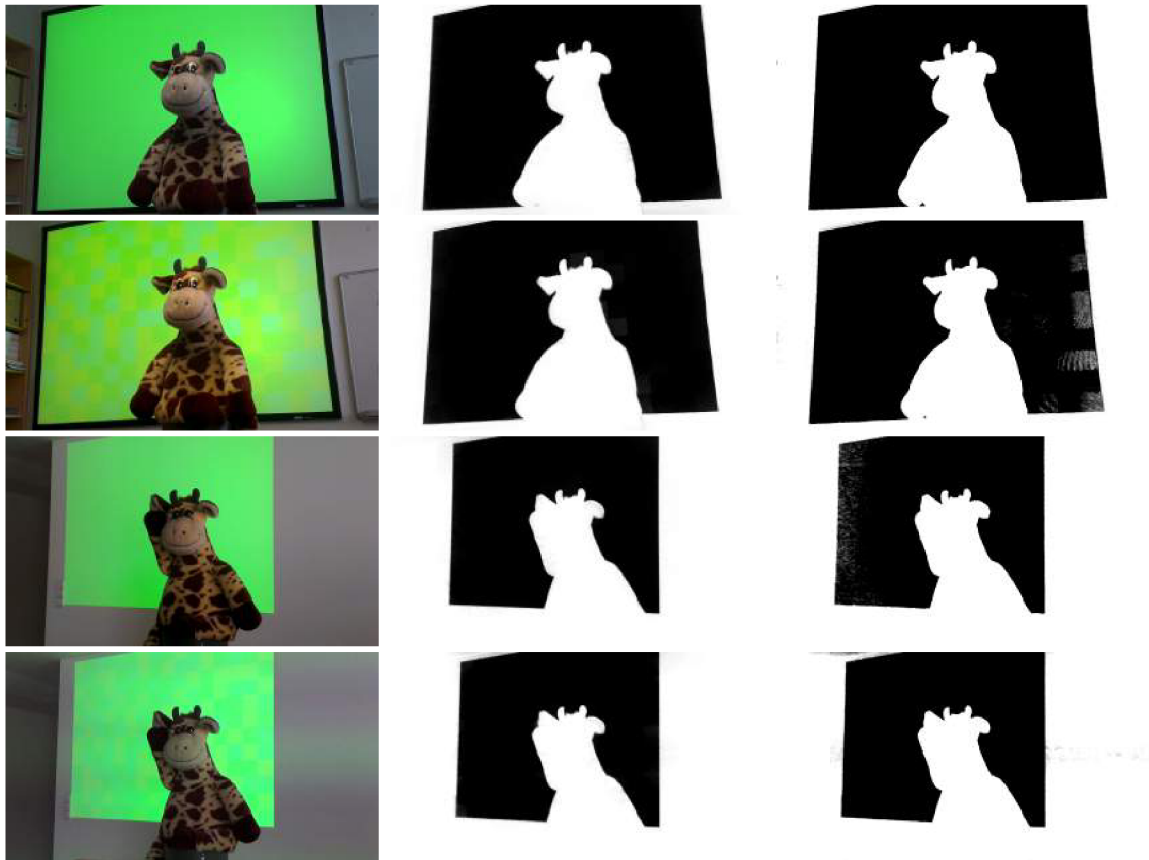


Figure 6.4: Sample images from the dataset. **Left:** original image. **Middle:** ground truth alpha mask (see Section 6.4.5). **Right:** our implementation using shaders. Odd rows showcase constant color and even rows show sample images with the encoded UMF marker.

### 6.3.4 Practical Set-Up: Projection

Classic green, blue or white canvas together with a projector can be used as a marker field canvas. Two options for organizing the set-up with a projector exist: front or back projection (Figure 6.5).

The main advantage of this approach is the opportunity to change the intensities/colors of the markers depending on the camera, lights and other conditions. Because of the additional light produced by the projector, this approach is not suitable for high-fidelity shots. However, it is fine for prototyping, simple scene shooting and instant tuning of the scene/setup.

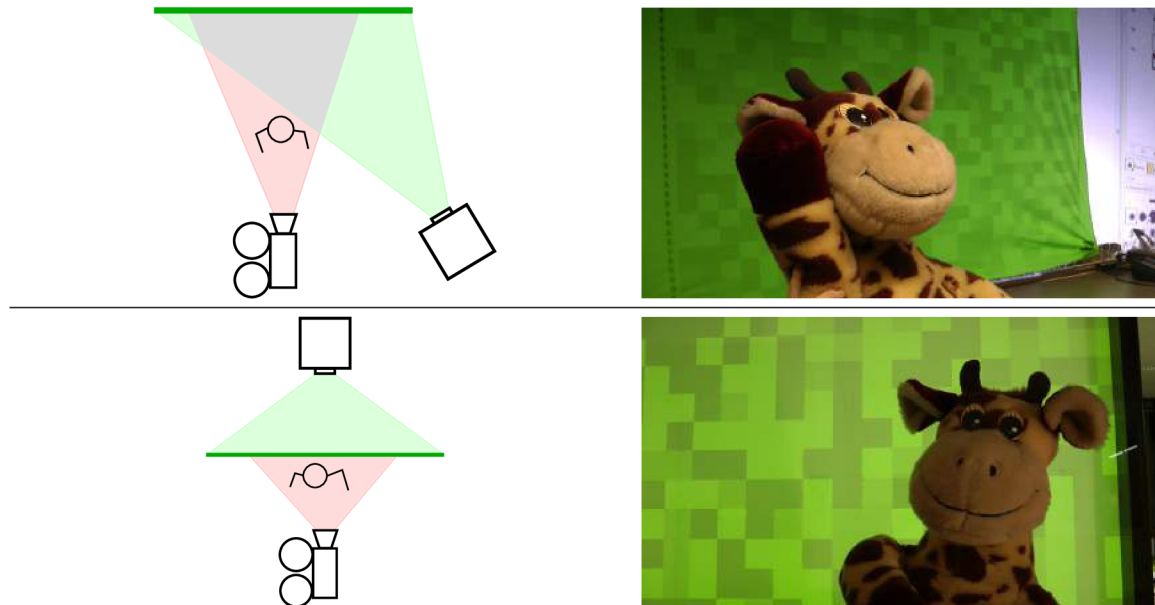


Figure 6.5: Projection of the marker field on green (or white or other) canvas. **top:** Front projection – the actor’s movement is limited because he must not interfere with the projection rays. **bottom:** Back projection – requires more space behind the greenscreen.

### 6.3.5 Practical Set-Up: Special Canvas

An alternative to the projection of the marker field is using a special canvas with the marker field printed on it. It is possible to synthesize marker fields of arbitrary dimensions by selecting suitable window shape/size and a proper number of colors (Section 3.4.4). The marker field does not have to cover the whole matting plate; instead, only a stripe or other shape can be covered by the marker field. Using several sheets requires mutual and world calibration which can be done completely automatically by quickly scanning over the scene with a camera.

## 6.4 Experiments and Evaluation

A thorough side-by-side comparison of the shades-of-grey marker field with alternative markers was given in Section 3.4.3. In Section 6.4.1 I describe our proof-of-concept implementation integrated into Unity 3D cross-platform game engine. The rest of the subsections show results of experiments of different aspects of our solution.

### 6.4.1 Unity 3D Plug-In

We targeted live streaming applications using a webcam for PC or an integrated camera on smart-phones. We created a plugin for Unity 3D, that integrates with our proof-of-concept implementation.

Unity 3D<sup>6</sup> is a free and cross-platform 3D game engine, that supports all our targeted platforms (see Figure 6.6).

For a real-world solution, even real-time performance of the detection algorithm causes significant lag due to other parts of the pipeline (frame acquisition, OS overhead, rendering overhead, texture copy between CPU and GPU). We proposed to process the frames asynchronously, which provided smooth video stream, but out-of-sync rendering placement. On the ARM platform, the camera stream was acquired directly from the operating system simultaneously for rendering (as texture) and detection (as bitmap). On the PC, the transfer speed between GPU and CPU memory was sufficient for real-time performance.

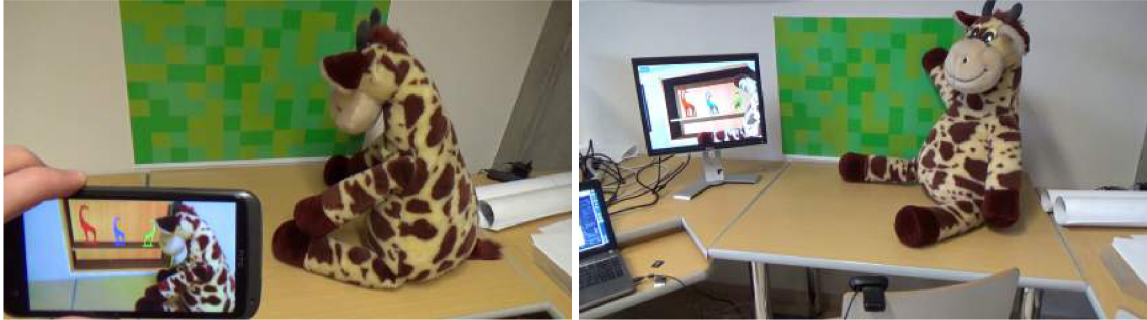


Figure 6.6: Camera pose estimation and chromakeying using green-screen markers on mobile and PC platforms.

To further free up CPU processing time, the costly high-quality chromakeying was done on the GPU using shaders. The detection algorithm used the simplified method described in Section 6.3.1.

Our plugin finally consists of a script attachable to the *camera object*, a custom shader material for the PC platform and native libraries for all supported platforms. The only inputs required from the user are the camera parameters for the PC platform. On Android OS, these parameters are provided by the operating system. Hence, our solution is extremely easy to use, suitable even for less experienced designers. For more advanced users, it should provide a very easy to setup tool to preview virtual scenes.

## 6.4.2 Detection Rates

To re-evaluate and compare the detection rates between grayscale and greenscreen markers, we created a small video dataset (16 videos). We used the setup with a projector projecting the marker from the front (Figure 6.5 top). We used two different projectors in combination with a smartphone ( $1280 \times 720 \text{ px}$ , 30Hz) and a hand-held dedicated video camera ( $1920 \times 1080 \text{ px}$ , 50Hz).

With tracking enabled, our algorithm was able to localize the camera in 99.9% of frames. Since tracking is a replacable part of the pipeline, Table 6.1 summarizes the percentage of frames for all combinations where the camera was successfully localized without tracking.

Camera/UMF marker	grayscale	greenscreen
smartphone	94.5 %	96.9 %
hand-held camera	97.7 %	87.7 %

Table 6.1: Detection rates between grayscale and greenscreen markers projected on canvas.

<sup>6</sup><http://unity3d.com>

With a smartphone camera, the detection rates were comparable between grayscale and green-screen markers. There was even a slight improvement, which might be caused by the reduced number of outlier edgels thanks to the background mask.

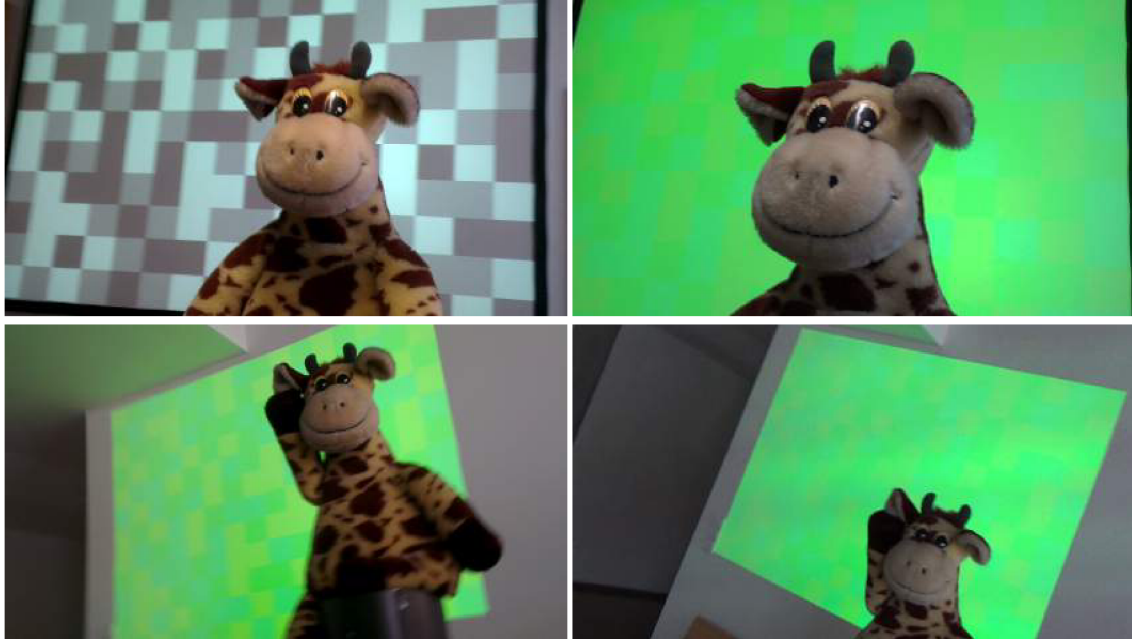


Figure 6.7: **Left:** Frames acquired with mobile camera with grayscale (top) and greenscreen (bottom) UMF. **Right:** Frames acquired with hand-held dedicated video camera. Notice the very low-contrast greenscreen UMF compared to the frames from the mobile camera. There is also visible banding on the right-bottom image (hand-held camera with second projector).

With the dedicated camera, the detection rates of the greenscreen UMF were significantly worse. This was caused by two specific issues. The dedicated camera did not do as aggressive white-balancing as the smartphone camera, hence the contrast of the greenscreen marker was significantly lower. Secondly, the frame-rate collision between one of the projectors and the camera caused banding and flickering (Figure 6.7 bottom right).

### 6.4.3 Computational Complexity

For the speed performance evaluations, we tested our solution for preview purposes with VGA ( $640 \times 480$ ) resolution camera stream. For the tracking, the algorithm used a sub-sampled resolution of  $320 \times 240$ . Since contemporary cameras provide sub-sampled color channels ( $C_b, C_r$  channels) used in our mapping to grayscale representation, sub-sampling should theoretically not cause any loss in the detection precision. We tested our solution both on PC (Intel(R) Core(TM) i7 2.2 GHz) and ARM platform (ARMv7 Processor rev. 9, 1.5 GHz).

For our experiments, we used our Unity plugin (Section 6.4.1) with a basic 3D scene. The Android implementation was running at 33.7 FPS on the GPU, and was processing frames (sent at 13 FPS to the CPU) asynchronously in 23.4 ms including Java overhead (theoretical 40 FPS). On the desktop PC, the web-camera was providing the VGA video stream at 30 FPS and each frame was processed by the detector in 6.4 ms on average. The measurement results are shown in Table 6.2 (*chroma*: the chromakeying process for the detector; *tracking and detection (t&d)*: the detection and tracking average time; *camera pose*: camera pose estimation based on the found matches).



The times include conversion from  $RGB$  to  $YC_bC_r$  for the PC and communication overhead from native to managed code for the ARM platform.

The main part of computational time on the ARM platform ( $\sim 63\%$ ) was taken by simple image manipulation: gradient computation and gaussian blur for the tracker, sub-sampling, matting mask, and mapping computations. We used the Halide language [131, 132] to create a solution for these tasks with more optimized memory access patterns. On the PC platform, this led to  $\sim 41\%$  speed improvement overall, with the chroma mapping being  $18\times$  faster. On the ARM platform, the speed improvement was  $\sim 8\%$  overall.

Platform	total	(chroma	t&d	cam.)
PC	<b>10.7</b>	3.7	1.9	1.4
PC with Halide	<b>6.3</b>	0.2	1.9	1.4
ARM	<b>25.3</b>	4.8	14.0	2.1
ARM with Halide	<b>23.4</b>	4.1	12.8	2.1

Table 6.2: Breakdown of the processing time in milliseconds. for VGA video.

#### 6.4.4 Preview Precision

To evaluate the camera pose precision for the preview use-case, we created a second small video dataset. The videos were shot with the smartphone camera (720p, 30Hz) from a 2-5 m distance from the canvas. We calibrated the smartphone camera including camera distortion for maximum precision. As reported earlier in Section 3.4.3, UMF detection algorithm outperforms alternative marker based solutions. We used our detector without any optimization and with precise calibration to establish a reliable reference for each frame.

To simulate the video stream processed by the detector on the mobile platform, we scaled down and cropped each video to VGA resolution. As calibration, we only used the camera  $fovy$  defined by the manufacturer. The median difference in the detected camera angle was  $1.5^\circ$  and the median distance from the reference camera pose was 5.91 *cm*.

#### 6.4.5 Matting Precision

In this section, I describe the conducted tests if the presence of the UMF in the greenscreen degrades the matting performance with our algorithm. In the experiment, we projected both plain green color and UMF encoded in the shades of green on the canvas and took images with both smartphone and dedicated cameras. We evaluated the precision of our matting algorithm using GPU shaders with reference to a state-of-the-art alpha-matting approach (KNNMatting [26] with manual annotation, see Figure 6.8). We quantified the error in the resulting alpha masks as the standard deviation of transparency values  $[0, 100)$ .

There was only a small difference in precision for the plain green color (standard deviation 3.62) and with the UMF marker present (standard deviation 4.5). This shows that using the green UMF as the background does not significantly decrease the segmentation compared to solid color. However, global methods (typically based on graph cuts) perform better on boundaries and on surfaces with bounced green light, at the price of much higher computational complexity.

#### 6.4.6 Sensitivity to Edge Contrast

For the greenscreen to be segmented reliably, the various green colors must be close enough to a common shade, i.e. the contrast on the edges between the neighboring squares of the marker field

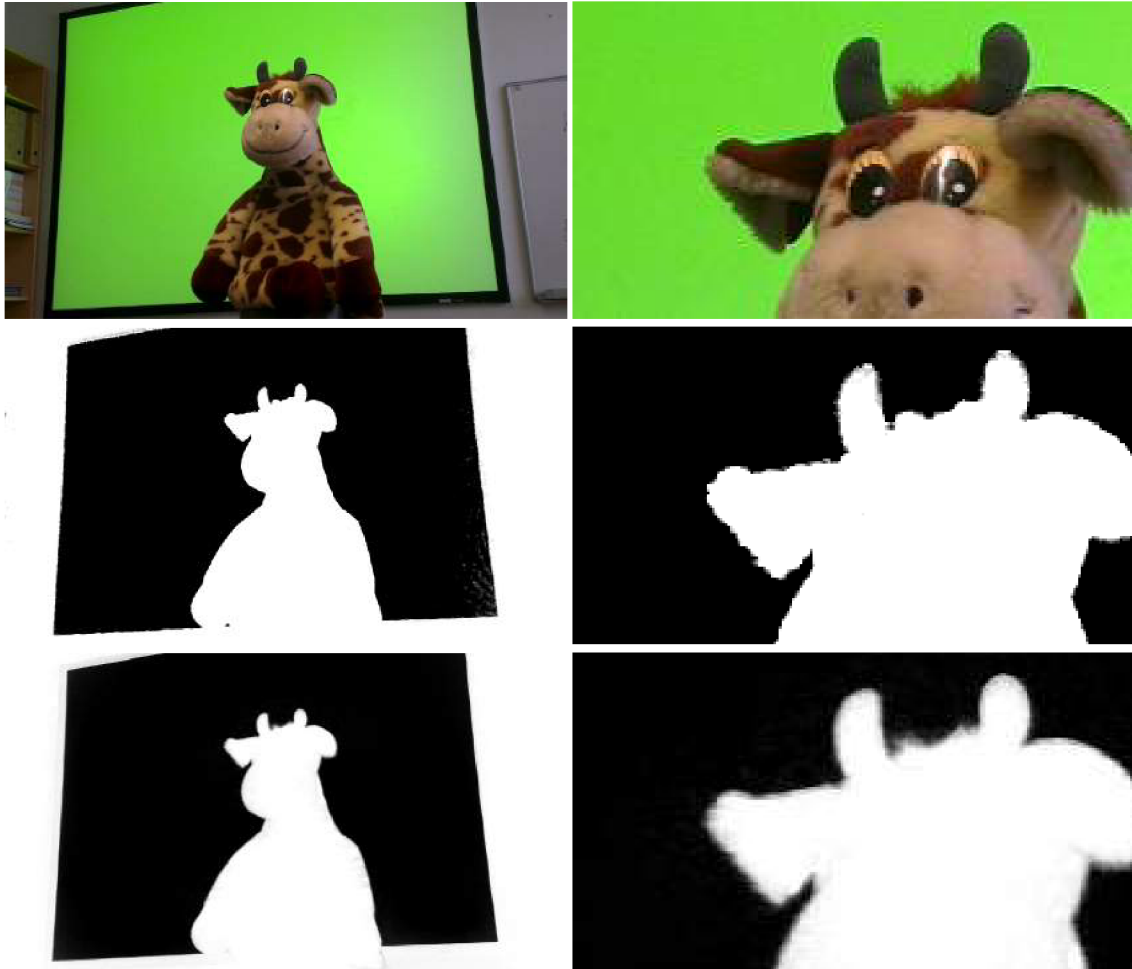


Figure 6.8: **top:** The original image from the camera. **middle:** Matting results from GPU shaders. **bottom:** Reference alpha mask acquired using KNNmatting with manual annotation.

must be low. At the same time, the marker field must be reliably recognized. An assumption fundamental to the viability of our approach is that these two contradictory requirements must be satisfied at the same time.

Figure 6.9 shows the results of our experiment targeted on this issue. We printed out multiple variants of a single generated marker field ( $14 \times 10$ ) with the green colors altered to change their “contrast”. We shot short video sequences with the hand-held dedicated camera observing the marker field from varying angles at normal daylight conditions in an office setting. The detection rate is the fraction of correctly recognized video frames without tracking. The reported graph therefore depicts a pessimistic look on the detection performance. With the tracking feature enabled and even with 75% detection rate, camera pose tracking is theoretically restored with 99.9% probability within the next 5 video frames.

## 6.5 Results

We were successful in creating an instant and reliable matchmoving solution for chromakeying-based film effects. It was based on the multi-color uniform marker fields. We proposed methodology

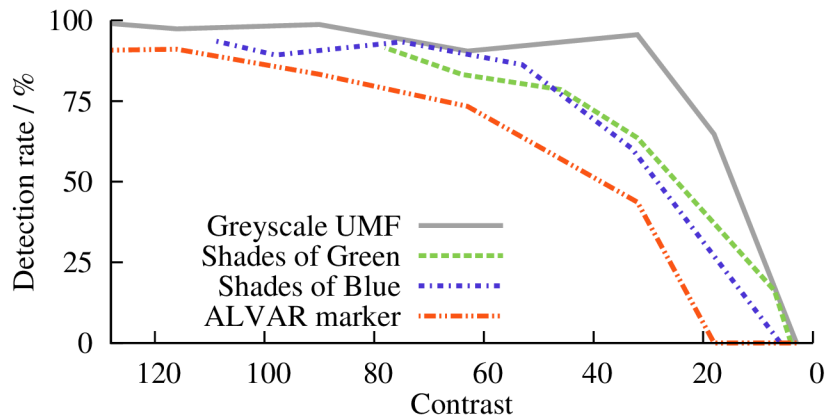


Figure 6.9: Detection rate as it depends on the marker “contrast”. **contrast 255** = extreme colors (darkest, brightest) are 255 units apart in the color channels. **contrast 0** = the colors in the field are all identical. Good news is that as assumed, the detection performance drops at reasonably low values of the “contrast” between the shades. The experimentally found tipping point ( $\sim 40$ ) intensity difference is a good choice for practical greenscreen canvases.

to design marker fields of shades-of-green (blue or any other practical color). Such marker fields can be detected, recognized (for accurate camera pose estimation), and precisely segmented out in order to be replaced by rendered content. The experiments show that the process is both very efficient (the applications can easily run on today’s ultramobile mid-range processors) and at the same time it is inexpensive and simple to use.



## Chapter 7

# Conclusion

This thesis presents in its core an efficient camera pose estimation for real-time augmented reality applications. Most importantly, I introduced the concept of Uniform Marker Fields, which overcome limitations of existing marker designs. Uniform Marker Fields are a marker solution with a checkerboard structure with unique sub-windows of size  $n^2$  in every rotation. Such marker fields can cover large areas and still be detectable from small distance. The marker field design, contrary to conventional markers (ARToolKit, ARTag), does not distinguish between localization and identification features.

The detection algorithm proposed in this thesis was designed to be highly efficient and have a small memory footprint. The algorithm relies on long edgels (connected edge pixels) to estimate two vanishing points and recover the marker fields' grid structure by line parametrization. The inter-relation between neighbor modules is used to recover the marker fields rotation and position. This information provides 2D-3D correspondences for module corners, which can be used to compute a full 6 degrees of freedom camera pose. Comprehensive evaluation shows that the described algorithm for Uniform Marker Fields was faster than alternative marker-based approaches with comparable or better performance.

The efficient implementation of the detection algorithm and its various modifications enabled real-time camera pose tracking even on mid-range commodity smartphones. I have demonstrated this on several applications of Uniform Marker Fields. These included the on-screen markers used for document reaccess, task migration, and other user-centric tasks. The last important application was the use of the UMF in film-making domain as a structured and "intelligent" green screen.

My PhD research was centered around the topic of camera pose estimation (mostly based on markers), but it naturally visited other related fields, such as mobile app development, algorithmic optimization, rendering, human-computer interactions, and a few distinct and specific applications. I would also like to give credit to my colleagues for providing knowledge and assistance in the vast research fields associated with this thesis.

Coming into this research, I was focusing on computer graphics and rendering during my earlier studies. I was pleased with the opportunity to further my education and broaden my horizon. Not only was I fortunate enough to dive into several fresh and rapidly developing research fields, I also contributed my own ideas and created original solutions. My work is already cited and used in several publications authored by other researchers.

# Bibliography

- [1] Aztec code bar code symbology specification. *ISO/IEC 24778:2008*, 2008.
- [2] ALVAR tracking subroutines library web page, 2012.  
<http://www.vtt.fi/multimedia/alvar.html>.
- [3] Qr-code bar code symbology specification. *ISO/IEC 18004:2015*, 2015.
- [4] Jonas Alftan. Robust detection of two-dimensional barcodes in blurry images. Master's thesis, KTH Computer Science and Communication, Stockholm, SE, 2008.
- [5] Abbas M. Alhakim. A simple combinatorial algorithm for De Bruijn sequences. *American Mathematical Monthly*, 117(8):728–732, 2010.
- [6] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5):578 – 589, may 2003.
- [7] S. Arnould, G.J. Awcock, and R. Thomas. Remote bar-code localisation using mathematical morphology. In *Seventh International Conference on Image Processing and Its Applications*, volume 2, pages 642 –646 vol.2, 1999.
- [8] Bradley Atcheson, Felix Heide, and Wolfgang Heidrich. CALTag: High precision fiducial markers for camera calibration. In *Proc. VMV*, 2010.
- [9] Ronald T Azuma. A survey of augmented reality. *Presence-Teleoperators and Virtual Environments*, 6(4):355–385, 1997.
- [10] Elizabeth Bales, Timothy Sohn, and Vidya Setlur. Planning, apps, and the high-end smartphone: exploring the landscape of modern cross-device reaccess. In *Proceedings of the 9th international conference on Pervasive computing*, Pervasive'11, pages 1–18, Berlin, Heidelberg, 2011. Springer-Verlag.
- [11] Rafael Ballagas, Jan Borchers, Michael Rohs, and Jennifer G. Sheridan. The smart phone: A ubiquitous input device. *IEEE Pervasive Computing*, 5(1):70–77, January 2006.
- [12] Rafael Ballagas, Michael Rohs, and Jennifer G. Sheridan. Sweep and point and shoot: phonecam-based interactions for large public displays. In *CHI '05 Extended Abstracts on Human Factors in Computing Sys.*, 2005.
- [13] E. Bardram. Activity-based computing: support for mobility and collaboration in ubiquitous computing. *Personal Ubiquitous Comput.*, 9(5):312–322, September 2005.
- [14] Dominikus Baur, Sebastian Boring, and Steven Feiner. Virtual projection: exploring optical projection as a metaphor for multi-device interaction. In *Annual Conference on Human Factors in Computing Systems*, 2012.

- [15] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.
- [16] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.
- [17] Peter Betko. Synthesis and detection of color markers for augmented reality. Master’s thesis, Brno University of Technology, Faculty of Information Technology, 2013.
- [18] L.F.F. Belussi and N.S.T. Hirata. Fast QR code detection in arbitrarily acquired images. In *Conference on Graphics, Patterns and Images, SIBGRAPI 2011*, pages 281 –288, aug. 2011.
- [19] Sebastian Boring, Manuela Altendorfer, Gregor Broll, Otmar Hilliges, and Andreas Butz. Shoot & copy: phonecam-based information transfer from public displays onto mobile phones. In Peter Han Joo Chong and Adrian David Cheok, editors, *Mobility Conference*, pages 24–31. ACM, 2007.
- [20] Sebastian Boring, Dominikus Baur, Andreas Butz, Sean Gustafson, and Patrick Baudisch. Touch projector: mobile interaction through video. In *SIGCHI Conference on Human Factors in Computing Systems*, 2010.
- [21] Sebastian Boring, Sven Gehring, Alexander Wiethoff, Anna Magdalena Blöckner, Johannes Schöning, and Andreas Butz. Multi-user interaction on media facades through live video on mobile devices. In *SIGCHI Conf. Human Factors in Comput. Sys.*, 2011.
- [22] Felix Bork, Bernhard Fuers, Anja-Katharina Schneider, Francisco Pinto, Christoph Graumann, and Nassir Navab. Auditory and visio-temporal distance coding for 3-dimensional perception in medical augmented reality. In *Mixed and Augmented Reality (ISMAR), 2015 IEEE International Symposium on*, pages 7–12. IEEE, 2015.
- [23] Peter Brandl, Clifton Forlines, Daniel Wigdor, Michael Haller, and Chia Shen. Combining and measuring the benefits of bimanual pen and direct-touch interaction on horizontal interfaces. In *Working Conf. Adv. Visual Interf.*, 2008.
- [24] J Burns and C J Mitchell. Coding schemes for two-dimensional position sensing. *Institute of Mathematics and Its Applications Conference Series*, 45:31, 1993.
- [25] Tsung-Hsiang Chang and Yang Li. Deep Shot: a framework for migrating tasks across devices using mobile phone cameras. In *Proc. SIGCHI*, 2011.
- [26] Qifeng Chen, Dingzeyu Li, and Chi-Keung Tang. Knn matting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(9):2175–2188, Sept 2013.
- [27] Y. Cho, J. Lee, and U. Neumann. A multi-ring fiducial system and an intensity-invariant detection method for scalable augmented reality. In *Proceedings of the international workshop on Augmented reality : placing artificial objects in real scenes: placing artificial objects in real scenes*, IWAR ’98, pages 147–165, Natick, MA, USA, 1999. A. K. Peters, Ltd.
- [28] Yung-Yu Chuang, B. Curless, D.H. Salesin, and R. Szeliski. A Bayesian approach to digital matting. In *CVPR 2001*, 2001.

- [29] Karen Church and Barry Smyth. Understanding mobile information needs. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, MobileHCI '08, pages 493–494, New York, NY, USA, 2008. ACM.
- [30] A.I. Comport, E. Marchand, and F. Chaumette. A real-time tracker for markerless augmented reality. In *ACM/IEEE Int. Symp. on Mixed and Augmented Reality, ISMAR'03*, pages 36–45, Tokyo, Japan, October 2003.
- [31] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of CVPR 2005*, volume 1, pages 886–893, 2005.
- [32] David Dearman and Jeffery S. Pierce. It's on my other computer!: computing with multiple devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 767–776, New York, NY, USA, 2008. ACM.
- [33] T. Dobbert. *Matchmoving: The Invisible Art of Camera Tracking*. Wiley Desktop Editions. John Wiley & Sons, 2006.
- [34] M. Donoser, P. Kotschieder, and H. Bischof. Robust planar target tracking and pose estimation from a single concavity. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 9–15, oct. 2011.
- [35] M. Dubská, I. Szentandrás, M. Zachariáš, and A. Herout. Poor man's SimulCam: Real-time and effortless matchmoving. In *12th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 249–250, Oct 2013.
- [36] Markéta Dubská, Adam Herout, and Jiří Havel. Real-time precise detection of regular grids and matrix codes. *Journal of Real-Time Image Processing*, 11(1):193–200, 2013.
- [37] Markéta Dubská, Adam Herout, Roman Juránek, and Jakub Sochor. Fully automatic roadside camera calibration for traffic surveillance. *IEEE Transactions on Intelligent Transportation Systems*, 2014(1):1–10, 2014.
- [38] Tuvi Etzion. Constructions for perfect maps and pseudorandom arrays. *Information Theory, IEEE Transactions on*, 34(5):1308–1316, 1988.
- [39] CT Fan, SM Fan, SL Ma, and MK Siu. On de Bruijn arrays. *Ars Combinatoria*, 19(MAY):205–213, 1985.
- [40] Olivier Faugeras. *Three-dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Cambridge, MA, USA, 1993.
- [41] M. Fiala. ARTag, a fiducial marker system using digital techniques. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 590–596, Washington, DC, USA, 2005. IEEE Computer Society.
- [42] Mark Fiala. Magic mirror system with hand-held and wearable augmentations. *IEEE Virtual Reality (VR)*, 0:251–254, 2007.
- [43] Mark Fiala. Designing highly reliable fiducial markers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32:1317–1324, July 2010.
- [44] J. Foster. *The Green Screen Handbook: Real-World Production Techniques*. Wiley, 2010.



- [45] Eric Foxlin, Leonid Naimark, et al. Vis-tracker: A wearable vision-inertial self-tracker. *VR*, 3:199, 2003.
- [46] John GF Francis. The qr transformation a unitary analogue to the lr transformation—part 1. *The Computer Journal*, 4(3):265–271, 1961.
- [47] Yasutaka Fujimoto, Ross T Smith, Takafumi Taketomi, Go Yamamoto, Jun Miyazaki, Haruhisa Kato, and Bruce H Thomas. Geometrically-correct projection-based texture mapping onto a deformable object. *Visualization and Computer Graphics, IEEE Transactions on*, 20(4):540–549, 2014.
- [48] Priyanka Gaur and Shamik Tiwari. Recognition of 2D barcode images using edge detection and morphological operation. *International Journal of Computer Science and Mobile Computing*, 3(4):1277–1282, 2014.
- [49] Basil Gordon. On the existence of perfect maps (corresp.). *Information Theory, IEEE Transactions on*, 12(4):486–487, 1966.
- [50] L. Gruber, T. Richter-Trummer, and D. Schmalstieg. Real-time photometric registration from arbitrary geometry. In *Proceedings of the 11th IEEE International Symposium on Mixed and Augmented Reality*, pages 119–128, 2012.
- [51] Nate Hagbi, Oriël Bergig, Jihad El-Sana, and Mark Billinghurst. Shape recognition and pose estimation for mobile augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, 17(10):1369–1379, 2011.
- [52] Michal Haindl and Stanislav Mikes. Colour texture segmentation using modelling approach. In *ICAPR (2)*, pages 484–491, 2005.
- [53] Mike Harding, Oliver Storz, Nigel Davies, and Adrian Friday. Planning ahead: techniques for simplifying mobile service use. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, HotMobile '09, pages 13:1–13:6, New York, NY, USA, 2009. ACM.
- [54] Chris Harris and Carl Stennett. Rapid – a video rate object tracker. In *BMVC*, pages 1–6, 1990.
- [55] J. Hart. *The art of the storyboard: storyboarding for film, TV, and animation*. Focal Pr, 1999.
- [56] Stephen G. Hartke. Binary De Bruijn cycles under different equivalence relations. *Discrete Mathematics*, 215:93 – 102, 2000.
- [57] RichardI. Hartley. Self-calibration from multiple views with a rotating camera. In Jan-Olof Eklundh, editor, *Computer Vision — ECCV '94*, volume 800 of *Lecture Notes in Computer Science*, pages 471–478. Springer Berlin Heidelberg, 1994.
- [58] Leigh Herbert, Nick Pears, Daniel Jackson, and Patrick Olivier. Mobile device and intelligent display interaction via scale-invariant image feature matching. In *PECCS*, 2011.
- [59] A. Herout, M. Zachariáš, M. Dubska, and J. Havel. Fractal marker fields: No more scale limitations for fiduciary markers. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 285–286, Nov 2012.

- [60] Adam Herout, István Szentandrás, Michal Zachariáš, Markéta Dubská, and Rudolf Kajan. Five shades of grey for fast and reliable camera pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1384–1390. IEEE Computer Society, 2013.
- [61] M. Hirzer. Marker detection for augmented reality applications. Technical report, Inst. for Computer Graphics and Vision, Graz University of Technology, Austria, 2008.
- [62] Huaqiao Hu, Wenhuan Xu, and Qiang Huang. A 2D barcode extraction method based on texture direction analysis. In *Fifth International Conference on Image and Graphics, ICIG '09.*, pages 759–762, sept. 2009.
- [63] Glenn Hurlbert and Garth Isaak. New constructions for De Bruijn tori. In *Designs, Codes and Cryptography* 6, pages 47–56, 1995.
- [64] Glenn Hurlbert and Garth Isaak. On the De Bruijn torus problem. *J. Combin. Theory Ser. A*, 64:50–62, 1995.
- [65] Jae-In Hwan, Elizabeth Adelia Widjojo, Seungmin Rho, Youna Lee, Jinwoo Lee, and Junho Kim. [demo] mobile binocular augmented reality system for museums. In *Mixed and Augmented Reality (ISMAR), 2015 IEEE International Symposium on.* IEEE, 2015.
- [66] J. Jachnik, R. A. Newcombe, and A. J. Davison. Real-time surface light-field capture for augmentation of planar specular surfaces. In *Proceedings of the 11th IEEE International Symposium on Mixed and Augmented Reality*, pages 91–97, 2012.
- [67] Keith Jack. *Video Demystified: A Handbook for the Digital Engineer, 5th Edition.* Newnes, Newton, MA, USA, 5th edition, 2007.
- [68] A.K. Jain and Y. Chen. Bar code localization using texture analysis. In *Second International Conference on Document Analysis and Recognition*, pages 41–44, oct 1993.
- [69] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. PointRight: Experience with flexible input redirection in interactive workspaces. In *UIST*, 2002.
- [70] Brett R. Jones, Hrvoje Benko, Eyal Ofek, and Andrew D. Wilson. Illumiroom: Peripheral projected illusions for interactive experiences. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '13*, pages 869–878, New York, NY, USA, 2013. ACM.
- [71] Radim Kříž. Uniform marker field on a cylinder. Master’s thesis, Brno University of Technology, Faculty of Information Technology, 2013.
- [72] R. Kajan, A. Herout, I. Szentandrás, and M. Zachariáš. On-screen marker fields for reliable screen-to-screen task migration. In *SouthCHI 2013: International Conference on Human Factors in Computing and Informatics*, pages 692–710, 2013.
- [73] Rudolf Kajan, István Szentandrás, Adam Herout, and Alena Pavelková. Reliable and unobtrusive inter-device collaboration by continuous interaction. In *Journal of WSCG*, pages 95–103. University of West Bohemia in Pilsen, 2014.

- [74] 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*, TEI '07, pages 69–74, New York, NY, USA, 2007. ACM.
- [75] Amy K. Karlson, Shamsi T. Iqbal, Brian Meyers, Gonzalo Ramos, Kathy Lee, and John C. Tang. Mobile taskflow in context: a screenshot study of smartphone usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2009–2018, New York, NY, USA, 2010. ACM.
- [76] H. Kato and M. Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *IWAR'99*, 1999.
- [77] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana. Virtual object manipulation on a table-top AR environment. In *IEEE and ACM International Symposium on Augmented Reality, 2000.(ISAR 2000). Proceedings*, pages 111–119, 2000.
- [78] G. Klein and T. Drummond. Robust visual tracking for non-instrumented augmented reality. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '03, pages 113–, Washington, DC, USA, 2003. IEEE Computer Society.
- [79] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.
- [80] Gabriel Klimek and Zoltan Vamossy. QR code detection using parallel lines. In *Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on*, pages 477–481. IEEE, 2013.
- [81] M. Krichenbauer, G. Yamamoto, T. Taketomi, C. Sandor, and H. Kato. Towards augmented reality user interfaces in 3d media production. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 23–28, Sept 2014.
- [82] Celine Latulipe and Craig S. Kaplan. Bimanual and unimanual image alignment: an evaluation of mouse-based techniques. In *ACM UIST*, 2005.
- [83] B.-J. Lee, J.-S. Park, and M. Sung. Vision-based real-time camera matchmoving with a known marker. In *Entertainment Computing - ICEC 2006*. 2006.
- [84] V. Lepetit, F. Moreno-Noguer, and P. Fua. EPnP: An accurate  $O(N)$  solution to the PnP problem. *Int. J. Comput. Vision*, 81(2):155–166, February 2009.
- [85] S. Leutenegger, M. Chli, and R. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *Proceedings of the IEEE International Conference on Computer Vision*, 2011.
- [86] Qiong Liu, Paul McEvoy, and Cheng-Jia Lai. Mobile camera supported document redirection. In *Proceedings of the 14th annual ACM international conference on Multimedia*, MULTIMEDIA '06, pages 791–792, New York, NY, USA, 2006. ACM.
- [87] Yue Liu, Ju Yang, and Mingjun Liu. Recognition of QR code with mobile phones. In *Chinese Control and Decision Conference, CCDC 2008*, pages 203 –206, july 2008.

- [88] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.
- [89] C.-P. Lu, G.D. Hager, and E. Mjolsness. Fast and globally convergent pose estimation from video images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(6):610–622, jun 2000.
- [90] Q.-T. Luong and O.D. Faugeras. Self-calibration of a moving camera from point correspondences and fundamental matrices. *International Journal of Computer Vision*, 22(3):261–289.
- [91] S. Ma. A note on binary arrays with a certain window property. *Information Theory, IEEE Transactions on*, 30(5):774–775, Sep 1984.
- [92] I. Scott MacKenzie, Tatu Kauppinen, and Miika Silfverberg. Accuracy measures for evaluating computer pointing devices. In *CHI*, 2001.
- [93] F Jessie MacWilliams and Neil JA Sloane. Pseudo-random sequences and arrays. *Proceedings of the IEEE*, 64(12):1715–1729, 1976.
- [94] Stéphane Magnenat, Dat Tien Ngo, Fabio Zund, Mattia Ryffel, Gioacchino Noris, Gerhard Rothlin, Alessia Marra, Maurizio Nitti, Pascal Fua, Markus Gross, et al. Live texturing of augmented reality characters from colored drawings. *Visualization and Computer Graphics, IEEE Transactions on*, 21(11):1201–1210, 2015.
- [95] E. Marchand, H. Uchiyama, and F. Spindler. Pose estimation for augmented reality: a hands-on survey. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.
- [96] Keith M. Martin, Z.D. Dai, M.J.B. Robshaw, and P.R. Wild. *Orientable Sequences*, pages 97–115. 1993.
- [97] P. McIlroy, Sh. Izadi, and A. Fitzgibbon. Kinectrack: Agile 6-DoF tracking using a projected dot pattern. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 23–29, nov. 2012.
- [98] Gerard Medioni and Sing Bing Kang. *Emerging topics in computer vision*. Prentice Hall PTR, 2004.
- [99] C.J. Mitchell. Aperiodic and semi-periodic perfect maps. *Information Theory, IEEE Transactions on*, 41(1):88–95, jan 1995.
- [100] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104:90–126, 2006.
- [101] T. Moons, L. Van Gool, M. Proesmans, and E. Pauwels. Affine reconstruction from perspective image pairs with a relative object-camera translation in between. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(1):77–83, Jan 1996.
- [102] R.A. Morano, C. Ozturk, R. Conn, S. Dubin, S. Zietz, and J. Nissano. Structured light using pseudorandom codes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):322–327, Mar 1998.

- [103] W. O. J. Moser Morton Abramson. More birthday surprises. *The American Mathematical Monthly*, 77(8):856–858, 1970.
- [104] R. Muniz, L. Junco, and A. Otero. A robust software barcode reader using the hough transform. In *International Conference on Information Intelligence and Systems*, pages 313–319, 1999.
- [105] Miguel A. Nacenta, Samer Sallam, Bernard Champoux, Sriram Subramanian, and Carl Gutwin. Perspective cursor: Perspective-based interaction for multi-display environments. In *CHI*, 2006.
- [106] Tomohiro Nakai, Koichi Kise, and Masakazu Iwamura. *Document Analysis Systems VII: 7th International Workshop, DAS 2006, Nelson, New Zealand, February 13-15, 2006. Proceedings*, chapter Use of Affine Invariants in Locally Likely Arrangement Hashing for Camera-Based Document Image Retrieval, pages 541–552. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [107] R.A. Newcombe, S. Lovegrove, and A.J. Davison. Dtam: Dense tracking and mapping in real-time. In *Proc. of the Intl. Conf. on Computer Vision (ICCV), Barcelona, Spain*, volume 1, 2011.
- [108] J. Newman, D. Ingram, and A. Hopper. Augmented reality in a wide area sentient environment. In *ISAR*, pages 77–86. IEEE Computer Society, 2001.
- [109] J. Newman, M. Wagner, M. Bauer, A. Macwilliams, T. Pintaric, D. Beyer, D. Pustka, F. Strasser, D. Schmalstieg, G. Klinker, and Technische Universität Wien. Ubiquitous tracking for augmented reality. In *In Proc. of International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pages 192–201, 2004.
- [110] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.
- [111] A.G. Balan O. Parvu. A method for fast detection and decoding of specific 2D barcodes. In *17th Telecommunications forum TELFOR 2009*, pages 1137 – 1140, 2009.
- [112] E. Ohbuchi, H. Hanaizumi, and L.A. Hock. Barcode readers using the camera device in mobile phones. In *Int. Conference on Cyberworlds (CW'04)*, pages 260 – 265, nov. 2004.
- [113] Timo Ojala and Matti Pietikäinen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32(3):477–486, 1999.
- [114] Peter Ondruska, Pushmeet Kohli, and Shahram Izadi. Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. *Visualization and Computer Graphics, IEEE Transactions on*, 21(11):1251–1258, 2015.
- [115] R. Orghidan, J. Salvi, M. Gordan, and B. Orza. Camera calibration using two or three vanishing points. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, pages 123–130, Sept 2012.
- [116] J. Orlosky, T. Toyama, K. Kiyokawa, and D. Sonntag. Modular: Eye-controlled vision augmentations for head mounted displays. *Visualization and Computer Graphics, IEEE Transactions on*, 21(11):1259–1268, Nov 2015.

- [117] T. Oskiper, S. Samarasekera, and R. Kumar. Multi-sensor navigation algorithm using monocular camera, imu and gps for large scale augmented reality. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 71–80, nov. 2012.
- [118] Mai Otsuki, Yuko Kamioka, Yuka Kitai, Mao Kanzaki, Hideaki Kuzuoka, and Hiroko Uchiyama. Please show me inside: Improving the depth perception using virtual mask in stereoscopic ar. In *SIGGRAPH Asia 2015 Emerging Technologies, SA '15*, pages 19:1–19:3, New York, NY, USA, 2015. ACM.
- [119] E. Ouaviani, A. Pavan, M. Bottazzi, E. Brunelli, F. Caselli, and M. Guerrero. A common image processing framework for 2D barcode reading. In *International Conference on Image Processing and Its Applications*, volume 2, pages 652–655, 1999.
- [120] Y. Oyamada, P. Fallavollita, and N. Navab. Single camera calibration using partially visible calibration objects based on random dots marker tracking algorithm. In *Proceedings of the 11th IEEE International Symposium on Mixed and Augmented Reality*, Atlanta, USA, November 2012.
- [121] D. Parikh and G. Jancke. Localization and segmentation of a 2D high capacity color barcode. In *IEEE Workshop on Apps. of Computer Vision (WACV)*, pages 1–6, 2008.
- [122] J. Park, S. You, and U. Neumann. Natural feature tracking for extendible robust augmented realities. In *International Workshop on Augmented Reality (IWAR)'98*, 1998.
- [123] Kenneth G. Paterson. New classes of perfect maps I. *Journal of Combinatorial Theory, Series A*, 73(2):302 – 334, 1996.
- [124] Alena Pavelková. Eye tracking during interaction with a screen. Master's thesis, Brno University of Technology, Faculty of Information Technology, 2013.
- [125] Jeffrey S. Pierce, Andrew S. Forsberg, Matthew J. Conway, Seung Hong, Robert C. Zeleznik, and Mark R. Mine. Image plane interaction techniques in 3d immersive environments. In *Proceedings of the 1997 symposium on Interactive 3D graphics, I3D '97*, pages 39–ff., New York, NY, USA, 1997. ACM.
- [126] Jeffrey S. Pierce and Jeffrey Nichols. An infrastructure for extending applications' user experiences across multiple personal devices. In *Proceedings of the 21st annual ACM symposium on User interface software and technology, UIST '08*, pages 101–110, New York, NY, USA, 2008. ACM.
- [127] Christian Pirchheim and Gerhard Reitmayr. Homography-based planar mapping and tracking for mobile phones. In *ISMAR*, 2011.
- [128] Alexander Páldy and Adam Herout. Advanced markers for augmented reality. *Proceedings of CESC G 2013: The 17th Central European Seminar on Computer Graphics*, 2013.
- [129] Parinya Punpongsanon, Daisuke Iwai, and Kosuke Sato. Softar: Visually manipulating haptic softness perception in spatial augmented reality. *Visualization and Computer Graphics, IEEE Transactions on*, 21(11):1279–1288, 2015.
- [130] L. Quan and Z. Lan. Linear n-point camera pose determination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(8):774 –780, aug 1999.

- [131] Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *ACM Trans. Graph.*, 31(4):32:1–32:12, July 2012.
- [132] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13*, pages 519–530, New York, NY, USA, 2013. ACM.
- [133] I. S. Reed and R. M. Stewart. Note on the existence of perfect maps. *IRE Transactions on Information Theory*, 8:10–12, 1962.
- [134] G. Reitmayr and T. W. Drummond. Going out: Robust tracking for outdoor augmented reality. In *Proc. of the 5th IEEE International Symposium on Mixed and Augmented Reality 2006*, pages 109–118, Santa Barbara, CA, USA, October 22–25 2006. IEEE and ACM, IEEE CS.
- [135] G. Reitmayr and T.W. Drummond. Going out: Robust model-based tracking for outdoor augmented reality. In *ISMAR 2006*, 2006.
- [136] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *Proceedings of the Third Asian Pacific Computer and Human Interaction, APCHI '98*, pages 63–, Washington, DC, USA, 1998. IEEE Computer Society.
- [137] C. Rhemann, C. Rother, A. Rav-Acha, and T. Sharp. High resolution matting via interactive trimap segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008*, 2008.
- [138] J. P. Rolland, L. Davis, and Y. Baillot. A survey of tracking technology for virtual environments. *Fundamentals of Wearable Computers and Augmented Reality*, 1st ed, W. Barfield and T.Caudell, Eds Mahwah, NJ: CRC, 2001.
- [139] Azriel Rosenfeld and John L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13:471–494, October 1966.
- [140] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *ICCV 2005*, 2005.
- [141] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Proceedings of the 9th European conference on Computer Vision - Volume Part I, ECCV'06*, pages 430–443, Berlin, Heidelberg, 2006. Springer-Verlag.
- [142] K. Satoh, S. Uchiyama, and H. Yamamoto. A head tracking method using bird's-eye view camera and gyroscope. In *Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 202 – 211, nov. 2004.
- [143] F. Schaffalitzky and A. Zisserman. Planar grouping for automatic detection of vanishing lines and points. *Image and Vision Computing*, 18:647–658, 2000.
- [144] Volker Scholz, Timo Stich, Michael Keckeisen, Markus Wacker, and Marcus Magnor. Garment motion capture using color-coded patterns. In *Computer Graphics Forum*, volume 24, pages 439–447. Wiley Online Library, 2005.

- [145] G. Schweighofer and A. Pinz. Robust pose estimation from a planar target. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2024–2030, Dec 2006.
- [146] M. Shin, B.-S. Kim, and J. Park. AR storyboard: An augmented reality based interactive storyboard authoring tool. In *ISMAR 2005*, 2005.
- [147] G. Simon. Tracking-by-synthesis using point features and pyramidal blurring. In *IEEE International Symposium on Mixed and Augmented Reality*, pages 85–92, 2011.
- [148] Mona Singh and Munindar P Singh. Augmented reality interfaces. *IEEE Internet Computing*, (6):66–70, 2013.
- [149] Alvy Ray Smith and James F. Blinn. Blue screen matting. In *SIGGRAPH '96*, 1996.
- [150] D. Stricker, G. Klinker, and D. Reiners. A fast and robust line-based optical tracker for augmented reality applications. In *Proceedings of the international workshop on Augmented reality : placing artificial objects in real scenes: placing artificial objects in real scenes*, IWAR '98, pages 129–145, Natick, MA, USA, 1999. A. K. Peters, Ltd.
- [151] Aidong Sun, Yan Sun, and Caixing Liu. The QR-code reorganization in illegible snapshots taken by mobile phones. In *International Conference on Computational Science and its Applications*, pages 532–538, aug. 2007.
- [152] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. In *SIGGRAPH 2004*, 2004.
- [153] I. E. Sutherland. The ultimate display. In *Proceedings of the Congress of the International Federation of Information Processing (IFIP)*, volume volume 2, pages 506–508, 1965.
- [154] C. Sweeney, J. Flynn, B. Nuernberger, M. Turk, and T. Hollerer. Efficient computation of absolute pose for gravity-aware augmented reality. In *Mixed and Augmented Reality (ISMAR), 2015 IEEE International Symposium on*, pages 19–24, Sept 2015.
- [155] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.
- [156] I. Szentandrasi, M. Dubska, M. Zacharias, and A. Herout. Poor Man's Virtual Camera: Real-time simultaneous matting and camera pose estimation. *IEEE Computer Graphics and Applications*, 2016.
- [157] I. Szentandrási, M. Zachariáš, J. Havel, A. Herout, M. Dubská, and R. Kajan. Uniform Marker Fields: Camera localization by orientable De Bruijn tori. In *11th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2012.
- [158] I. Szentandrási, A. Herout, and M. Dubská. Fast detection and recognition of QR codes in high-resolution images. In *Proceedings of 28th Spring conference on Computer Graphics*, Bratislava, SK, 2012. UNIBA.
- [159] Keisuke Tateno, Itaru Kitahara, and Yuichi Ohta. A nested marker for augmented reality. In *ACM SIGGRAPH 2006 Sketches*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.
- [160] G. Thomas. Mixed reality techniques for tv and their application for on-set and pre-visualization in film production. In *Int. Workshop on Mixed Reality Technology for Filmmaking*, 2006.



- [161] Shamik Tiwari, VP Shukla, SR Biradar, and AK Singh. A blind blur detection scheme using statistical features of phase congruency and gradient magnitude. *Advances in Electrical Engineering*, 2014, 2014.
- [162] Shamik Tiwari, VP Shukla, SR Biradar, and AK Singh. Blur parameters identification for simultaneous defocus and motion blur. *CSI transactions on ICT*, 2(1):11–22, 2014.
- [163] Roger Y. Tsai. Radiometry. chapter A Versatile Camera Calibration Technique for High-accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses, pages 221–244. Jones and Bartlett Publishers, Inc., USA, 1992.
- [164] Michael Tsang, George W. Fitzmzurice, Gordon Kurtenbach, Azam Khan, and Bill Buxton. Boom chameleon: Simultaneous capture of 3D viewpoint, voice and gesture annotations on a spatially-aware display. In *ACM SIGGRAPH*, 2003.
- [165] H. Uchiyama and E. Marchand. Deformable random dot markers. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR '11*, pages 237–238, Washington, DC, USA, 2011. IEEE Computer Society.
- [166] H. Uchiyama and H. Saito. Random dot markers. In *IEEE Virtual Reality Conf. (VR)*, 2011.
- [167] Hideaki Uchiyama and Eric Marchand. Toward augmenting everything: Detecting and tracking geometrical features on planar objects. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011.
- [168] L. Vacchetti, V. Lepetit, and P. Fua. Combining edge and texture information for real-time accurate 3D camera tracking. In *Proceedings of the 3rd IEEE International Symposium on Mixed and Augmented Reality*, 2004.
- [169] M.C. Vaz and C. Barron. *The Invisible Art: The Legends of Movie Matte Painting*. Chronicle Books, 2002.
- [170] J. Ventura, C. Arth, G. Reitmayr, and D. Schmalstieg. Global localization from monocular SLAM on a mobile phone. *Visualization and Computer Graphics, IEEE Transactions on*, 20(4):531–539, April 2014.
- [171] S. Vogt, A. Khamene, F. Sauer, and H. Niemann. Single camera tracking of marker clusters: Multiparameter cluster optimization and experimental verification. *2012 11th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 0:127, 2002.
- [172] D. Wagner and D. Schmalstieg. First steps towards handheld augmented reality. In *ISWC'2003*, 2003.
- [173] A. Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.
- [174] Hao Wang and Yanming Zou. Camera readable 2D bar codes design and decoding for mobile phones. *IEEE International Conference on Image Processing*, pages 469–472, 2006.
- [175] Thomas P. Weldon, William E. Higgins, and Dennis F. Dunn. Gabor filter design for multiple texture segmentation. *Optical Engineering*, 35:2852–2863, 1996.
- [176] Reg Willson. *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1994.

- [177] Grace Woo, Andy Lippman, and Ramesh Raskar. VRCodes: Unobtrusive and active visual codes for interaction by exploiting rolling shutter. In *Proceedings of ISMAR*, 2012.
- [178] S. Wright. *Digital compositing for film and video*. Focal Press, 2010.
- [179] Peng Yang, Itaru Kitahara, and Yuichi Ohta. [POSTER] remote mixed reality system supporting interactions with virtualized objects. In *Mixed and Augmented Reality (ISMAR), 2015 IEEE International Symposium on*, pages 64–67. IEEE, 2015.
- [180] X. Yang and K.-T. Cheng. Ldb: An ultra-fast feature for scalable augmented reality on mobile devices. In *12th IEEE International Symposium on Mixed and Augmented Reality*, pages 49–57. IEEE Computer Society, 2012.
- [181] Ka-Ping Yee. Peephole displays: Pen interaction on spatially aware handheld computers. In *CHI*, 2003.
- [182] X. Yu, H. Ch. Lai, S. X. F. Liu, and H. W. Leong. A gridding Hough transform for detecting the straight lines in sports video. In *Proc. IEEE International Conference of Multimedia and Expo, ICME*, 2005.
- [183] Jean yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.
- [184] M. Zachariáš, I. Szentandrás, and A. Herout. Visual correction of position drift using uniform marker fields. In *Spring conference on Computer Graphics*, Bratislava, SK, 2016. UNIBA.
- [185] X. Zhang, S. Fronz, and N. Navab. Visual marker detection and decoding in ar systems: a comparative study. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, pages 97 – 106, 2002.
- [186] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 666 –673 vol.1, 1999.
- [187] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, November 2000.
- [188] Zhengyou Zhang. Camera calibration with one-dimensional objects. In *Proceedings of the 7th European Conference on Computer Vision-Part IV, ECCV '02*, pages 161–174, London, UK, UK, 2002. Springer-Verlag.
- [189] F. Zhou, H.B.-L. Duh, and M. Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 193 –202, sept. 2008.
- [190] Zhiwei Zhu, V. Branzoi, M. Wolverson, G. Murray, N. Vitovitch, L. Yarnall, G. Acharya, S. Samarasekera, and R. Kumar. Ar-mentor: Augmented reality based mentoring system. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 17–22, Sept 2014.

# List of Publications

For my masters thesis, I investigated several methods for fast approximation of global illumination using GPU shaders. A subset of these experiments, concerning mainly Screen-Space Directional Occlusion was published in:

- [I] I. Szentandrás (supervisor A. Herout). **Modern methods of realistic lighting in real time.** In *Proceedings of The 15th Central European Seminar on Computer Graphics*, pages 17–24. Technical University Wien, 2011.

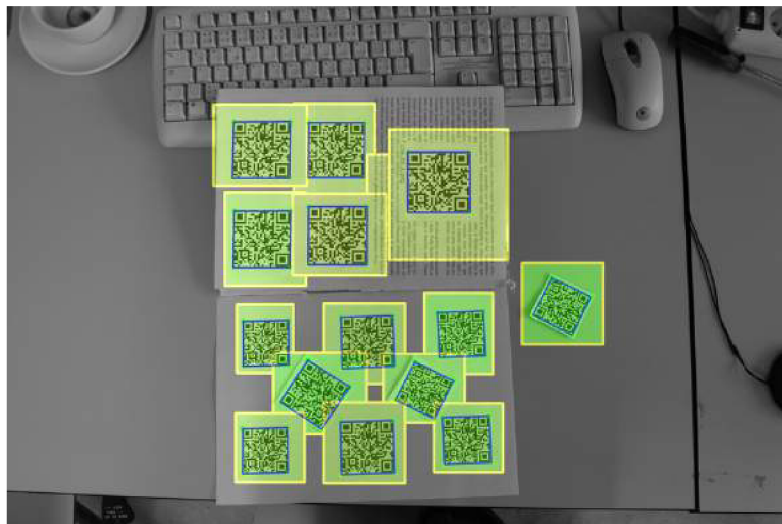


Figure 1: Fast detection and recognition of QR codes in high-resolution images [II].

Dubská et al. [36] introduced a QR code detection algorithm using Hough transformation (PC-lines). In a follow up work (Figure 1), I proposed a method for fast localization of checkerboard patterns as candidate positions for QR code detection:

- [II] I. Szentandrás, A. Herout, and M. Dubská. **Fast detection and recognition of QR codes in high-resolution images.** In *Proceedings of 28th Spring conference on Computer Graphics*, Bratislava, SK, 2012. UNIBA.

The main contributions of this thesis have been published in the following four publications. Uniform Marker Fields (UMF) are fiduciary markers with checkerboard pattern, where every  $n^2$  window is unique in every rotation (Figure 2). They were first published in:

- [III] I. Szentandrás, M. Zachariáš, J. Havel, A. Herout, M. Dubská, and R. Kajan. **Uniform Marker Fields: Camera localization by orientable De Bruijn tori.** In *11th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2012.

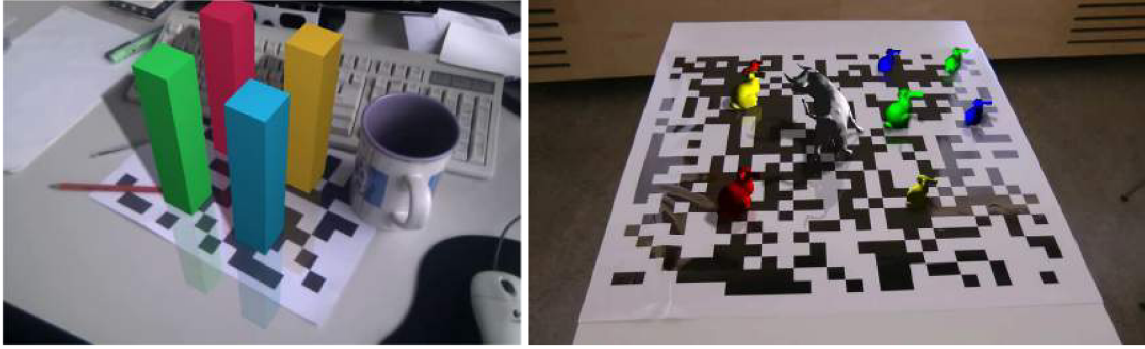


Figure 2: Binary Uniform Marker Fields with 3D augmentations [III].

My contribution involved mainly the detection algorithm of UMF and evaluations. The algorithm is highly memory efficient with low computational complexity. In collaboration with Kajan et al. we proposed to use UMF to aid inter-device communication by establishing relative pose between two devices. The UMF detection algorithm proved robust against high transparency and image clutter.

[IV] R. Kajan, A. Herout, I. Szentandrás, and M. Zachariáš. **On-screen marker fields for reliable screen-to-screen task migration.** In *SouthCHI 2013: International Conference on Human Factors in Computing and Informatics*, pages 692–710, 2013.

A major improvement and generalized concept of Uniform Marker Fields was published in:

[V] A. Herout, I. Szentandrás, M. Zachariáš, M. Dubská, and R. Kajan. **Five shades of grey for fast and reliable camera pose estimation.** In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1384–1390. IEEE Computer Society, 2013.

By using the edge gradients between UMF modules allowed for larger, less obtrusive marker fields compared to binary UMF. Thanks to the changes, I was able to make the detection algorithm more robust against occlusion, lighting changes and motion blur (Figure 3). I also further optimized the detection algorithm to work on mid-range smartphones in real time.

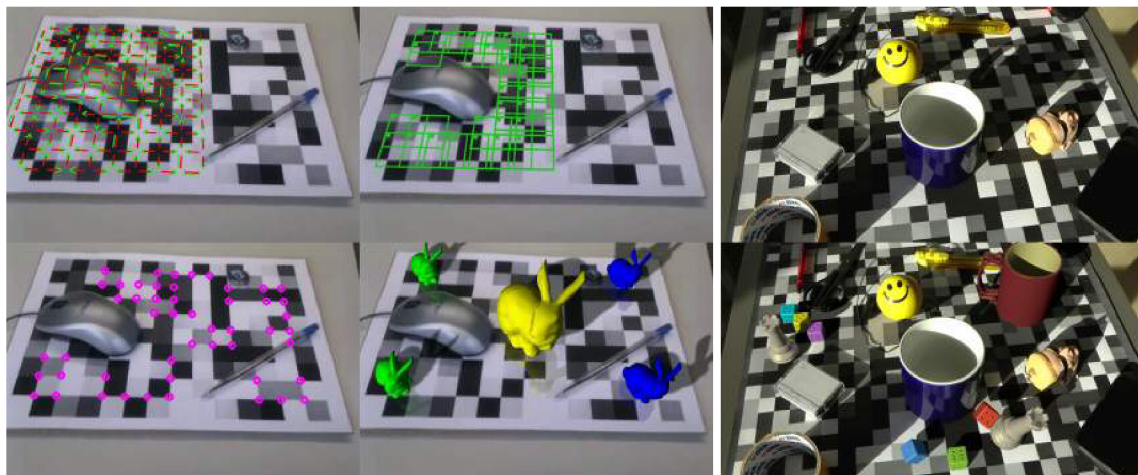


Figure 3: Grayscale Uniform Marker Fields detection algorithm and example of augmentations [V].

Thanks to the positive properties of Uniform Marker Fields design and robust detection algorithm, we also experimented with extremely low-contrast markers. We first proposed using RGB markers as part of a greenscreen in:

- [VI] M. Dubska, I. Szentandrasi, M. Zacharias, and A. Herout. **Poor man’s SimulCam: Real-time and effortless matchmoving.** In *12th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 249–250, 2013.

I was involved in the effort to create easily detectable deformable Marker Fields with a hexagonal structure. The published solution was able to reconstruct the deformed marker in  $5\text{ ms}$  for VGA resolution.

- [VII] Zs. Horváth, A. Herout, I. Szentandrasi, and M. Zachariáš. **Design and detection of local geometric features for deformable marker fields.** In *Proceedings of 29th Spring conference on Computer Graphics*, pages 85–92. Comenius University in Bratislava, 2013.

We developed further our task-migration solution based on a video recording metaphor [IV]. The resulting fully Augmented Reality solution was published in:

- [VIII] R. Kajan, I. Szentandrasi, A. Herout, and A. Pavelková. **Reliable and unobtrusive inter-device collaboration by continuous interaction.** In *Journal of WSCG*, pages 95–103. University of West Bohemia in Pilsen, 2014.



Figure 4: Augmented Reality for video streams from static cameras [IX] using [37].

- [IX] I. Szentandrasi, M. Zachariáš, R. Kajan, J. Tinka, M. Dubska, J. Sochor, and A. Herout. **INCAST: Interactive camera streams for surveillance cams AR.** In *Proceedings of the 2015 14th IEEE International Symposium on Mixed and Augmented Reality*, pages 1–5. Institute of Electrical and Electronics Engineers, 2015.

In this work, we dealt with the challenging task of camera pose estimation using static cameras and its usability for augmented reality applications. We presented a prototype system for broadcasting an augmented video stream, and demonstrated the possibilities on several use-cases – non-interactive demos and simple AR games (see Figure 4).

In collaboration with M. Zachariáš and A. Herout, we have investigated the viability of using unobtrusive colored UMF markers for indoor navigation. We tested the precision of the UMF detection algorithm relative to real-world ground truth distances. This work has been accepted to SCCG 2016:

- [X] M. Zachariáš, I. Szentandrási, and A. Herout. **Visual correction of position drift using uniform marker fields.** In *Spring conference on Computer Graphics*, Bratislava, SK, 2016. UNIBA.

We developed further the concept of using low-contrast markers as part of a greenscreen in [VI]. I proposed a new mapping of Uniform Marker Fields into  $YC_bC_r$  channels for more robustness, improved detection algorithm for camera pose estimation, presented a distribution of computations between different computational units for real-time performance, and described an efficient matting method using GPU shaders. We created a plugin into the popular Unity 3D engine. To evaluate our solution, we implemented a real-time preview application (see Figure 5). This work has been accepted and pending publication in IEEE Computer Graphics and Applications journal.

- [XI] I. Szentandrási, M. Dubská, M. Zachariáš, and A. Herout. **Poor Man's Virtual Camera: Real-time simultaneous matting and camera pose estimation.** *IEEE Computer Graphics and Applications*, 2016.

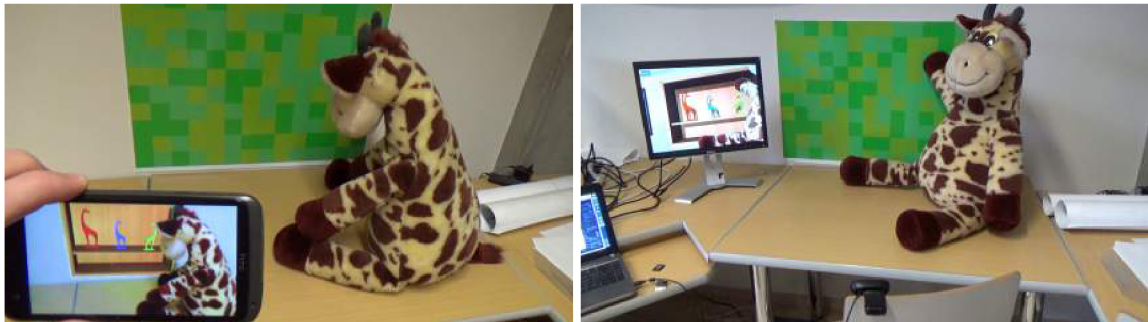


Figure 5: Camera pose estimation and chromakeying using greenscreen markers on mobile and PC platforms [XI].