Evaluation of Printed Augmented Reality Markers

Thomas M. Irrer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im November 2019

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, November 24, 2019

Thomas M. Irrer

Contents

Abstract

The increase in the computational capacities of phones, in addition to the increased quality of the cameras installed in them, gave rise to untold possibilities for augmented reality applications. With the wider use of AR systems, the range of situations marker systems are used in also increased. Naturally, this leads to marker systems being used in suboptimal conditions, causing failure in the system. The reasons for this failure, however, may not always be obvious. The initial goal of this project was the creation of a tool that allowed the developer of an AR application to quickly evaluate the quality of a locale or display mode for their AR markers, as well as provide feedback on what may be responsible for the failure. The intention was to test many images to find a correlation between obvious problems of the images, such as blur or poor lighting, in order to determine which problems influence the detection process. In this paper the attempt at finding a method by which to judge the quality of images for augmented reality applications is detailed. It includes a brief overview of the topic of AR and a deeper dive into the sub-genre of visual marker-based AR. The path to finding said method required the analysis of images containing AR markers, specifically QR codes. As such the way by which QR codes are detected and decoded is explained in detail, including most reasons why the detection might fail. Additionally, the program by which the analysis took place is discussed in its implementation, from the software architecture, the libraries used to the ways the implemented process differs from the standard QR code detection as well as from other detection methods. The results of testing hundreds of images on image conditions and the ability to find a QR code in them make up one chapter, which also includes an explanation of the results and their meaning for Code detection. The results of this testing did reveal some of the issues and their influence on the detection, however, no satisfactory method of judging or grading could be found to accurately represent the issues present in the image.

Kurzfassung

Fortschritte in der Rechenleistung und der Qualität der verbauten Hardware in Mobiltelefonen erlaubt die Entwicklung immer größerer und ambitionierteren Augmented Reality (AR) Anwendungen. Durch die weiter verbreitete Nutzung, stieg allerdings auch die Verwendung von AR-Systemen in ungeeigneten Situationen, was zum Scheitern der Projekte führen kann. Die Gründe für dieses Scheitern sind aber nicht immer offensichtlich. Zu Beginn war das Ziel dieses Projektes die Entwicklung von analytischer Software um Entwickler von AR-Anwendungen mit visuellen Marker-Systemen mehr Daten über die Bedingungen unter welchen die Marker-Systeme verwendet werden, zu liefern. Spezielle Probleme, die es zu erkennen und quantifizieren galt waren die Lichtbedingung und Ausrichtung der visuellen Marker. Das Ziel war Gründe für das Scheitern zu benennen, zu quantifizieren und Feedback über mögliche andere Probleme zu liefern. Diese Arbeit beschreibt die Methodik, mit welcher ein Versuch getätigt wurde, ein System zu entwickeln, welches Bilder für ihre Nutzbarkeit für AR Anwendungen bewertet. Die Arbeit beinhaltet eine kurze Zusammenfassung der historischen Entwicklung von AR Systemen und eine tiefer gehende Erklärung des Subgenres von AR-Systemen basierend auf visuellen Markern, speziell QR-Codes. QR-Codes sind die primären Test Marker für dieses Projekt und als solches wird die Funktionalität, welche die Erkennung und Decodierung ausmacht, genau unter die Lupe genommen. Von speziellem Interesse sind die Bedingungen unter welchen die Erkennung von QR-Codes scheitert, um daraus Schlüsse über die allgemeine Erkennung von AR-Markern zu ziehen. Um eine Korrelation zwischen bestimmten Aspekten von Bildern, wie zum Beispiel die Lichtsituation, und dem Scheitern von der Erkennung von QR-Codes zu finden, wurde Software implementiert um eine große Menge an Bildern testen zu können. Das Programm entwickelt für diesen Zweck wird in seiner Architektur und Funktion erklärt, die Libraries verwendet, in ihrer Rolle für dieses Projekt angesprochen. Die Resultate des Evaluierens von über 100 Bildern, welche QR-Codes in verschiedensten Situationen zeigen, macht ebenso ein Kapitel aus. Das Ziel war es hier nicht zu messen wann die Erkennung von QR-Codes scheitern, sondern zu hinterfragen was sich aus dem Scheitern einer speziellen Methode über die im Bild gegebenen Bedingungen schliessen lässt.

Chapter 1

Introduction

1.1 Context

With the rise in computing power in mobile devices as well as the increase in mobile internet connection speeds, an abundance of Augmented Reality (AR) applications have been developed. Given the breadth of real-world scenarios, AR is applied in suboptimal use conditions are inevitable. The following paper is interested in the particular conditions that affect the quality of use for AR systems using visual AR markers. A wide array of images containing visual markers were analyzed through different means of computer vision and general image analysis combined with a marker detection and analysis tool in order to attempt to establish a connection between data points of image quality and failure to detect the code correctly. The operative question arising through this process is: Which performance-relevant criteria are essential for evaluating the quality of a printed or displayed augmented reality marker?

1.2 What is Augmented Reality?

The classification of Virtual Reality(VR) and AR is a problem the research community has been struggling with for a while. VR is generally associated with the science fiction idea of a *Star Trek* type Holodeck, allowing a user to fully immerse themselves in a synthetic, computer-generated world, which emulates emulated all stimuli associated with the experience in that world. The current technological possibilities are far removed from the science fiction, as the virtual worlds only include visual and auditory feedback and are generally rather limited in the interaction possibilities. Finding the connection between AR and VR proves difficult as, either has been described as special cases of the other; AR being a special case of VR, or AR being a general concept and Virtual reality being a specific field under it. For the sake of this paper the connection will be drawn as described by Bimber and Raskar in the introductions to their book on the topic of Spatial Augmented Reality: “Rather than immersing a person into a completely synthetic world, AR attempts to embed synthetic supplements into the real environment(...)” [3, p. 2]. The integration of synthetic information into the real environment knows barely any bounds and finding the edges of what can reasonably be defined as AR seems close to impossible. One aspect which is required for an AR application is a user in order to create a spatial connection between the real environment and the augmentations. This allows the distinction between the mimicking of hologram technologies by projecting images onto glasses and the displaying of the same images on a monitor, the first being an example of AR and the second arguably not. Similar to the above mentioned Holodeck, the synthetic information generated does not need to be strictly visual, as both auditory and sensory applications are not only imaginable but were implemented to a degree. Through the use of a phones geo-location feature linked with a specifically designed app, an art installation was created at a Holocaust memorial in Berlin. The installation allows visitors to experience a concert that was performed in 2008 at the memorial for which the musicians were positioned throughout the area. As the orchestra was spread all over the area of the memorial, the app uses the GPS position of the phone to recreate the music which would have been heard at any point in the memorial. This project fulfills the requirement explained in the book Spatial Augmented Reality of creating a spatial relation ("registration") between the real environment and the augmentations [3, p. 2]. An example of how touch and movement can be synthetically recreated, hydraulically mounted driving simulators fulfill the requirements somewhat. By tilting the chair gravity on the user can emulate the forces experienced, while driving. Added screens and surround sound can further increase the immersion into the system. By combining the hydraulic seat with a head-mounted display, the setup could easily be classified as Virtual Reality. This shows that the transition between AR and VR is no strict separation, but more of a gradient of how much of the real environment flows into the experience. The connecting tissue for all AR is the extension of the human experience of the real world by a technological component. For the sake of this thesis, these applications are of no relevance, as the artistic value of AR or its influence on user experience is of no interest. The primary form of AR used for this project is visual AR systems. One prominent example of such systems are the Google Glasses, head-mounted computers, which use a camera and a small display create a head-up display. Using image recognition the device was able to project objects into a user's field of view i.e., a keyboard projected onto the user's hand in order to accept calls. Somewhat related to AR glasses, projecting 3D objects onto surfaces is a common application of AR systems. By viewing an AR marker through an application designed for this purpose using the device's camera, 3D objects can be projected on the marker. The use of markers provides the advantage of being able to select the 3D model based on the marker, allowing for multiple different objects to be projectable, as well as providing better data to the system allowing for a more stable projection. The use of visual markers for storing information causes a return to the question of whether the system can be classified as AR. An Example of QR codes, which are the primarily tested system of this work, can be used to link to websites, connect to WIFI networks, used for cashless payment and much more. Using a code to link to an internet site definitely qualifies as augmenting reality, however, the interaction possibilities are limited to make the classification of AR arguable at best.

1.3 Goal

As eluded to previously, visual AR marker systems were the primary driver of this thesis and project. The initial intent of this work was the creation of a tool allowing developers to gain deeper insight into why their designed AR markers failed in a specific situation. The goal was the creation of a rating system as well as providing feedback on what may be hindering the detection process. This turned out to be virtually impossible to achieve with classical methods, due to multiple complications. The nature of AR markers allows issues in early steps of the detection process to permeate only causing failure later. Additionally finding a correlation between calculated or measured image statistics to whether or not AR markers can be found in the image proved to difficult. As a consequence, attempts at creating the mentioned rating system, all suffered from high levels of arbitrary selection of values. A combination of these factors led to a change in approach to only gathering data, which could be used in future works to feed a neural network in order to use the self-adjusting nature of such systems to create a system that could evaluate images in a way impossible to achieve for human capabilities.

Chapter 2

Visual Augmented Reality Markers

2.1 Introduction

This chapter serves to provide an introduction into the field of visual augmented reality Markers, in both their applications as well as their functionality. This includes a short overview of the general development of augmented reality as a whole, as well as a more detailed explanation of the application and functionality of visual marker systems in general, and QR codes specifically.

2.2 History

The field of Augmented reality is not limited to a specific device or medium and can even vary in the sensory form it is perceived as. As such the definition of AR is fraught with inconsistencies, as the introduction already eluded to. In order to potentially give a better idea of what AR entails the next section will briefly address the development of augmented reality. Augmented reality systems use a wide array of different marker technologies from GPS markers, as shown with incredible virality in *Pokemon GO*, RFID Tags used in automatic museum guides or Sound markers better known as speech activation. Historically the field of augmented reality was a topic of research since 1968 when Ivan Sutherland created the first augmented reality system [12] which used two tiny CRT displays and semi-reflective mirrors to allow a user to see both the image of the monitors and their surroundings. In conjunction with a position sensor, it enabled the projection of simple wire-frame objects into the field of view of a user. The technical limitations of the time restricted the usage to an area of about two square meters and a head tilt of a maximum of 40 degrees. The term Augmented Reality referring to the superimposing of computer-generated data onto the view of the real world was coined in 1992 by Tom Caudell and David Mizell, who also discussed the required registration for the alignment of the real and virtual world. The improvements of mobile devices in conclusion with the 1993 introduced GPS System allowed for new developments in AR. 1996 saw the introduction of 2D matrix markers allowing for camera tracking with six degrees of freedom by Jun Rekimoto, a development that culminated in 1999 with the presentation of the open-source ARToolKit pose-tracking library, which is still in use to this day. The general barcode marker systems used for data storage, existed long before that, as the first barcodes were developed back in 1948 in the form of one-dimensional data storage systems for commercial payment tracking in checkout lines, which would go on to reduce human error as well as repetitive strain injuries. The improvement to 2D barcode occurred in 1987 with the development of the PDF417 codes, which stored data in patterns of 4 bars and spaces, overall 17 modules long. Seven years later in 1994, the *Toyota* subsidiary *Denso Wave* was tasked with improving the barcodes used in the automotive factories, with more speed and error-free decoding. The decision to publicize the specifications allowed for the widespread adaption of the system, which can be seen today. The increased availability of mobile phones with built-in cameras lead to advancements in the tracking of real camera images and the addition of virtual objects. Improvements in computational power in mobile devices also saw the return of wearable devices with more precise tracking sensors to ensure minimal drift in the head-mounted displays. Increases in precision in GPS tracking and computational power allowed for more applications to run on mobile devices with lower effort, leading to the development of more consumer applications such as games or PDA applications. 2005 saw the introduction of three-axis accelerometers into mobile phones, allowing for the development of better orientation detection systems. From that point to the current moment the biggest developments were a product of the incredible increase of computation power as well as mobile internet connectivity, allowing for borderline limitless AR applications. An extensive list of developments and examples, including the ones mentioned here, was found in a technical report Institute for Computer Graphics and Vision at the Graz University of Technology [1].

2.3 Visual Marker Systems

Visual markers are the primary target technology of this work. As the name suggests require a camera to record either specific designs which are either stored in a marker library of the corresponding software or uses computer vision techniques to detect and project onto a surface in the image. Storing information in square barcode was possible since the 1950s. The tracking and superimposing of images and 3D models onto these codes, however, is a rather new development, which arrived in 1999 available to the public in the form of ARToolkit. As this form of marker tracking was one of the failing processes which inspired this project, the functionality of the system is the topic of the next paragraph. The initial goal of ARToolkit was the creation of an AR supported conferencing system. This includes virtual monitors, which project a video stream of a communication partner onto the markers for the user wearing the head-mounted display (HMD) and a tabletop surface bounded by six markers, within which pens can be tracked to write or draw on in the virtual world. The process of locating the size-known markers, which represent the base of the virtual monitors, begins by thresholding the input image and finding contour regions which can be fitted by four line segments, which are then extracted [9, p. 4]. Through perspective transformation, the position of the markers in the camera space is determined, which can then be used to calculate the transformation parameters needed to inject the video-stream back into the image. The ARToolkit project itself, seems basic for modern standards, but the library, created from the project, allowing for pose-tracking with six degrees of freedom, using square fiducials and an approach to the recognition of markers, released open-source under the GPD license, would go on to allow a wide range of people to work with AR. The system is still in use to this day with One of the more common applications of this technology being the projection of 3D models onto the code or the surface when viewed through the camera lens. The process by which QR codes are located in an image share some similarities with the process of ARToolkit, which will be the topic of the following chapters.

2.4 Why QR-Codes?

To test the impact of certain image features on marker detection quality, a particular visual marker is necessary to attempt some marker detection. For this project, Quick Response codes (QR Codes) were chosen as a visual marker. The reasons for this choice were the accessibility of the system, given its open-source nature, making it easier to analyze the detection process. Additionally, the QR system is particularly robust, with multiple layers of error correction and redundancy, allowing for decoding even in disadvantageous situations. Particularly the error correction capabilities allow for additional analysis as the difference between the code which was detected in the image might differ from the actual marker, the disparity between which can be measured, providing more data. The specific structure and subsequent detection process of QR Codes also allow for gradual failure, as only parts of the code might get found, which further enables the recognition of problems with the image, a different marker system might not allow for. Even the downside of using a highly specialized marker such as QR codes are limited, as the steps necessary for successful detection are shared with many AR Systems, allowing for results to be more generically applicable. The specifics of how and why QR code detection fails is addressed in a later chapter.

2.5 Versions

There are five different variants of QR codes on the market currently. The most common and most likely to be referred to as QR codes are the model 1 and model 2 codes, model 2 being an improvement on the original model 1 byte increasing the maximum version from 14 (73×73 modules) to version 40 (177×177 modules), a module hereafter referring to one black or white segment of the QR code. QR codes are primarily used to store text or number sequences, requiring the version of the code to be selected based on the amount of data supposed to be stored. Additionally, the amount of data that can be encoded in a QR code depends on the error correction level of the code, which represents a trade-off between the amount of storable data and the amount of data which could be restored in case parts of the code are not correctly read. Reasons for an incomplete or incorrect reading of the code were discussed in the previous section 2.6. As an example a version 1 code, consisting of 21×21 modules can store 152 bits of data at Error correction level L, the lowest level, but only 82 bits at the highest level H. In order to store the same 152 bits of data at the highest level of error correction a version 3-H would be required, which allows for up to 208 bits of data to be encoded. Version 40, the highest version of model 2 codes, built up of 177×177 modules enables up to 23648 bits of data to be encoded. In order to store numbers or letters in the code, an encoding structure is required. This is however not the limit on the amount of data that can be stored, as one of the modes describing the data structure encoded in the bit data, the *Structured Append* mode allows for the encodations of the data from a particular message to be split over multiple QR Codes, by encoding the length of the data sequence and one codes position in it, in each QR code. Other encoding modes describe the default data structure of the bit-stream. The first of which being the numeric mode. It enables up to 7089 numeric characters from the decimal digit set 0 to 9, by storing three digits per 10 bits, to be encoded in a version40-L code. The alphanumeric mode enables the encoding of 45 separate characters, i.e. 10 numeric digits (0-9), 26 alphabetic characters (A-Z), with no capitalization as well as 9 symbols (space, \$, %, $*$, $+$, $-$, ., /, :). To achieve this, two characters are represented by 11 bits, reducing the number of characters storable in a version 40 code to 4296. The most common encoding for the commercial use of QR codes is an 8-bit character set. Given that QR codes are an invention of a Japanese company the initial encoding system for 8 bit data was the JIS X 0201 standard which enables the storage of the same 10 numeric digits (0-9), $26 \cdot 2$ alphabetical characters, for lower case and capitalized letters, 32 symbols, as well as 96 phonetic Japanese katakana signs. Modern implementations allow for single-byte encoding in the ISO8859-1 standard, also known as Latin 1, UTF-8 and Shift_JIS, the first two used for western languages and the latter offering encoding for the Japanese language. Using 8 bit to encode one character lowers the number of storable characters to 2953 or roughly 1/8 of the bit capacity. Given its Japanese roots, a standard for encoding Kanji symbols exists which requires a 13-bit binary codeword, per symbol. The FNC1 mode is used for messages encoded according to the UCC/EAN Application Identifiers standard as well as in accordance with industry standards according to AIM International. Lastly, the Extended Channel Interpretation (ECI) mode is used to include data stream interpretations different from the default, allowing international character sets, general-purpose interpretations for e.g, encryption, user-defined interpretations for closed industry applications or control information for structured append. Through this and the binary form of the data, any feasible way of encoding data could be theoretically be encoded in a QR code. This allows QR codes to find uses from industrial labeling to cashless payment methods.

2.5.1 Structure

The exact structure of QR symbols as described in the ISO/IEC 18004:2000(E) [15, p. 6] standard consists of function patterns which include, position detection patterns, timing patterns and alignment patterns, as well as an encoding region which contains format and version information as well as the data and error correction codewords. The entire symbol needs to be surrounded by a quiet zone as per standard. An illustration of this structure can be seen in figure 2.1. The finder pattern consists of three identical position detection patterns at the upper left, upper right and lower left corners of the symbol respectively. One position detection pattern is 7×7 modules of size and containing concentric squares of alternating black and white color, as can be seen in figure 2.2. The advantage of this specific structure is its rotational invariance, visualized in figure 2.3, as regardless of the direction the pattern is parsed from the ratio is 1:1:3:1:1 of black:white:black:white:black module. The identification of all three position detection patterns unambiguously defines the location and orientation of the symbol, ignoring the

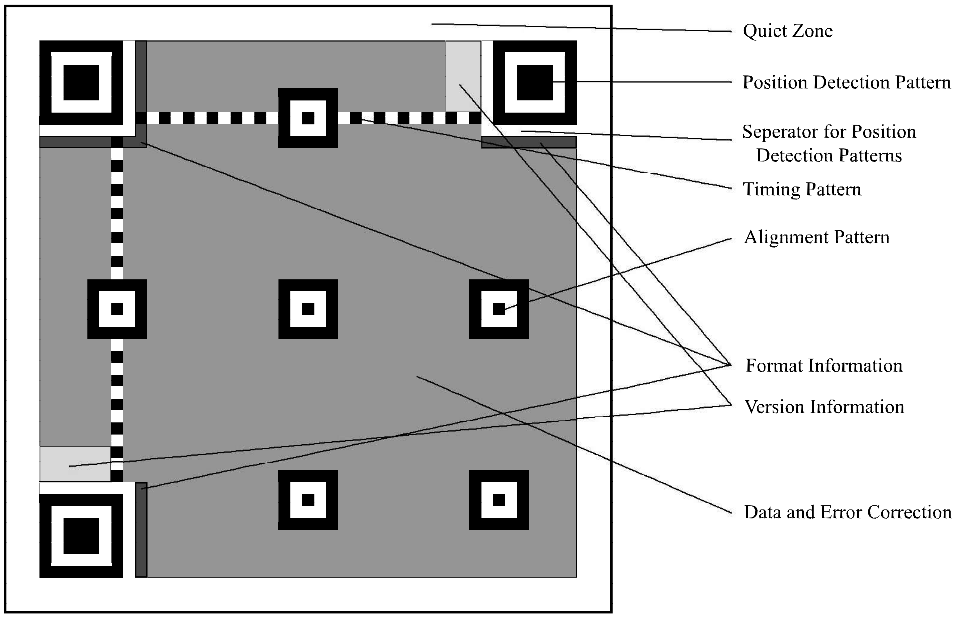

Quiet Zone
Position Detection Pattern
Seperator for Position Detection Patterns
Timing Pattern
Alignment Pattern
Format Information
Version Information
Data and Error Correction

Figure 2.1: Structure of QR codes recreated from [15, p. 13].

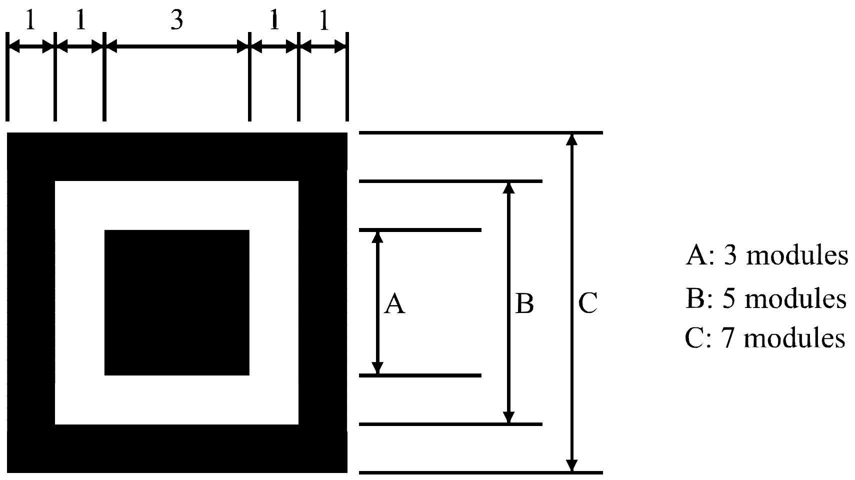


Figure 2.2: Structure of a position detection pattern recreated from [15, p. 13].

possibility of image being mirrored or flipped. Omitted from the previous listing of the components of a QR code, the position detection patterns are surrounded by a one module wide separator, constructed of all light modules. Row and column six contain the timing pattern, alternating light, and dark one module wide pattern, connecting the position detection patterns. The regular nature of the pattern allows for the establishing of estimated module size and module density. "Each Alignment Pattern may be viewed

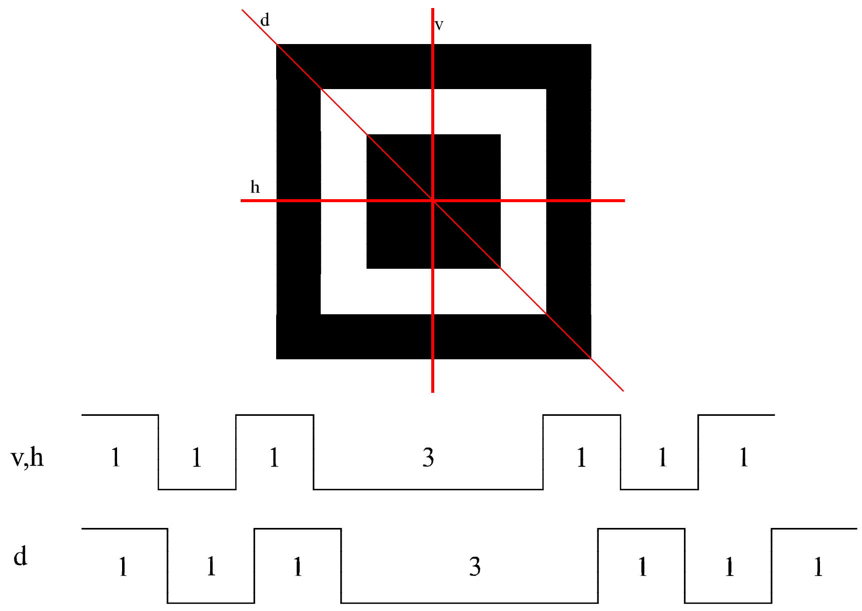


Figure 2.3: Visualization of the rotational invariance of position detection patterns recreated from [2, p. 3].

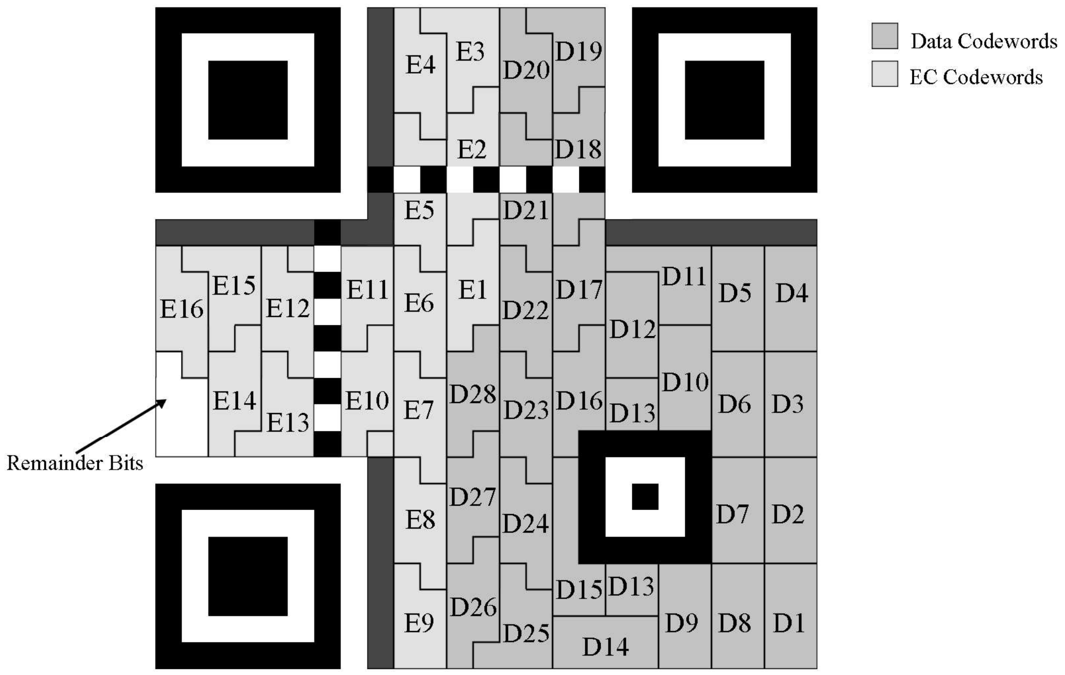


Figure 2.4: Data and error correction codeword structure in Version 2-M QR code recreated from [15, p. 48].

Error Correction Level	Binary Indicator
L	01
M	00
Q	11
H	10

Table 2.1: Binary indicator of error correction level in format information.

as three superimposed concentric squares and is constructed of dark 5x5 modules, light 3x3 modules and a single central dark module [15, p. 13].” The number of alignment pattern is dependant on the version of the code and only exist in codes of model 2 Version 2 or larger. They are position symmetrically on either side of the diagonal from the top left and bottom right corner and spaced as evenly as possible. The primary purpose of the alignment patterns is the compensation of image distortion. Lastly, a crucial component of QR codes is the mandatory quiet zone around the symbol of at least 4 modules width, free of any markings and of similar brightness as the light modules of the code. The remaining descriptors from figure 2.1 are related to the encoded information. Figure 2.4 shows an example of how data codewords, as well as error correction codewords, are stored in a code of version 2 with error correction level M. There are four different levels of error correction Low (L), Medium (M), Quartile (Q) and High (H). They are capable of restoring approximately 7, 15, 25 and 30% of the code for the respective level. QR codes generate a series of error correction codewords which are added to the data codewords, meaning the higher the correction level the lower the amount of actual data can be stored in a code of the same version. Two types of mistakes can be corrected in the data: erasures and errors. erasures are erroneous codewords at known locations and errors at unknown locations. Erasure would be an unscanned or undecodable symbol character. An error is a misdecoded character. The number of erasures e and errors t correctable by the system can be calculated using the following formula 2.1, where d is the number of error correction codewords and p is the number of misdecode protection codewords, followed by an example calculation taken from the QR code ISO standardization document[15, p. 33]:

$$e + 2t \leq d - p. \tag{2.1}$$

“For example, in a version 6-H symbol there is a total of 172 codewords, of which 112 are error correction codewords (leaving 60 data codewords). The 112 error correction codewords can correct 56 misdecodes or substitution errors, i.e. 56/172 or 32.6% of the symbol capacity.”

Information on the error correction level of a code is stored in the format information area adjacent to the position detection patterns. It contains a 15-bit sequence of 5 data bits and 10 error correction bits. The first two bits contain the error correction level. The binary indicators and their respective error correction level can be seen in table 2.1. The remaining three bits contain information on the masking pattern used for the code. Code masking is done to ensure no structural pattern, particular the 1011101-bit order found in the position detection patterns, appear by accident in the data area of the

Mask binary indicator	Condition
000	$(i+j) \bmod 2 = 0$
001	$i \bmod 2 = 0$
010	$j \bmod 3 = 0$
011	$(i+j) \bmod 3 = 0$
100	$(\frac{i}{2}+\frac{j}{2}) \bmod 2 = 0$
101	$(i \cdot j) \bmod 2 + (i \cdot j) \bmod 3 = 0$
110	$((i \cdot j) \bmod 2 + (i \cdot j) \bmod 3) \bmod 2 = 0$
111	$((i \cdot j) \bmod 3 + (i \cdot j) \bmod 2) \bmod 2 = 0$

Table 2.2: Masking condition for each binary indicator.

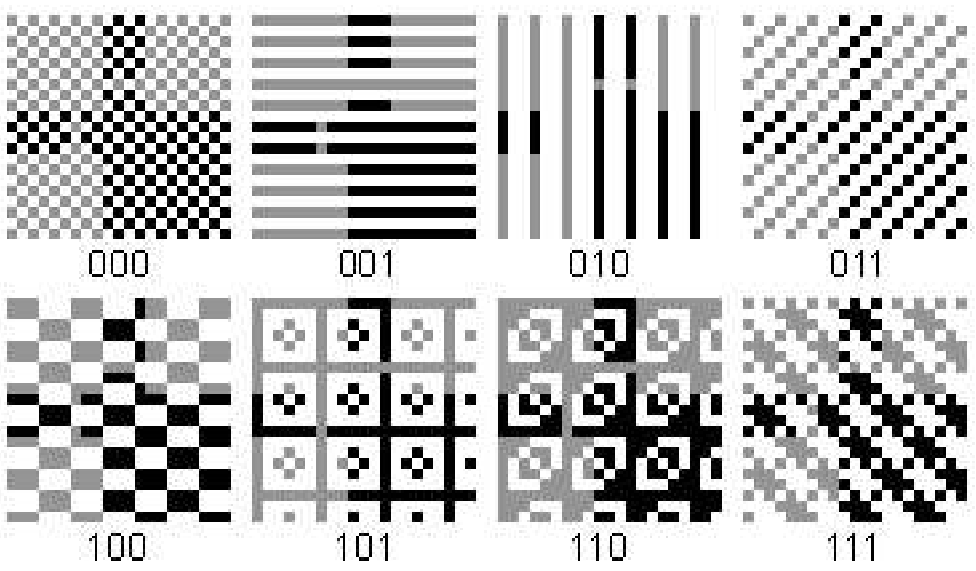


Figure 2.5: QR code mask patterns for their respective binary indicator.

code, as well as attempting to ensure a balance between light and dark modules. The masking patterns are generated as functions of the conditions in table 2.2. The resulting binary matrices, as seen in figure 2.5 are applied to the codes binary module pattern, excluding the function patterns, as well as the format and information areas, reversing each module which corresponds to a dark module in the masking pattern. The resulting matrix is evaluated through penalties for undesirable features. The penalties calculated for the masking results consider, adjacent modules of the same color within the same row or column, blocks of the same color, the sequence dark, light, three dark, light, dark, appearing in the result as well as the proportion of dark modules in the entire code. This process is repeated for each masking pattern and the result with the lowest penalty is selected. The selected mask is stored in the last three bits of the format information to enable unmasking, the reversal of the masking operation, in order to read the code. The version information encoded adjacent to the bottom left and top right position detection pattern contains an 18-bit sequence of 6 data bits and 12 error

	Data Codewords					Error Correction Codewords			
Block 1	D1	D2	...	D11		E1	E2	...	E22
Block 2	D11	D13	...	D22		E23	E24	...	E44
Block 3	D23	D24	...	D33	D34	E45	E46	...	E66
Block 4	D35	D36	...	D45	D46	E67	E68	...	E88

Table 2.3: Data structure example for Version 5-H QR Code.

correction bits. Only versions 7 to 40 contain the Version information, making an all-zero data string impossible. It is encoded twice in the code since its correct decoding is essential to the decoding of the complete symbol. Although the version of the code can be estimated, if more than one position detection pattern could be found in the image, a mismatch between the calculated and the encoded version could give insight into display problems, however. The encodation process requires the input data to be converted into a bitstream, segmented into a mode indicator, a character count indicator as well as the data bitstream itself. In addition, an Extended Channel Interpretation mode allows for en- and decoding of alternative interpretations of the byte values, e.g. Greek letters. The default encoding modes include numeric, alphanumeric, JIS8 byte encoding, as well as Kanji characters. Additional mode indicators exist for the end of the message in the form of 0000, as well as structured append indicators, which are used for data that is stored across multiple QR codes. Lastly, the FNC1 mode is used for data encoded according to specific industry standards agreed upon with AIM International or the UCC/EAN Application Identifiers standard. To construct the final codeword sequence to be stored in the symbol data codeword sequence is divided into n blocks according to the version and error correction level. The final sequence is constructed by taking data from each block in turn: data block 1, codeword 1; data block 2, codeword 1, ..., data block $n-1$, final codeword, data block n, final codeword, and similarly for the error correction blocks. In order to exactly fill the number of modules in the encoding region, 3, 4 or 7 remainder bits may be required dependant on the version. The ISO standard provides an example of how this process is applied to a code of version 5 2.3, which is also the table referred to in the quote [15, p. 46]:

> “For example, the Version 5-H symbol comprises four data and four error correction blocks, the first two of each of which contain 11 data and 22 error correction codewords respectively, while the third and fourth pairs of blocks contain 12 data and 22 error correction codewords respectively. In this symbol, the character arrangement can be depicted as follows. Each row of the table corresponds to one block of data codewords (shown as Dn.”) followed by the associated block of error correction codewords (shown as En); the sequence of character placement in the symbol is obtained by reading down each column of the table in turn.”

The stored codeword sequence is therefore in order: D1, D12, D23, ..., D33, D45, D34, D46, E1, E23, E45,..., E22, E44, E66, E88. The calculation of the polynomial arithmetic for QR codes is done using bit-wise modulo 2 arithmetic and byte-wise module 100011101 arithmetic. This results in a Galois filed of 2^8 with 100011101 representing the field's prime modulus polynomial $x^8 + x^4 + x^3 + x^2 + 1$. The data codewords are the coefficients of the terms of a polynomial, sorted from first being the highest power term, to the last being the lowest. The error correction codewords are the remainder after a division of the data codewords with a polynomial $g(x)$ based on the version and level of the code. The extensive list of all thirty-one generator polynomials can be found in the ISO\IEC 18004:2000(E) QR code standard [15, pp. 67-73]. The remainder of this division results in the error correction codeword, again sorted by the order of coefficients, the highest being the first error correction codewords and the zero power coefficient being the last. As the full extent of the Reed-Solomon error correction method used for QR codes extends the scope of this paper, only a brief overview of the steps will be provided here. The Reed Solomon decoder attempts to identify the position and magnitude of t errors or e erasures, as well as correct the errors and erasures. The received codewords with errors $R = (r_0, r_1, r_2, ..., r_n)$, all being elements of $GF(2^8)$, are fed into the syndrome calculator, generating e syndromes, which only depend on the errors. The calculating is done by substituting the roots of the generator polynomial into the elements of R. Using these syndromes the locations of the errors can be calculated using Euclid's algorithm, which in turn enables the calculation of the size of the error. By adding the complement of the error size to the value at the error position, the error can be corrected.

For completion sake the 4 other types of QR codes described on the website of Denso Wave, will be quickly addressed in the following, however only model 1 and 2 are of interest for this paper. IQR codes are a further advancement to the structure by enabling not only rectangular codes in addition to solely square codes while also raising the theoretical maximum to version 61, which corresponds to a code of 422 by 422 modules. In a similar vein, Micro QR codes feature only one orientation pattern enabling the printing of these codes in spaces to small for regular QR codes. The so-called frame QR system allows for areas of the code to be left blank for logos or designs. Lastly, the SQRC (Secure Quick response codes) don't differ from model 2 codes in appearance, but require data keys to decode the information stored within correctly enabling the codes to be used store private information in order to use them, e.g. as a cashless payment system.

2.6 Visual marker problems

This chapter will cover aspects of images and QR codes that may influence the detection or decoding process. How the project software detect and subsequently deals with these problems will be addressed at a later point. In the figure 2.6 some of the issues with printed markers are displayed, from top left:

1. General noise and perspective distortion are just a factor of any form of photography used to scan the code no matter the system.
2. This image represents two different problems. On the one hand printing errors and on the other inconsistent lighting, both of which can influence the contrast of the code to its surroundings.
3. The third image is meant to show general image issues such as slight blur from e.g failing auto-focus or movement either by the camera or the code.

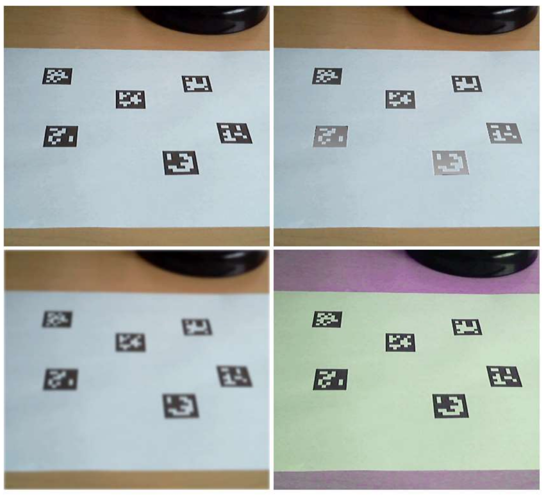
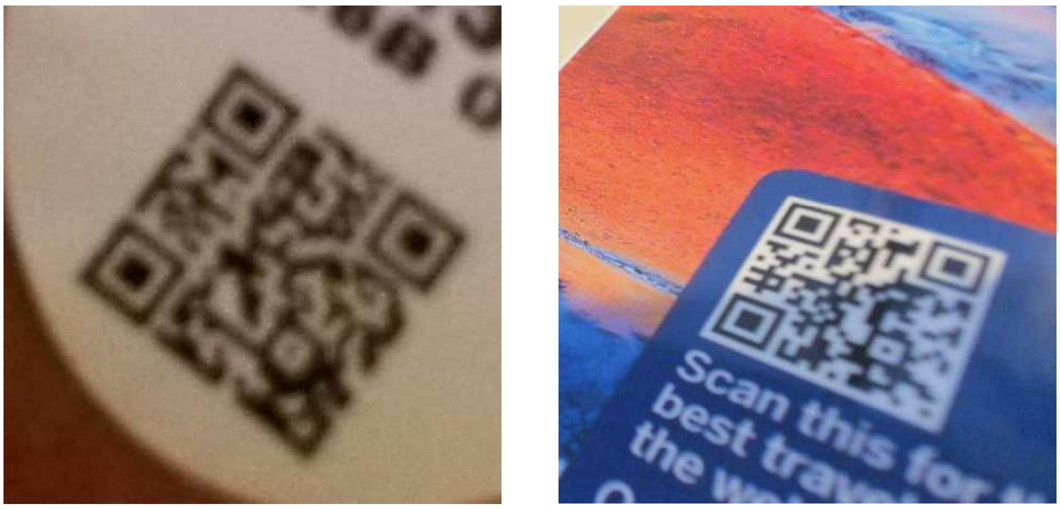


Figure 2.6: Issues with printed markers.

4. It cannot be guaranteed that a code will always be used in the ideal or even same environment, as such contrast changes from colored light or shadows might interfere with the detection.

Obviously, visual AR markers need to be recorded by cameras. This first step, before any image processing or decoding, can even take place, might already be flawed enough to prevent any successful continuation. Causes of errors range from user error in shaky footage and too large distance to the code to problems arising through poor lighting conditions, such as glare or shadows on the code to simply technical limitations or failure, in the form of defect sensors or an unclean lens. The main goal of this project is to determine whether or not these mentioned problems can not only measured and detected, but to what degree they can be corrected for in the detecting software. Unfortunately some of the problems share features in the recorded images. As an additional problem more than one image distorting problems might be present in an image, further exacerbating the gathering specific data. To combat this the images used for testing are specifically selected and digitally created to highlight one specific image feature, allowing for more precision in the data gained. The gained results are additionally compared to other tests, which shared the goal to find the limitation of the QR code detection algorithm. The images used for testing are included for the specific problem present in the image and allow for a theory on why the detection or decoding fails. Blurred images can fail in any of the steps required, dependant on the amount and type of blurring present in the image. Most light blur should be compensated for by the binarization of the image as this should restore edges although rather imprecisely. If the blur has a particular direction, i.e. motion blur this effect might be uneven across the code causing perceived shifting in the position of the modules and either failing to locate the code or overwhelming the error correction system. Uneven blurring across the code through the low depth of field or caused by lens-distortion correction algorithms, as seen in figure 2.7 (b) could lead to similar problems. The correct lighting conditions are crucial for the successful detection and decoding of QR codes. The selected images exhibiting



(a) (b)

Figure 2.7: Examples of blurred QR codes.

suboptimal lighting are therefore wider in scope. Shadows and glare on the code itself can cause the detection to fail if the position detection patterns are disrupted by them, as seen in figure 2.8 (a,b). Additionally, the overall brightness and corresponding contrast between the area of the QR code and the surrounding quiet zone is of interest as it might influence the quality of the code detection. Similarly, images with inconsistent lighting conditions cause problems through incorrect binarization. Dark images containing bright areas will represent this problem.

The next set of QR code problems are the result of not following the specifications set for better QR code detection. This includes codes with designs and logos within the encoding area, which should be correctable by to error correction system. This also includes colored codes or portions of codes that require the proper grayscaling method to ensure the light and dark areas of the code remain correct, shown in figure 2.9 (b), which contains a code with one position detection pattern in color. Additionally, redesigned position detection patterns may cause a failure to detect the code. To properly test the limitations of noncompliant QR codes deliberately broken ones are included in this category. This includes filling in quiet zones, areas of the code colored in or removed, disrupted position detection patterns, as well as multiple types of noise added to the code, for example of which can be seen in figure 2.9 (c,d). In this category, unintentionally broken codes are also covered. Scratches, marks or printing errors cause mistakes in the encoding region of the code which need to be corrected through the error correction. The last category of images deals with the incorrect usage of codes by displaying them in flawed ways. Displaying a QR code on a monitor will introduce noise and as can be seen in figure 2.10 (b) patterns which might cause the detection or decodation to fail. Also included are codes which are hindered in their detectability by the position of the camera. Codes viewed at to close of a distance will cause the required quiet zone to be obstructed as well as patterns and noise as a result of the print becoming visible, same as codes viewed at to great a distance causing the code to be potentially illegible. Rotated

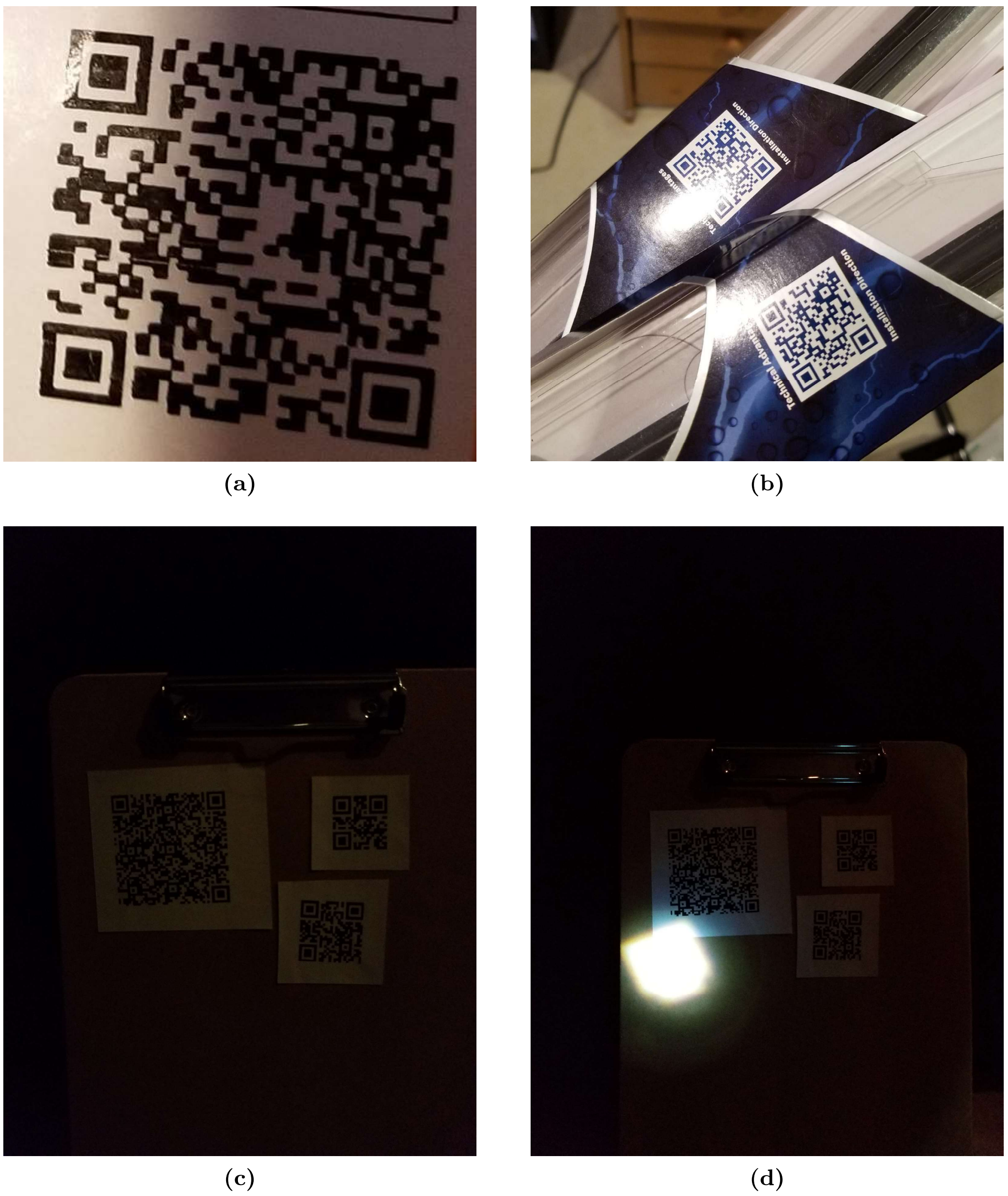


Figure 2.8: Examples of QR codes with flawed lighting conditions.

codes are included in this category, although the nature of the position detection system makes rotations around the normal of the code inconsequential. Viewing the code at an angle, however, may cause errors and failure at a certain angle. Lastly, codes displayed on curved surfaces may break the process of transforming the image data to a binary matrix. The goal of this project is to find ways of detecting these image problems within images and attempt to counteract their effects. This requires an understanding of the process by which a QR code is detected and subsequently decoded, which is explained

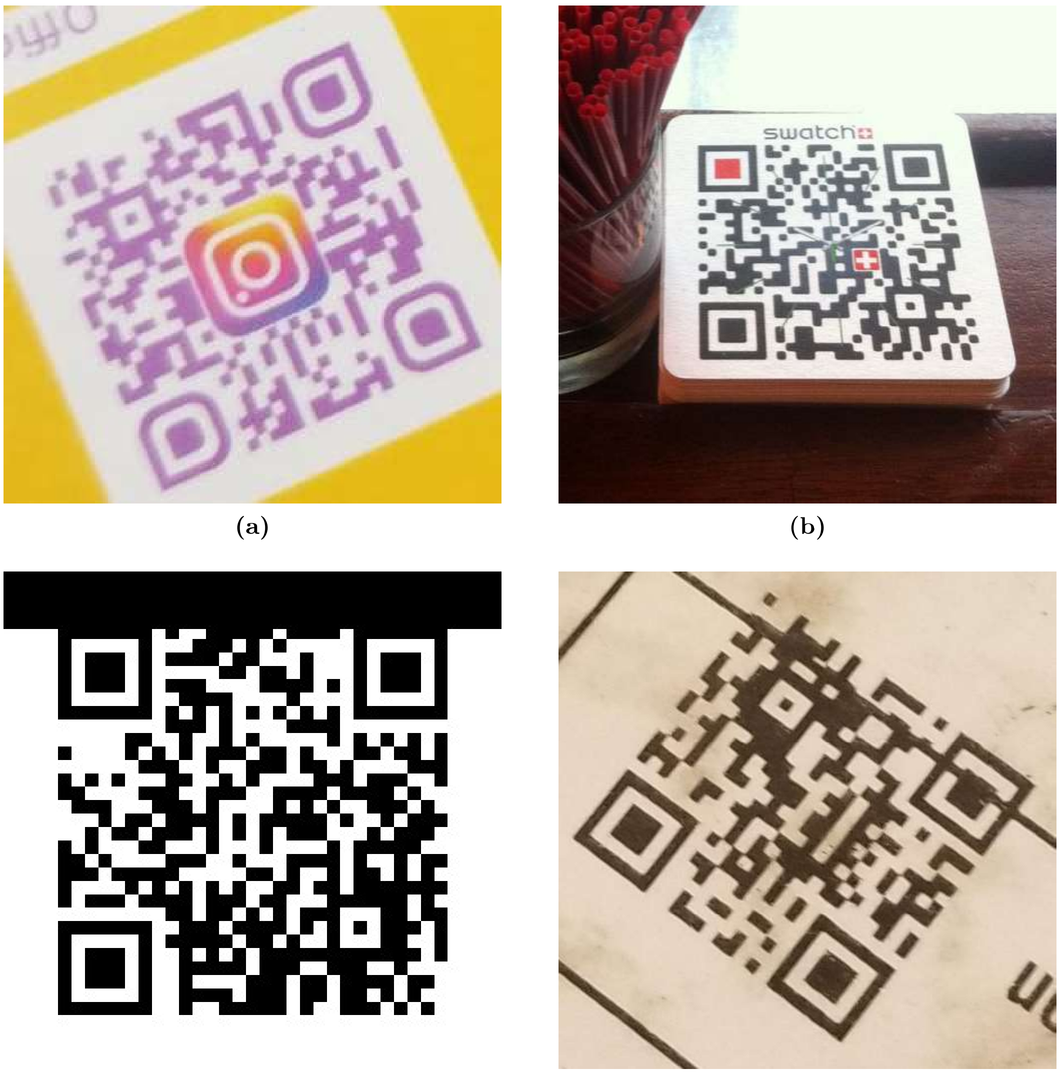
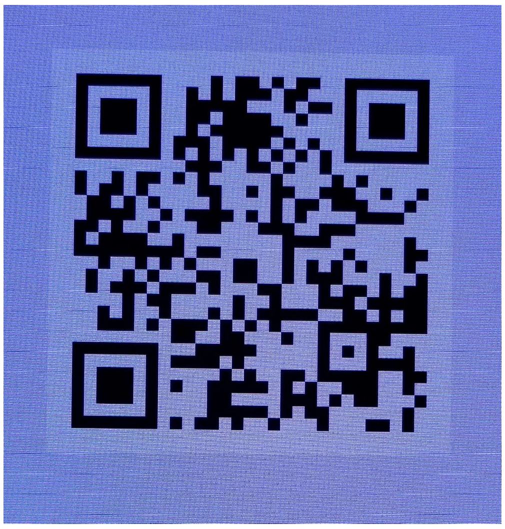
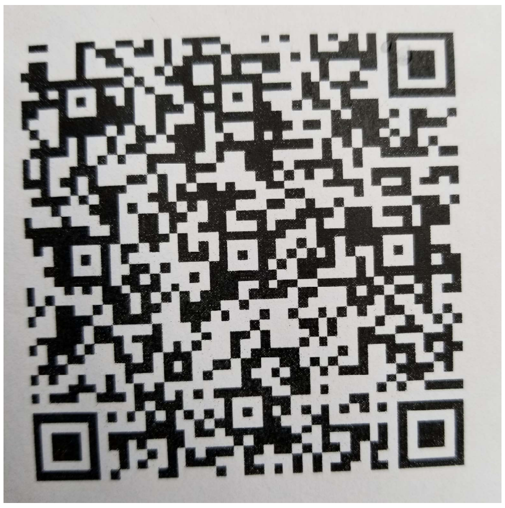


Figure 2.9: Purposeful misuse of QR codes.

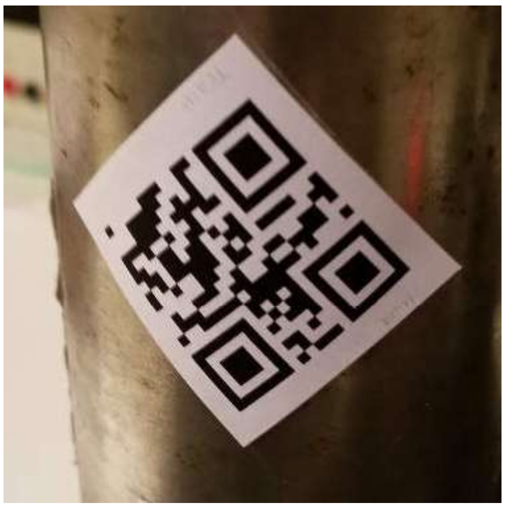
in the following chapter.



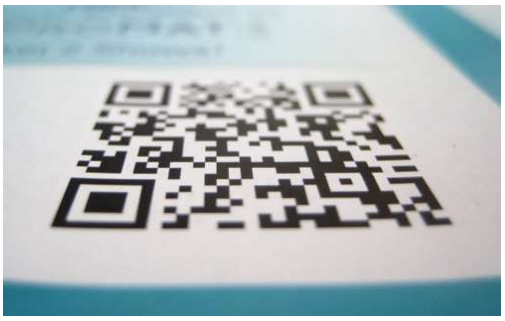
(a)



(b)



(c)



(d)

Figure 2.10: Examples of poorly positioned QR codes.

Chapter 3

Functionality of QR Codes

3.1 Introduction

The steps of the algorithm by which a QR code is analysed can be separated into three distinct sections:

- pre-processing,
- detection,
- decoding.

This represents an optimized implementation of the QR code detection process, as the ISO standardized detection does not require preprocessing and works on RGB images. This, however, requires the contrast between the code and its surroundings to be already high, which is one of the results of preprocessing the image. Each of these steps provides a rather specific output. The pre-processing produces a binary image, the detection process finds and extracts the code itself and provides a binary matrix of the QR code and lastly, the decoding step extracts the data and error correction codewords, and the encoding method from the matrix to generate the string of characters encoded therein. This structure could allow for each of the steps to be exchanged to suit an application, although the detection is married rather tightly to the decoding, through the structure of the code. It is, however, possible to adapt the decoding step to be used for different applications from simple data storage.

3.1.1 Pre-Processing

Grayscale

The goal of the preprocessing step is the transformation of the image into a more analyzable form, namely into a binary image or binary matrix. As an initial step, the original color image from the camera must be transformed into a gray-scale image. This transformation reduces the number of possible colors from 256^3 to 256 requiring a method to reduce the complexity wisely and not lose crucial image data. For this project, six different gray-scaling methods were implemented. The nomenclature was taken from Kanan and Cottrell's paper on the influence of the grayscaling method on image recognition quality [8, pp. 1-3]. Reasons for the inclusion of specific algorithms stem from either their special ability to emphasize certain aspects of the colors in the original image, their approach to emulating human color perception or their relation to other grayscaling methods that they seek to improve upon. What role the separate algorithms play in the detection of visual augmented reality markers is addressed in chapter 5, dedicated to testing and comparing the different approaches. This section only serves to provide an introduction to the math applied. In order to improve the readability of the gray-scaling section, the names of gray-scaling algorithms will be italicized and the first letter capitalized, e.g. *Luminance*. All of the following functions G require R, G, and B, corresponding to the color channels of the pixel operated on. The input values are stored in an 8bit form and may or may not require mapping to the range 0 to 1, dependant on the method. The output values are byte values as well and may require mapping to that number space as well. The first two color-to-grayscale algorithms are *Intensity* and *Gleam*. Both utilize a mean of the RGB channels:

$$G_{\text{Intensity}} = \frac{(R + G + B)}{3}, \tag{3.1}$$

$$G_{\text{Gleam}} = \frac{(R' + G' + B'}{3}, \tag{3.2}$$

the later however using gamma corrected values R', G' and B' of the input values. They were arrived at through the gamma correction function $\Gamma(t) = t' = t^{\frac{1}{2.2}}$. Unlike these initial two methods *Luminance* attempts to match human brightness perception by weighing the input channels differently:

$$G_{\text{Luminance}} = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B. \tag{3.3}$$

Luminance is a standard grayscaling algorithm frequently used in computer vision and image processing software. Similarly *Luma*, calculated as follows

$$G_{\text{Luma}} = 0.2126 \cdot R' + 0.7152 \cdot G' + 0.0722 \cdot B', \tag{3.4}$$

uses a weighted sum of the gamma-corrected RGB values to arrive at a solution. Of particular interest are the functions *Luma* and *Gleam* as the general goal of gamma correction is the approximation of the nonlinear brightness perception of humans from the linear digital data. The accurate representation of brightness, however, is not the primary goal of the testing but the maximizing of the contrast between the light and dark areas of the code is. As such experimentation with the gamma value used for the correction is a goal of the testing.

$$G_{\text{Value}} = \max(R, G, B). \tag{3.5}$$

Value represents the achromatic V channel of the HSV color space and provides an absolute brightness value for each pixel. On a similar note *Luster* corresponds to the L channel of the HLS(Hue, Lightness, Saturation) color space. It is calculated as the mean of the minimum and maximum RGB values,

$$G_{\text{Luster}} = \frac{\max(R, G, B) + \min(R, G, B)}{2}. \tag{3.6}$$

Luster is significantly less sensitive to outliers in color intensity unlike *Value*, which is maximised by one color channel being bright, *Luster* requires all three channels to be highly saturated in order to maximise its value. The *Lightness* grayscaling method is intended to closely correspond to human perception through a non linear transformation of the RGB color space into the CIELAB and CIELUV color spaces. The calculation of the transformation is done as follows,

$$G_{\text{Lightness}} = \left(\frac{1}{100}\right) \cdot 116 \cdot f(Y) - 16), \tag{3.7}$$

where the lightness nonlinearity $f(t)$ is calculated as

$$f(t) = \begin{cases} t^{\frac{1}{3}} & \text{for } t > (\frac{6}{29})^3, \\ \frac{1}{3} \cdot (\frac{29}{6})^2 \cdot t + \frac{4}{29} & \text{otherwise,} \end{cases} \tag{3.8}$$

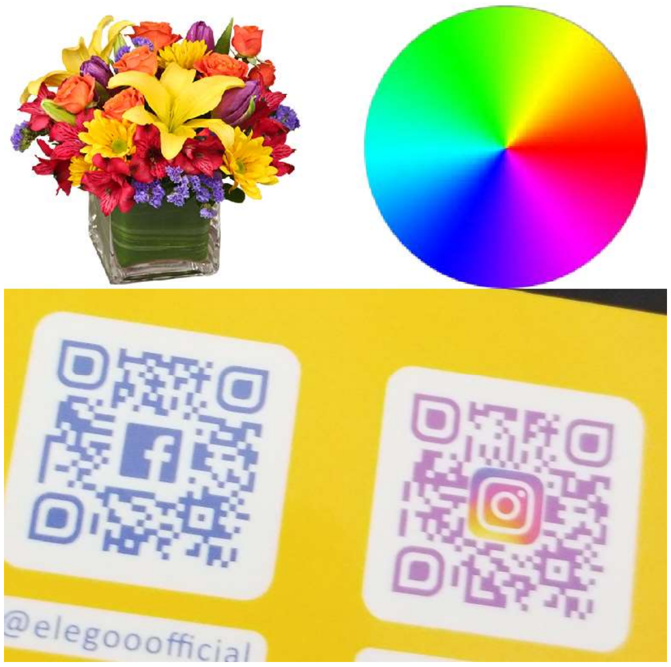
with Y being the perception equalized sum of the input values

$$Y = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B, \tag{3.9}$$

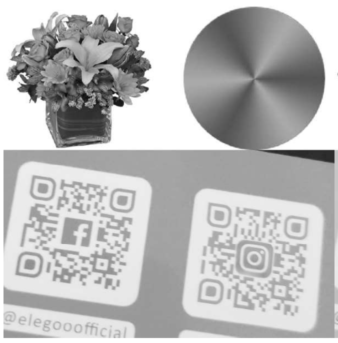
which results in a gamma-corrected value between 0 and 1. Examples of five of these grayscaling methods can be seen in figure 3.1, which shows *Intensity* b, *Luminance* c, *Lightness* d, *Value* e and *Luster* f. The inclusion of the HSV color circle demonstrate some of the grayscaling properties, such as the achromatic nature of *Luster* and *Value* resulting in a monochrome circle, or the *Intensity* being calculated from all three color channels equally, resulting in a repeating pattern unlike the other two methods, to which the colors contribute to different degrees. Lastly *Decolorize*, which was not successfully implemented for this project, also uses a different color space, in this case, the Y channel in YPQ space to derive its brightness. The exact implementation as described by Dodgson and Grundland exceeds the scope of this section. The resulting color channels, however “consist of an achromatic luminance channel $Y_{i,j}$ and a pair of chromatic opponent-color channels: yellow-blue (...) and red-green” [6, p. 7]. The resulting image contains innate gamma correction as well as preserving color contrast in its Y component. The design objectives set out for this grayscaling method include the magnitude of the grayscale contrast reflecting the magnitude of the color contrast in the original image [6, p. 4], which leads to the assumption, that it may provide a higher contrast between the light and dark QR code modules. At this point an image wide contrast in the form of the Root-mean-square (RMS) contrast could be calculated, to potentially judge the quality of the image or grayscaling method. This form of contrast evaluation is calculated from the sum of the deviation of pixel intensities from the average intensity of the entire image. The correlation between this value and the quality of code detection is difficult to establish, it can, however, serve as a data-point to compare the different grayscaling methods by.

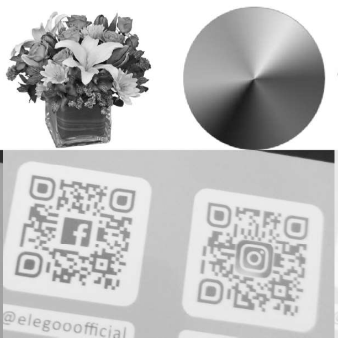
Binarization

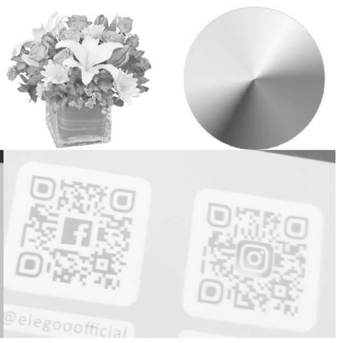
The last step to bring the image into a form suitable for QR code detection is the binarization of the image. The initial goal of image thresholding was the extraction of objects from their background in an image, the resulting images, however, serve the QR code



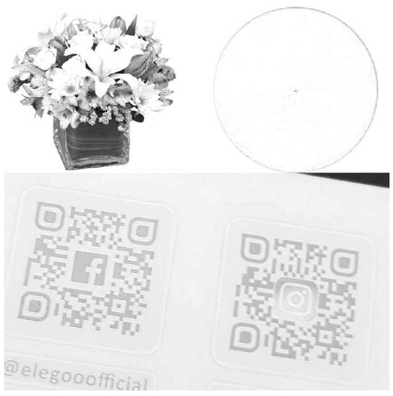
(a)

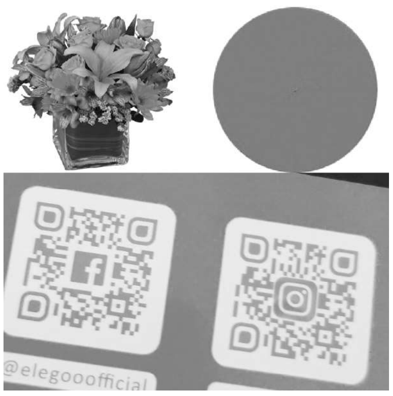






(b) **(c)** **(d)**





(e) **(f)**

Figure 3.1: Grayscaling methods used for testing in the project. In order, (b) Intensity, (c) Luminance, (d) Lightness, (e) Value and (f) Luster.

detection well as the code strongly differs from the surrounding area. Image thresholding or binarization can be separated into two distinct types of algorithms: global and local thresholding. Global binarization establishes a single value all pixels of the image are compared against and sorted into foreground and background pixels and colored black or white accordingly. Local or adaptive methods calculate multiple threshold values for smaller areas of the image, either segments of a grid applied to the image or for areas centered on each individual pixel. Generally speaking, adaptive binarization is computationally more expensive, for the advantage of being less susceptible to brightness changes over the image. For this project, two global and five local binarization methods were implemented and compared. The first and least complex method is to use the mean value T of the image I as a threshold. The image of size $n \cdot m$ contains pixels of gray values $L[1, 2, ..., L]$ denoted as $l_{i,j}$, i and j denoting the row and column position of a specific pixel. It follows that the global mean value of the image is calculated as:

$$T(I)_{\text{global\_mean}} = \frac{\sum_{i=0,j=0}^{n,m} l_{i,j}}{n \cdot m}, \tag{3.10}$$

which improves on using the median of the color space as a threshold, by allowing the overall brightness of the image to factor into the background/foreground separation. The second global thresholding implemented is a version of Otsu's thresholding method [10, p. 63], it proposes the generation of a threshold value through the analysis of the histogram of the grayscale image. The values l of the image I are transformed into a histogram of width L, which for this implementation is 255, for sake of storage as a byte. Subsequently, the histogram is normalized and for sake of the calculation regarded as a probability distribution using the following formulas 3.11 and 3.12. The number of pixels at a level h is denoted as n_h with the total number of pixels being $N = n_1 + n_2 +, ..., n_L$, calculated as:

$$p_h = \frac{n_h}{N}, \tag{3.11}$$

$$p_h \geq 0, \sum_{h=0}^{L} p_h = 1. \tag{3.12}$$

Presupposing two classes C_0 and C_1 of pixels in the image, background and objects, separated via the threshold at the level k, C_0 containing all gray value of the histogram with the levels [0,1, ..., k] and C_1 containing [k+1, ..., L]. It follows that the probabilities of class occurrence ω_0 and ω_1 and the class mean levels μ_0 and μ_1 are calculated as:

$$\omega_0 = \sum_{h=0}^{k} p_h = \omega(k), \tag{3.13}$$

$$\omega_1 = \sum_{h=k+1}^{L} p_h = 1 - \omega(k), \tag{3.14}$$

and

$$\mu_0 = \sum_{h=0}^{k} h \cdot \frac{p_h}{\omega_0} = \frac{\mu(k)}{\omega(k)}, \tag{3.15}$$

$$\mu_1 = \sum_{h=k+1}^{L} h \cdot \frac{p_h}{\omega_1} = \frac{\mu_T - \mu(k)}{1 - \omega(k)}. \tag{3.16}$$

The zeroth- and first-order cumulative moments of the histogram up to level k are calculated as,

$$\omega(k) = \sum_{h=0}^{k} p_k, \tag{3.17}$$

$$\mu(k) = \sum_{h=0}^{k} h \cdot p_h. \tag{3.18}$$

$$\mu_T = \mu(L) = \sum_{h=0}^{L} h \cdot p_h, \tag{3.19}$$

μ_T being the total mean level of the original picture. The measures of class separation, which are used to determine the quality of a selected threshold k require the variance of the classes σ_0 and σ_1:

$$\sigma_0^2 = \sum_{h=0}^{k} (h - \mu_0)^2 \cdot \frac{p_h}{\omega_0}, \qquad \sigma_1^2 = \sum_{h=k+1}^{L} (h - \mu_1)^2 \cdot \frac{p_h}{\omega_1}. \tag{3.20}$$

The measures of threshold quality referred to as the “goodness” of the threshold by Otsu [10, p. 63] are discriminant criterion measures λ, κ and η:

$$\lambda = \frac{\sigma_B^2}{\sigma_W^2}, \qquad \kappa = \frac{\sigma_T^2}{\sigma_W^2}, \qquad \eta = \frac{\sigma_B^2}{\sigma_T^2}, \tag{3.21}$$

where

$$\sigma_W^2 = \omega_0 \cdot \sigma_0^2 + \omega_1 \cdot \sigma_1^2, \tag{3.22}$$

$$\sigma_B^2 = \omega \cdot (\mu_0 - \mu_T)^2 + \omega \cdot (\mu_1 - \mu_T)^2, \tag{3.23}$$

$$\sigma_T^2 = \sum_{h=0}^{L} (h - \mu_T)^2 \cdot p_h. \tag{3.24}$$

The optimal threshold $k*$ is the value that maximizes η or σ_B^2. To arrive at that value a simple sequential search is performed to find the value which provided the highest class separation. Even though it is referred to as a global thresholding method, Otsu’s method could feasibly be applied to smaller sections of the image to compensate for lighting changes over the image. This would run the risk of creating unintended edges at the borders between two sections of the image as the threshold would most likely differ between them. Calculating Otsu for every pixel based on the surrounding pixels in a radius is technically possible, however computationally rather expensive due to the nonlinear time complexity. The nonlinear time complexity is a trait shared by the following binarization methods. Using the above-mentioned technique of calculating the mean of the image considers the global lighting condition of the image, as every pixel contributes to the threshold. The fact remains that a global threshold cannot correctly distinguish between fore- and background is the brightness of the image is not constant throughout because of, e.g, shadows or glare in the image. To compensate for this problem, local adaptive mean thresholding calculates the mean value of an area around each pixel. The threshold $t_{i,j}$ for each pixel of the image is calculated as such, with $\{d \in 2\mathbb{Z}+1\}$ being half the side length of the search area. d must be an odd number to ensure an even distribution of pixels around the target point. The calculation of the local threshold using r, half the diameter rounded, i.e., down,

$$t_{i,j} = \frac{\sum_{x=i-r,y=j-r}^{i+r,j+r} n_{x,y}}{2r^2}. \tag{3.25}$$

must be repeated for every single pixel $n_{i,j}$ in the image. In order to alleviate the problem of noise, a constant C is added to the calculated mean to ensure a more even binarization in areas of similar colors. This can correct for some of the problems, however only an increase in the size of the search area can improve the binarization quality significantly. This however becomes very expensive to compute very rapidly. As an cheaper approximation adaptive median thresholding only calculates the median value of an area around each pixel as a threshold, this results in a cheaper to calculate, but less accurate binarization method. A trade-off which applies to most of the methods proposed in the literature. The Gaussian adaptive thresholding extends the adaptive mean threshold by, weighing the individual pixel values in the search area not equally, but based on their distance to the primary pixel, based on a Gaussian distribution. Bernsen's locally adaptive binarization method [7] uses the range between maximum and minimum values in a local window to establish a threshold. The implementation follows Can Eyupoglu's implementation [5, p. 622], who used the method for the extraction of text from images. The method requires a radius r similar to the previous methods, as it is applied to an area around each pixel just the same. Initially the highest Z_{high} and lowest Z_{low} values of the area are gathered. The threshold t value and the measure of the contrast c are calculated for the area centered on pixel $n_{i,j}$:

$$t_{i,j} = \frac{Z_{low} + Z_{high}}{2}, \tag{3.26}$$

$$c_{i,j} = Z_{high} - Z_{low}. \tag{3.27}$$

Using these values the pixel $n_{i,j}$ is labeled as fore- or background. By calculating a basic measure of contrast the downfall of locally adaptive thresholding methods, large areas of low contrast can be detected and false positives avoided. This method is similar in time complexity to using the median of the area as a threshold. Lastly, a considerably cheaper method was implemented. Instead of using an area around each pixel as a measuring area for the threshold, this method separates the image into a grid, calculating a mean threshold for each segment. In order to avoid sharp edges at the borders between two grid elements the threshold for a specific pixel is calculated from the four surrounding grid centers, and linearly scaling the respective thresholding values based on the distance of the pixel to the grid center. The second variant of this grid-mean approach is used as well, which differs in two points. One of the implementations uses a constant size for the grid elements, while the other uses a constant number of grid elements. The second implementation determines the threshold not via linear interpolation, but as the average of the 3×3 grid neighborhood. All the steps in the preprocessing of the image need to be applied carefully, as no reduction in data complexity can occur without some overlap in results, causing errors in the following steps. Given the wide array of possible image manipulations that can occur in the preprocessing, particular attention needs to be paid to how these processes influence the following detection step. This is also the reason why a plurality of the testing done occur ed for the image manipulation in the first step.

3.2 Detection

The initial detection of a QR code in an image is done by parsing the image for the position detection patterns, seen in figure 2.1. Many approaches exist to speed up this step such as using feature detection [2] or edge detection [13]. As detection speed is not the goal of this project, but accuracy is, the implementation follows the reference decode algorithm given in the ISO specification for QR Codes [15, p. 60]. The initial step in the reference algorithm, suggests using the median of the minimum and maximum reluctance value of the image as a threshold. The thresholding, however, is covered in the previous section. The first step of the detection requires the detection of the position detection patterns. The patterns are rotationally invariant in ratio, allowing for the parsing of the image line by line, or more precisely every third line, as the center block of the pattern needs to be 3 modules wide, requiring the smallest possible code that can be displayed in an image, to have position detection patterns with a center of at least 3 pixels in width and height. To compensate for angular distortions a tolerance of half a module width is possible. When the ratio of 1:1:3:1:1 of dark:light:dark:light:dark pixels is found, the position of the first and last points within the pattern are established and stored. The distance between these points is stored and used as an initial rough estimate of the size of a position detection pattern. The parsing for the patterns is repeated for the columns between the edge points established before, establishing vertical boundaries for the pattern. The horizontal parsing is repeated within these borders, enabling the calculation of the center point of the pattern, as well as an estimate of the size of a module in pixels, as a seventh of the width of the position detection pattern. This process is repeated until three or more position detection patterns are found. Should more than three patterns get found in an image, the three patterns are selected which have the closest module sizes. In order to establish the orientation of the code, the relation between the position detection patterns has to be analyzed. Given the structure of a QR code the distance between the top right and bottom left detection pattern, must always be bigger than the distance to the third pattern. Assuming the code to be not mirrored only one arrangement of the three patterns enables this to be true, allowing construction as it is implied by the orientation of the finder pattern. With the position detection patterns found, a provisional version V of the code can be calculated as

$$V = \frac{\frac{d}{X} - 10}{4}, \tag{3.28}$$

where d is the distance between centers of the upper left and upper right position detection pattern, and X being the nominal X dimension of the symbol, calculated from the widths w_{ul} and w_{ur} of the same patterns

$$\frac{w_{ul} + w_{ur}}{14}. \tag{3.29}$$

Dependant on the version calculated this way the algorithm continues differently. Should the version be 6 or less, this is specified as the defined version. If the version is higher the detection process is extended by a detection step for the alignment patterns. Using the positions of the position detection patterns a provisional position of the alignment patterns is established and the image region around that position is parsed for the 1:1:1:1:1 dark:light:dark:light:dark module pattern to find the exact position of the alignment pattern. Should this not be successful the estimated position is used in the following step. The ISO standard for QR codes suggests applying a grid for each section separated by alignment patterns, with the intersections of the grid-lines coinciding with the centers of the modules of the codes segment. By sampling every intersection of the grid-lines and determining the pixel at that position, a binary matrix can be constructed by mapping dark pixels to a binary 1 and light pixels to a 0.

3.3 Decoding

From the previously generated bit matrix the format information encoded adjacent to the upper left position detection pattern is read and decoded in order to receive the error correction level and the mask pattern. As an additional safety measure, these properties are encoded two more times adjacent to the other two position detection patterns. In the next step, the mask pattern is applied to the encoding region of the bit matrix, in the form of an XOR condition on each dark module of the mask pattern and the bit matrix. The data stored in the QR code are position in two-module wide columns beginning in the lower right corner and alternating upwards and downwards from the right to the left side. The block interleaving described in chapter 2, section 2.5.1 needs to be reversed in order to arrive at the final bit stream. This data containing data and error correction codewords, and potential errors and erasures, is subjected to the error correction process described in the previous chapter to restore the original bitstream message intended to be stored in the code. This bit-stream is divided into segments corresponding to the mode and character count indicators contained within every single segment. Each segment is then decoded according to the rules for the mode.

Chapter 4

Implementation

This chapter covers the implementation of the software produced in conjunction with this thesis, as the program was used to generate the testing data shown in the following chapter. As the process by which a QR code is detected in an image is described at length in the previous chapters only the aspects in which the implementation differs from the ISO standard will be explored here.

4.1 Libraries

Libraries used for this project are OpenCV, BoofCV, and ZXing ("Zebra Crossing") for their image processing and QR Code detection capabilities. Additionally, JavaFX was used to create the user interface and to display data generated by the software. The full extent of the capabilities of the computer vision libraries vastly exceeds what was necessary for the project. Therefore the decision was made to create a separate implementation of the QR code detection, allowing for more control over the process and also easier access to exact values used in the detection and decoding. As an added factor the QR code detection and decoding of BoofCV and ZXing are designed to be used for other means, therefore the reason for failure difficult to extract from the process. The final implementation draws heavily on the ZXing implementation, although heavily altered in the preprocessing step, as well as the added ability to persist more of the data relevant for the detection and decodation process, to allow for partial finds of codes existing. Aspects kept in almost their entirety are the error correction process, which needed to be adapted to fit the altered steps leading up to it, as well as the classes which store QR code relevant data for damasking, version information, format information, error correction level information and the positions of alignment patterns for the respective version of QR code. These data classes cannot be altered as they are required by the QR code standard and successful detection and decoding would not be possible without them. As for the use of BoofCV and ZXing they primarily served as a starting point to learn about the process by which QR codes are detected. BoofCV is the more powerful, not only code detection system, but also general computer vision library. Its wider scope makes it more difficult to follow the flow of information through the detection process. ZXing, on the other hand, proved easier to understand in its implementation, making it the primary source to draw from for implementing a very specialized QR code detector. Both libraries are designed for integration in other software making functionality the primary goal and failure an edge case. As failing codes and the reasons for that failure is the primary interest of this project, work had to be done to ensure every failure state is properly documented in order to aid the analysis of code detection failing. OpenCV is perhaps the more well-known computer vision library and it was initially used for its computer vision and image manipulation capabilities. With the adaptation of the ZXing implementation, it lost some of its purposes in the project as the image classes were adapted to serve the project better. It is now mainly used for gathering still images from a live video feed via an external camera or from a video file. The goal was to use this to increase the speed at which images can be analyzed as a video could contain a stream of different frames containing relevant information. This proved to be a poor approach to testing multiple images as the video stream parsing did not perform as expected as it could not be slowed down or paused in a way that was conducive to the overall goal, making the testing with slower methods of binarization for example impossible. The loading of video files did provide some valuable insight as it allowed for the testing with moving QR codes, codes changing sizes or the image gradually being blurred more. The goal of being able to quickly test multiple images was achieved by allowing the selection of multiple files when loading images into the program.

4.2 Architecture

In order to allow for the testing of BoofCV, ZXing and the new implementation, as well as specialized trials, tester classes were created which handle both the calling of the implementations which need to be tested, as well as compiling a report dependant on what is tested. All the QR detection algorithms were embedded to allow the creation of a report which contains as much information as the algorithm can return. The BoofCV implementations provide the positions of the finder-pattern and alignment pattern as well as the version and the encoded text as well as the corner points of the code for easier visualization. Additionally, it includes a failure cause, which only documents the step at which the code failed. The ZXing algorithm returns the positions of the finder pattern and the encoded message. It also contains the raw byte data as well as information on the barcode format of the result, as the same result class is used for any of the 2D barcodes ZXing can decode. The custom detection algorithm allows for more as it was designed to provide more information. This includes all found pattern structures in the code, the binary matrix form of the found code, the error corrected code in bit matrix form, the message stored within and the format and version information of the code. Unlike the other two implementations, a report can be generated from a failed detection as well, allowing for a report containing only two position detection patterns to exist. Comparing the amount of found position patterns over different preprocessing scenarios provides additional data aiding in the analysis. This setup of using tester classes in order to create specific reports for tested situations allows for i.e. the comparison of two detection methods, testing of multiple different grayscaling methods or just the detection of a single QR code in a single image by a single implementation. It is close to a Facade pattern in its structure, as the tester classes serve as a connecting point between the user interface and the underlying logic of the QR code detection systems. Additional a

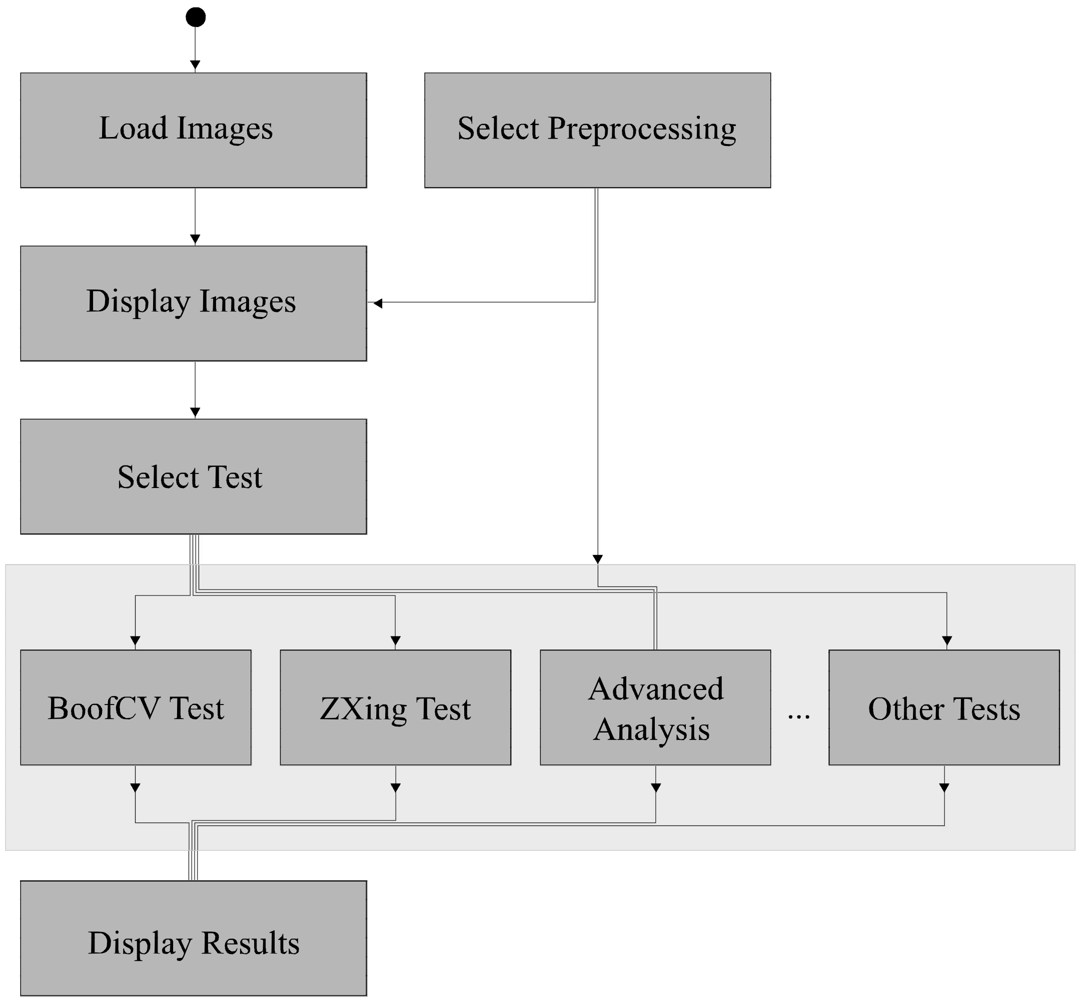


Figure 4.1: Process flowchart of the implemented software for testing images.

class for documenting the goings-on to aid in debugging was implemented as a singleton to be accessible anywhere in the software, without slowing down the program through system print output at runtime. This logging class is also used to document any failure state in the detection and decoding process, including which step of the process failed and the reason for that failure. The user interface serves to enable the customization of the testing processes as well as displaying the results of these tests. One particular goal was to display the binarized image before beginning the test allowing for informed testing, as some combinations of grayscaling and thresholding will result in nonsensical images, which cannot result in successful detection.

4.3 Optimization

The generation of the binary matrix containing the QR code after the detection process differs from the method proposed in the ISO standard, of using the found position detection and alignment patterns to position a grid over the image with each intersection of grid-lines corresponding to the center of one module of the code. Based on the version of the code and the number of alignment patterns in it, the code is separated, with the alignment patterns separating the sections. The resulting grids are parsed with the color of the pixel at each gridline intersection being compared to a threshold value, calculated as the median of the highest and lowest brightness value of the still color image. The generation allows for an anisometric grid, however, no angular distortion can be accounted for with the proposed system, as no method of pixel selection for grid intersections is described in the standard. To account for distortion in the image the parsing of the modules constituting the code is not done via a grid, but by calculating a transformation matrix to transform the code into a binary matrix of the dimensions of the code. This allows for the correct parsing of trapezoidal shapes, likely to result from viewing the square code at an angle. To calculate this transformation 4 pairs of points are needed, as the points in the desired bit-matrix can be considered as transformations of the original points [4, p. 381]. For this process, the center of the position detection patters, as well as the center of the alignment pattern are used. These points are detected in the original image, the pairings for these points are the locations these points are mapped to, which can be seen in figure 4.2 showing the two quadrilaterals Q1 and Q2 used for this transformation. The pairs of points are notated as x_1 to x_4 as the points in the original image, and the transformed points as x'_1 to x'_4.The transformation can be described as

$$x' = M \cdot a, \tag{4.1}$$

or

$$\begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 \cdot x'_1 & -y_1 \cdot x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 \cdot y'_1 & -y_1 \cdot y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 \cdot x'_2 & -y_2 \cdot x'_2 \\ 0 & 0 & 0 & x_1 & y_2 & 1 & -x_2 \cdot y'_2 & -y_2 \cdot y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 \cdot x'_3 & -y_3 \cdot x'_3 \\ 0 & 0 & 0 & x_1 & y_3 & 1 & -x_3 \cdot y'_3 & -y_3 \cdot y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4 \cdot x'_4 & -y_4 \cdot x'_4 \\ 0 & 0 & 0 & x_1 & y_4 & 1 & -x_4 \cdot y'_4 & -y_4 \cdot y'_4 \end{pmatrix} \cdot \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{pmatrix}. \tag{4.2}$$

One particular point can be calculated as

$$\begin{aligned} x'_i =& a_{11} \cdot x_i + a_{12} \cdot y_i + a_{13} - a_{31} \cdot x_i \cdot x'_i - a_{32} \cdot y_i \cdot x'_i, \\ y'_i =& a_{21} \cdot x_i + a_{22} \cdot y_i + a_{23} - a_{31} \cdot x_i \cdot y'_i - a_{32} \cdot y_i \cdot y'_i, \end{aligned}$$

which allows the creation of a system of linear equations through which the unknown parameters a can be solved. The transformation is applied row by row, with the end-points of the resulting array of points being controlled, as imperfect detection of the finder patterns could cause a slight shifting of the pattern. Points that fall outside the

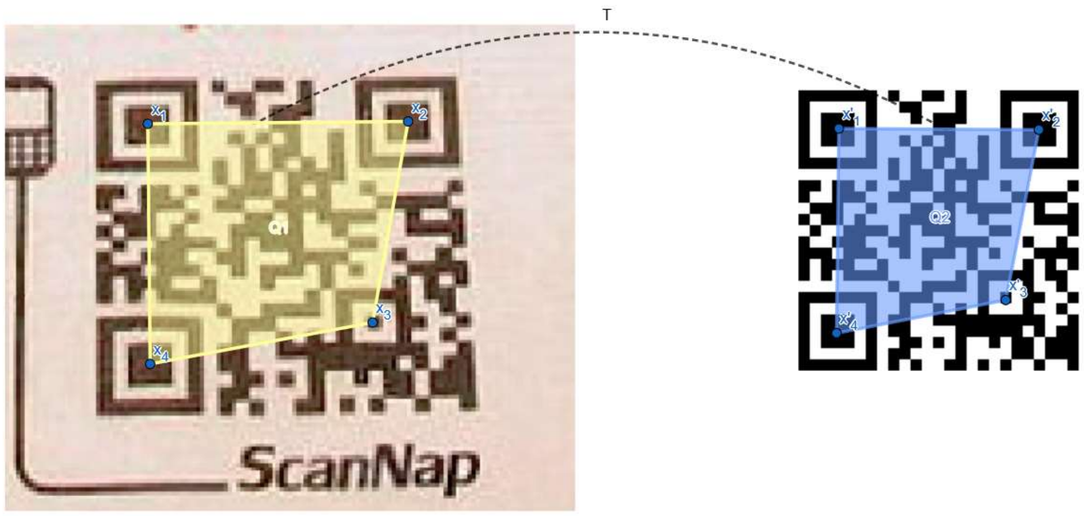


Figure 4.2: Example of the points used for transformation.

image by less than one pixel are moved to be back in the image, which is valid as the list of points is linear. For the sake of efficiency, the list of points is analyzed from the ends inward until a pixel falls inside the image. The entire process is repeated for every row in the code until a binary matrix of the correct size for the code version is created.

4.4 Time complexity

All of the described grayscaling methods have a linear time complexity as the gray output value of each pixel is calculated from the three color values of the same pixel, requiring every element to be calculated once. The optional denoising steps of either median or mean filtering, are non linear functions as they require the calculation of a mean or median of the neighborhood for each pixel, causing the resulting per pixel complexity to be $O(r^2 \log(r))$ with r being the radius of the filter kernel, which was selected as 1, as it is only used for salt and pepper noise removal. The same applies to the mean filtering, as is also requires all pixels of the pixel neighborhood to be looked up. The same time complexity applies to the image sharpening filter, as it requires the calculation of values for each pixel based on the filter kernel. There are proposed solutions for median filtering in constant time [11], as well as improvements on the calculation speed of the image improvement methods. This, however, was not implemented as time efficiency is not the goal of implementation, but the influence of the process on the detection quality. These improvements may be relevant should the tool be applied to, i.e., video-streams significant increases in calculation time would be of interest. A similar sentiment applies to the implemented binarization methods. The global mean thresholding has a linear time complexity of $O(n)$ as the mean of all pixels has to be established. The application of the threshold to the image is not part of this or the following calculations as it is the same for every single algorithm in its linear time complexity. The time complexity for Ostu's method [10] is dependant on the number of bins in the histogram required to calculate the threshold and is, therefore, dependant on both the size of the image and the amount of different gray values in it. It is calculated as $\max(O(n), O(k^2)$, with k being the number of gray values possible in the input image. The local thresholding methods all require quadratic time complexity based on the size of the pixel neighborhood in which the threshold is calculated from. It follows that the time complexity is $O(n \cdot r^2)$ as the threshold has to be calculated for every pixel from its surrounding area. This applies to the adaptive mean, adaptive median, adaptive Gaussian thresholding methods as well as Bernsens's technique. A possible workaround is a clustering approach to these local methods. Segmenting the image via a grid and calculating the threshold for each grid segment separately reduces the computational effort significantly. To prevent sharp edges at the grid borders, the threshold values for every single pixel are linearly interpolated based on their distance to the nearest grid centers. This allows for linear time complexity as every pixel is only involved once in the threshold calculation.

4.5 QR code failure

This section will go over the entire process of detecting and decoding a QR code in order and most importantly describe the conditions, by which each step can fail. This is done in preparation for the following chapter which describes the results of attempting QR code detection on a variety of flawed images. Figure 4.3 shows the steps taken for the QR code detection and decoding process, starting with the pattern detection. Looking at the chart in detail it can be seen that it begins with a binary image. The methods applied to arrive at this image are omitted from the chart as well as this section of the paper as they were described at length in previous chapters and can not fail pre se, but can however produce nonsensical images, may cause failure during future steps. The pattern detection step scans the image for the position detection patterns to establish location information in order to proceed with the transformation. The first failure state which can occur is the image size being smaller than 23 by 23 pixels, as that is the smallest image that could house a QR code of 21 by 21 modules and one pixel for contrast around the edges. This can only occur with doctored digital images and would never result from a recording, but is included for completion's sake. The next set of failure conditions all come from the amount of position detection patterns found in the image. Less than three patterns being found are recorded and the detection process stopped as the following steps require three positional patterns exactly. If more than three patterns are found the module sizes of the patterns are compared and the three patterns with the closest matching size are selected. This can fail if the size differences between the patterns exceed a certain value, at which the perspective distortion would be too great to ensure correct decoding. Assuming the correct three patterns were linked to the module size for the entire code can be calculated by counting the black and white changes between the positional patterns. If this fails the average module size of the position detection patterns is used for the entire code. Should the size of the detected modules differ by a large amount the process stops as the found patterns do not belong to the same QR code and were matched by either incorrectly connecting patterns from multiple codes in the image or false positives arising in noise or background patterns. By dividing the distance between two position detection patterns by the module size the number of modules can be estimated and with it the version of the code. The position of the alignment patterns

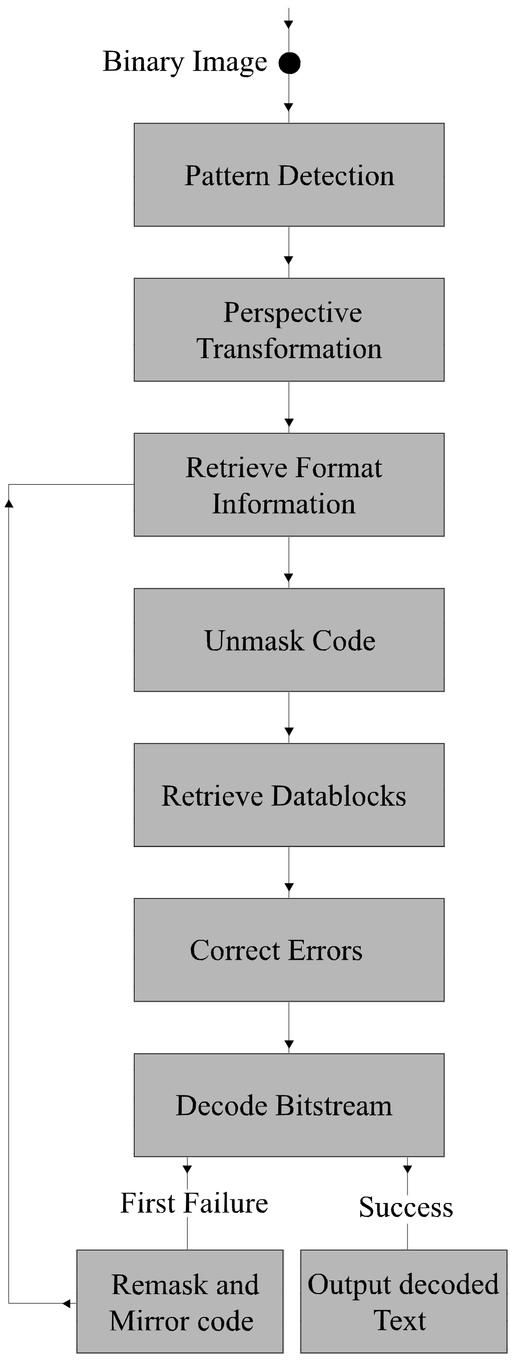


Figure 4.3: Flowchart displaying the process of detecting and decoding a QR code in a binary image.

are dependant on the version of the code, making the detection of the simpler black and white sequence more feasible. This can still fail as it requires the position of the alignment pattern to be estimated based on the position of the locator patterns, which will cause decoding problems if the code is distorted. Figure 4.4 is used to visualize the detection and decoding process showing the results of the image transformations, a being the binary input image, arrived at by using *Intensity* gray-scaling and a global Otsu thresholding, and 4.4 (b) showing the binary matrix after the transformation. The calculation of the perspective transformation matrix can not fail destructively, although the calculated matrix may be incorrect for the QR code if the patterns were mapped incorrectly, causing the bit-matrix to be generated by sampling the wrong area of the image. Should this be the case the following step will fail as the reading of the metadata from the code will not match the code in the image. Only codes version 7 or higher contain the version information bits in the code as this is also the first version of code containing more than one alignment pattern and with that multiple blocks of data. The 18 bits of version information are extracted from the matrix and decoded to find the closes matching version. Up to three bits of error is tolerated here as no two version info codewords differ in less than 8 bits. The same is repeated for the second copy of the version information at the diagonally opposed position detection pattern. If none of the versions found this way match the dimensions of the code the detection must stop as no information on the data block alignment can be gathered. The format information is encoded in 5 bits, the first three describing the data mask and the last two the error correction level. In order to compensate for errors in this information ten bits of error correction data are included with the format information, allowing for the calculation of a Hamming distance between the bit-string and the entries in the associated format information lookup table, from which the closes matching entry is selected. Should no match with a distance of less than 3 be found the detection is considered a failure. In the following step the binary matrix is unmasked, the result of which can be seen in figure 4.4 (c), by flipping bits of the matrix where the condition of the data mask found in the previous step, is true. A function pattern that includes the finder pattern, separator and format information, the alignment patterns, the timing patterns, and the version information areas is created and every bit of the matrix not contained therein considered as the data area of the code. Following the data arrangement of the version of the QR code, the bit data is extracted from the data area and based on the version and split into blocks. The error correction is applied block by block until the entire bit sequence is corrected. The error correction can fail if the number of errors exceeds the correction capabilities of the level, which manifests itself in the calculation of the error positions failing. In the following decoding of the bit-stream errors generally, originate from a miss-match between the number of bits read at one time and the number of bits required for the decoding mode. This overview of the failure conditions of the QR code detection algorithms makes the necessity of analyzing the input image apparent, as errors at the initial steps can propagate through the entire system. Finding correlations between specific problems with the image and the point a code fails at in the detection and decoding process is the goal of the testing in the following chapter.

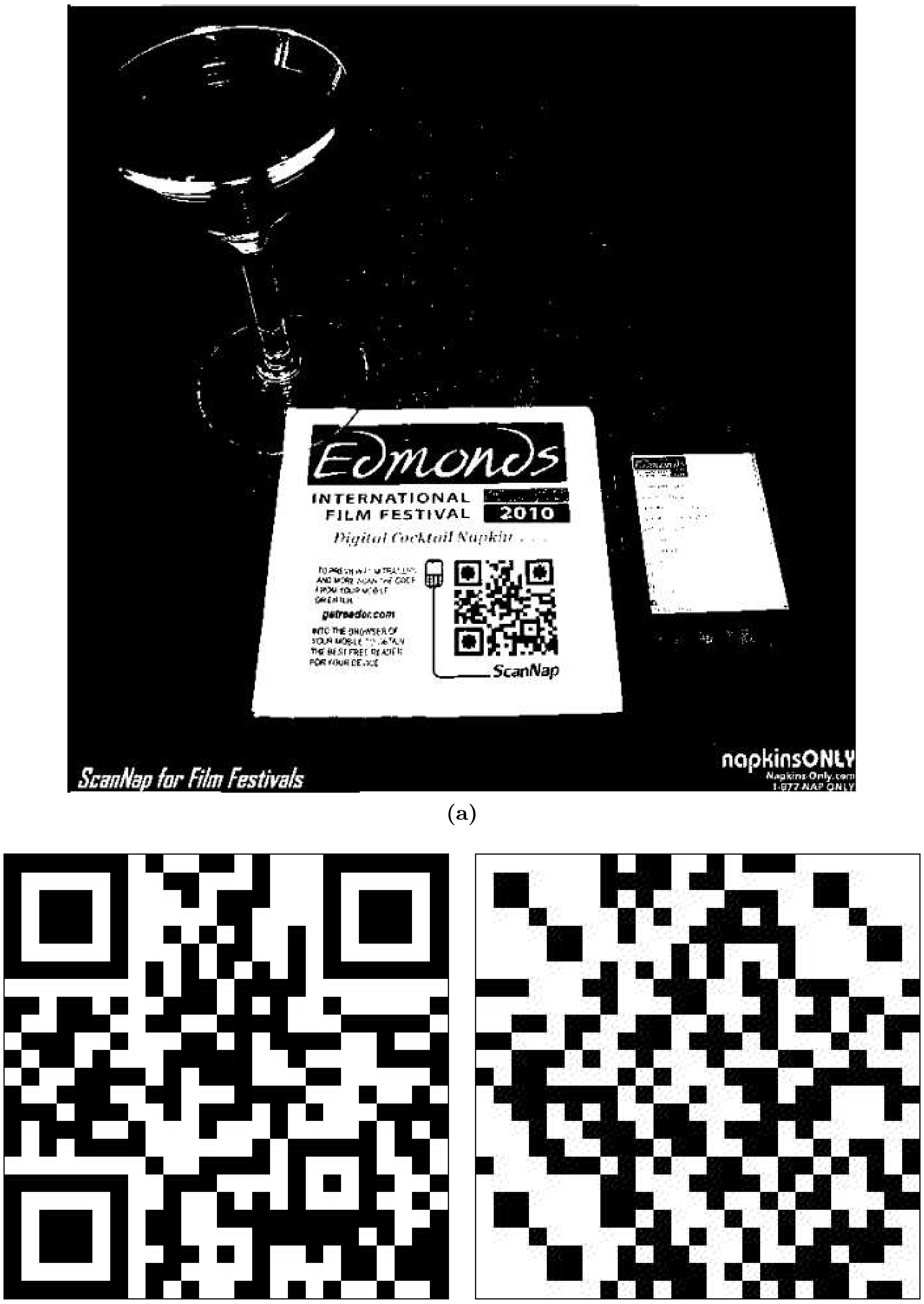


Figure 4.4: Example of the results of the demasking. The first image (a) shows the binarized version of the image, (b) the code extracted from the image and (c) the binary matrix after applying the datamask to it.

Chapter 5

Evaluation

5.1 Introduction

Continuing on the changes made the evaluation of the project also needed to be adapted to the changing goals. As a result, the goal of the evaluation is to find significant differences in the detection quality between different approaches to steps in the algorithm. This allows for assumptions to be made on what might impede the detection in the image as the different approaches may accentuate or lessen certain image statistics. The testing was done using the same image set used by the creator of BoofCV to compare their QR code detection to four different detection algorithms [14]. The image set contains approximately 450 images of one or multiple QR codes in a variety of situations. The application used for testing is not designed to detect multiple QR codes in an image, therefore images containing multiple codes were either not used for testing or edited using image manipulation software to remove codes while attempting to keep the overall brightness of the image approximately the same. The images range in size from approximately 550×300 pixels to 4000×4000 pixels. This size disparity revealed the first aspect which required testing.

5.2 Code size

In the initial testing phase, the problem of large images requiring significantly more time to calculate thresholded images became apparent, this could be corrected through better-optimized forms of thresholding. The size of the image appeared to influence the chance of detection, however. The following test will attempt to find if the size of the image influences the chance of finding the position detection patterns of a QR code. The testing set contains copies of the same image at full and half size, which did not get achieve detection at the same rate. This leads to the first question of how does the size of the QR code in the image in both dimensions in pixels, as well as the percentage of the whole image affect the detection process. To increase the amount of test possible with the limited testing data the images used for testing which contain more than one QR code were manipulated as to make the detection of all but one code impossible by disrupting the position detection pattern, repeating for each code present in the image. This allowed for the doubling of the testing data for the first evaluation. The

	Global Mean	Adaptive Mean	Grid mean:size	Grid mean
All Found	6	6	5	6
2 found	0	0	1	0

Table 5.1: Results of testing 6 images of different sizes.

	Global Mean	Adaptive Mean	Grid mean:size	Grid mean:amount
All Found	4	6	10	9
2 found	1	0	0	0
1 found	2	2	0	0
failure	3	2	0	1

Table 5.2: Amounts of position detection patterns found in images of differing sizes.

first test serves as a control group, using two copies of three images showing two codes at even lighting conditions, an example can be seen in figure 5.1. It shows a cropped version where the code takes up a bigger percentage of the image while keeping the pixel dimension of the code equal. The testing is done using *Intensity* grayscaling and four different binarization methods:

1. Global Mean,
2. Adaptive Mean,
3. Grid Mean: Size,
4. Grid Mean: Amount.

Adaptive mean thresholding is the least susceptible to brightness inconsistencies and therefore serves as a control group as it should have a higher success rate than global mean in unevenly lit images. Lastly, two implementations of a grid thresholding algorithm are compared, one using a fixed grid-size the other using a fixed amount of grid elements. The theory would suggest that grid-based thresholding would be the most affected by changes in the size of the image. The following table documents the results of the testing, by counting the amount of position detection patterns were found correctly for each method. The one failing detection in the grid mean thresholding with constant grid size seems to arise from the border between grid elements coinciding with the position detection pattern causing a slight amount of noise and disruption. To achieve more data on the binarization process the same set of binarization methods was tested on 5 images containing two QR codes at different image sizes or codes taking up different amounts of space in the image in relation to its size. As an added challenge the codes are lit differently, one falling in shadow, while the other is lit directly. Examples of the images used can be seen in figure 5.2, both of which exist in multiple version of different sizes. Additionally the images were duplicated and one of the codes made undetectable, in order to ensure no incorrect pattern matching. The results of this test were rather surprising as the detection of the position detection patterns failed for adaptive mean binarized images, especially images in which the code size in pixels was particularly high. The reason for this becomes apparent when analysing the binary images, as well as the thresholding method closer. Looking at the binary image in figure 5.3 (b) it can be seen

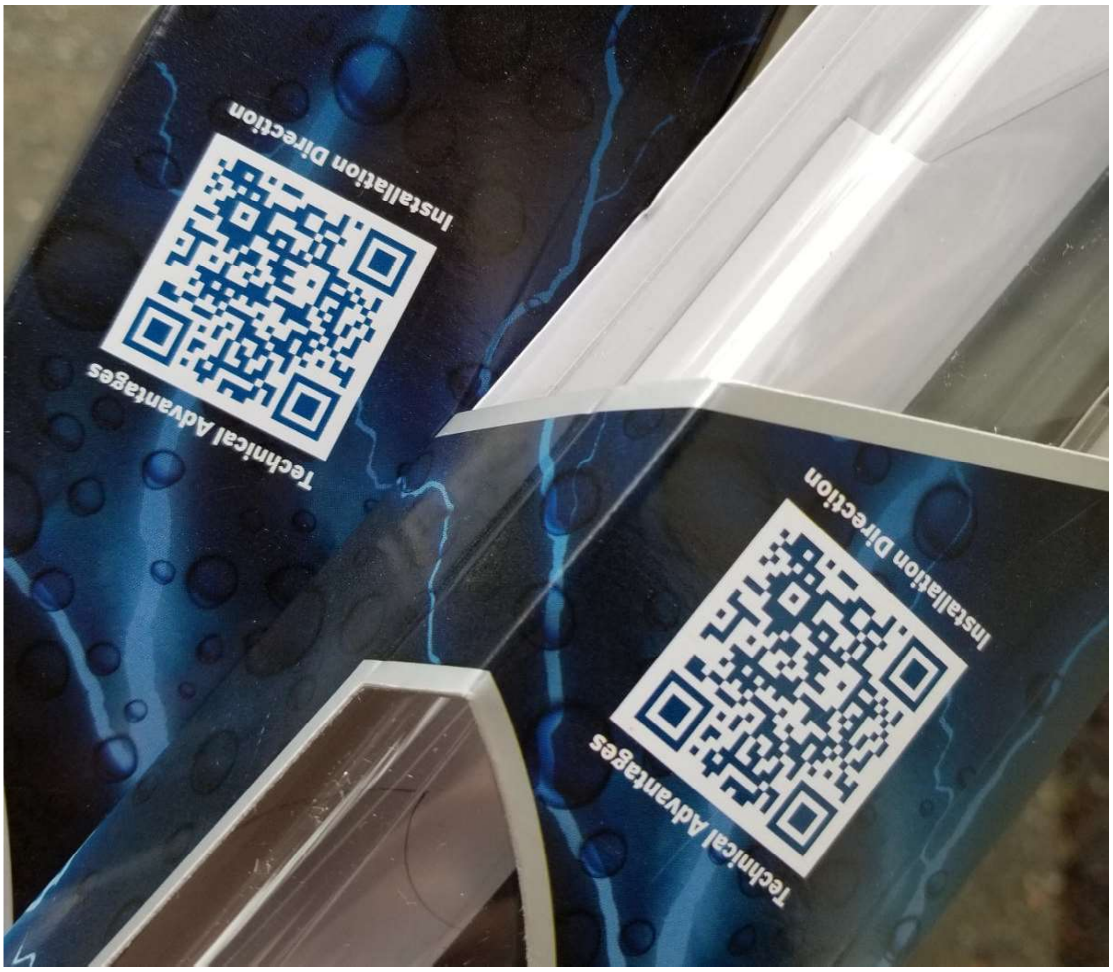


Figure 5.1: Control group image.

that the position detection patterns are binarized incorrectly. The code in the original image is about 800 pixels wide, making the 3×3 module wide center of one position detection pattern larger than 100 pixels, the threshold for each pixel is determined for a square neighborhood around each pixel of 57×57 pixels however, meaning only dark areas of the code are evaluated. As a result of this the threshold of pixels at the center of the position locator patterns is close to the value of the pixel itself, resulting in noise being enough to elevate individual pixels above the threshold. A solution to this problem is the selection of a threshold mean neighborhood which at least the width of three QR code modules. The one failure of the grid mean thresholding with a constant amount of grid elements also occurred in a large image. The constant amount of gird elements, in this case 50, results in more pixels being part of one grid square and the distance of the borders between grid elements being further from the calculated value at the center. This lowers the accuracy of the process and causes noise. This noise not only caused the position detection pattern to be obscured, but also allowed for the detection of three finder pattern sequences in the noise of the background, which seems to be a result of the larger image size, as the amount finder pattern candidates is significantly higher

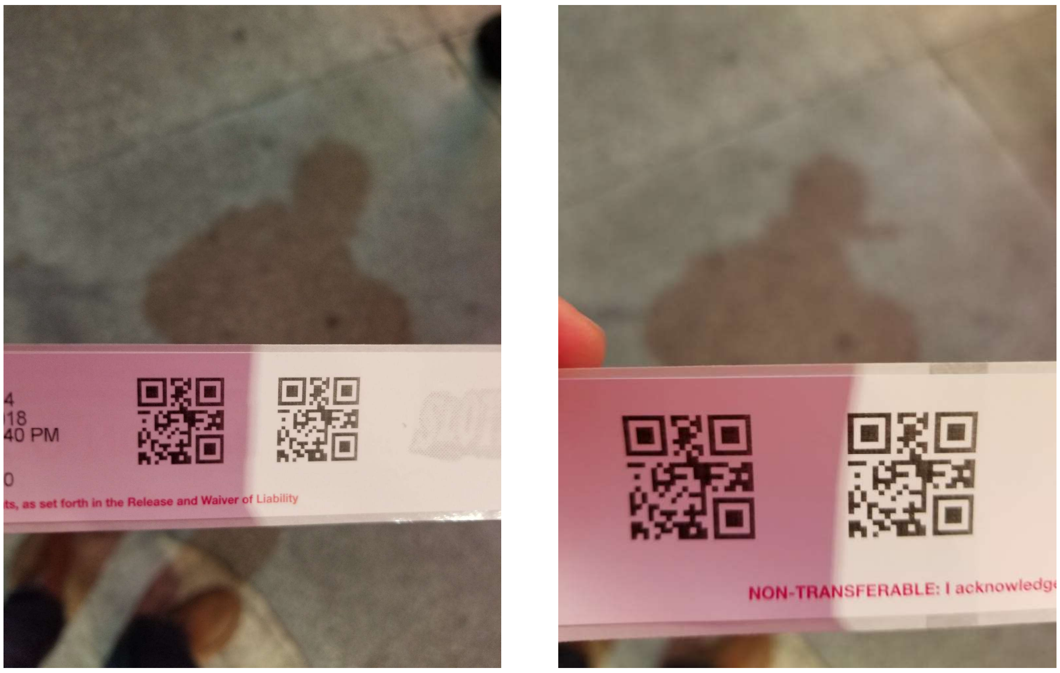

4
18
40 PM
0
ts, as set forth in the Release and Waiver of Liability
NON-TRANSFERABLE: I acknowledg

Figure 5.2: Example of the tested images for different sizes and lighting conditions.

in the large images. Generally these false positives are removed for not matching the estimated module size of other position locator patterns. In that particular case 5.3 (c) the patterns in the noise matched in module size causing the detection to stop as three patterns were found. This could be corrected by comparing the position of the finder patter centers to each other. The position detection patterns create a triangle between them, which can not have an angle bigger than 90°. This could be tested for, allowing for slight variation from perspective distortion or curved surfaces, to exclude mismatched patterns from the detection. The failures of the global thresholding are unrelated to the image size, but the result of the selected threshold not matching the entire image evenly, because of the inconsistent lighting, causing destructive thresholding as can be seen in figure 5.3 (a). As such unevenly lit images are the target of the next test.

5.3 Lighting

To avoid the size of the image and QR code to influence this test the selected images were scaled and cropped to ensure equal conditions for all test images. The testing is done for two groups of images one with the overall brightness of the image varying and the second one with inconsistent lighting across the image in the form of bright spots or shadows. The techniques tested are global- and local mean thresholding, global Otsu thresholding, which may improve upon regular global mean thresholding, as the determining of the threshold is significantly more advanced, and lastly local Gaussian thresholding, which is used to add additional context to the local mean threshold. The first test uses 15 images containing three QR codes, which were manipulated so only

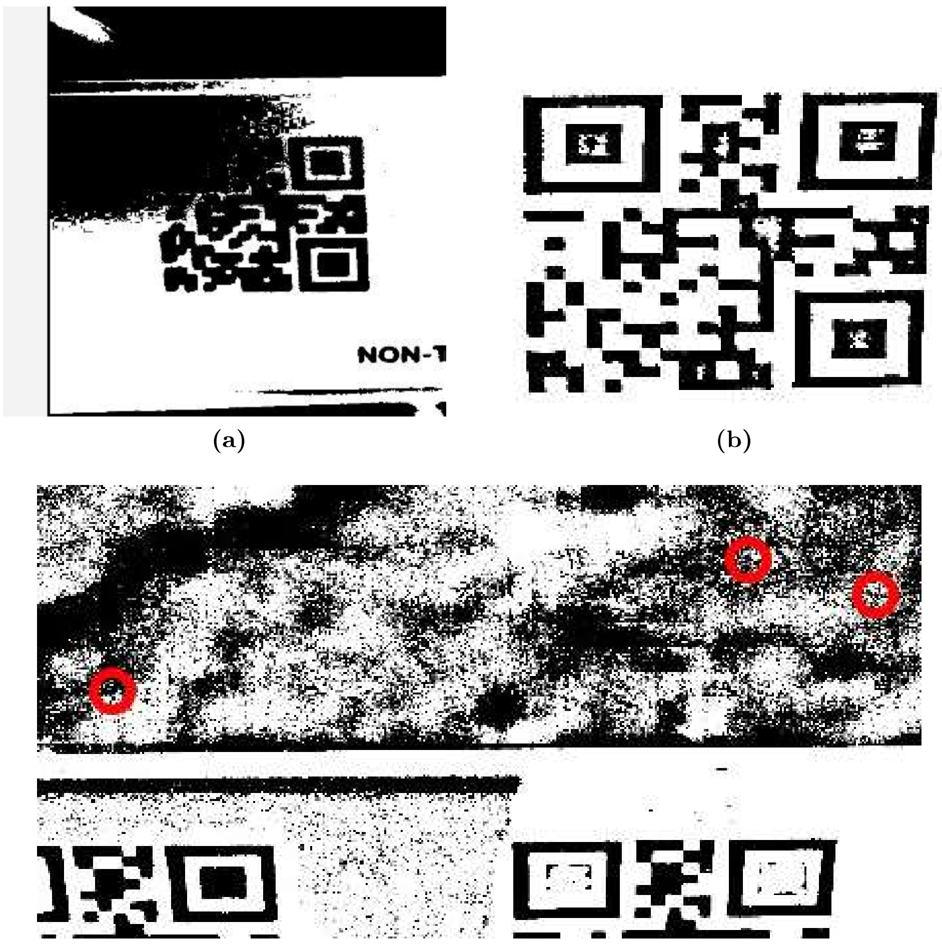


Figure 5.3: Examples of the errors occurring in the binarization process. (a) shows the result of global mean thresholding on images with inconsistent lighting. (b) is the result of the diameter selected for adaptive mean thresholding being smaller than the center of the position detection pattern, resulting in incorrect binarization. (c) shows an example of how noise can cause incorrect detection and prevent the search for the correct finder pattern.

one is detectable, at different overall brightness levels. The goal is to find the position detection patterns in the code, which generally correlates to a decodable QR code. Table 5.3 shows the number of finder patterns that could be found for each binarization method. The results of this test do not contain any surprises. The lightest image 5.4 failed for all methods which can be attributed to the overblown light areas disrupting the ratio between light and dark modules in the position detection patterns. The other cause of failure were the darker images, which suffered from a similar issue as encountered with local thresholding in the previous test. The range of values present in the image is exceptionally low, meaning the threshold value is close to both light and dark areas, resulting in miss-determination for slight deviations in brightness, resulting in noise in the final image. As before noise can interfere with the detection of the locator patterns

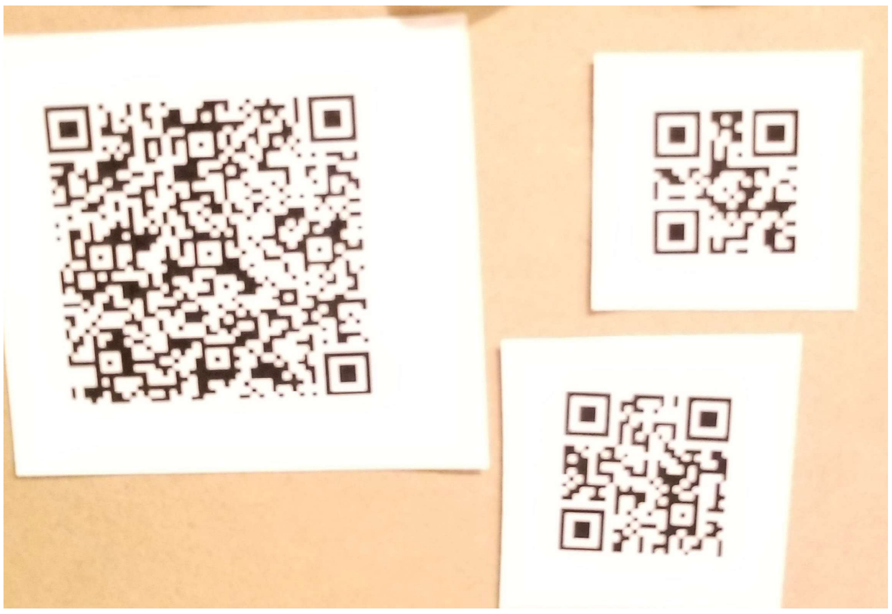
	Global Mean	Global Otsu	Local Mean	Local Gaussian
All Found	9	8	9	9
2 found	1	2	3	0
1 found	1	1	0	0
failure	4	4	3	6

Table 5.3: Results of attempting QR code detection on images of varying brightness.

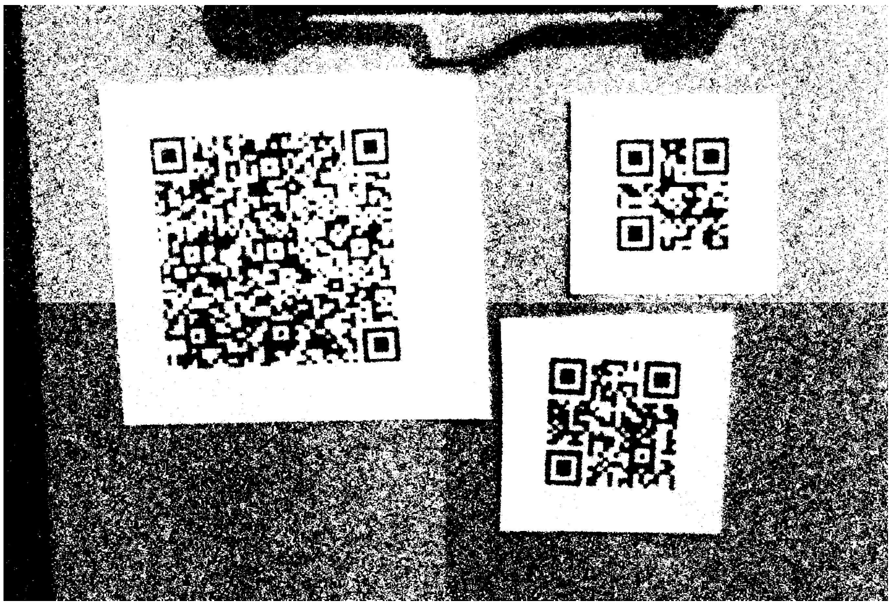
		G. Mean	G. Otsu	L. Mean	L. Gaussian
Bright spots	All Found	3	0	5	8
	2 found	3	0	2	0
	1 found	3	0	1	0
	failure	0	9	1	1
Shadows	All Found	4	5	7	7
	2 found	1	0	1	1
	1 found	3	3	0	0
	failure	0	0	0	0

Table 5.4: Results of QR detection on images with inconsistent lighting in the form of shadows and bright spots. Comparing global (G.) and local (L.) thresholding methods.

as the sequence may be broken due to it, as well as causing false positives outside of the code. How close the image values are can be seen in 5.4 (b) which uses values 4, 5, 6 and 7 to generate the binarization for each of the quadrants. As expected constant poor lighting conditions affect all methods about equally, variations in brightness across the image however do not. The next test attempts pattern detection with the same 4 binarization methods on 17 images which either contain a significantly brighter area than the rest or a shadows being cast across all or parts of QR codes. In order to more easily distinguish between the results, they are separated in the table. 5.4. This clearly shows how Otsu's method is not suited to calculate a global threshold, for images containing a significantly brighter area. The goal of Otsu's method is the calculation of a threshold that separates the largest values which contribute to the image. The extremely bright area of the tested images results in a value that separates the bright area from the rest of the image, causing the QR code to be lost in the background. Figure 5.5 visualizes this dynamic using one of the example images, displaying the binarized image using Otsu's method on the left and the global mean threshold on the right. Through both methods, only one position detection pattern could be found, however. Otsu's method is significantly better suited for images with a lesser gap between the brightest and darkest areas, such as the shadow test images, on which it performs above slightly better than global mean thresholding. For those images neither can compete with locally adaptive methods of thresholding however, both tested methods only fail to detect the position detection pattern which is disrupted by glare and therefore impossible to detect for most systems. Comparing the detection results of local and global thresholding methods can be used to determine whether an image contains a shadow or other source of brightness change, as the local thresholding will generally be more successful in these scenarios.



(a)



(b)

Figure 5.4: Examples from the testing of particularly light or dark images. Image (a) shows the brightest image in the tested set. The position detection patterns can not be found as the overabundant light disrupts the ratio of the pattern. (b) shows the result of applying global thresholding to the darkest image used. The four quadrants of the image were binarized with different values in order left to right: 4, 5, 6, 7, in order to visualize the small range of brightness across the image.

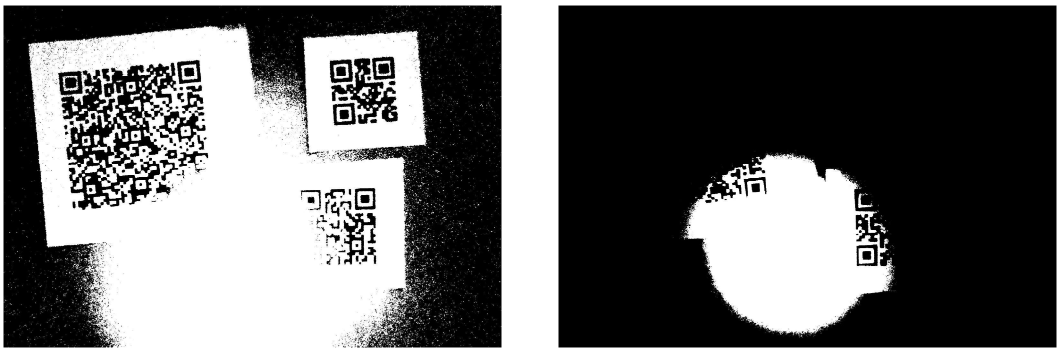
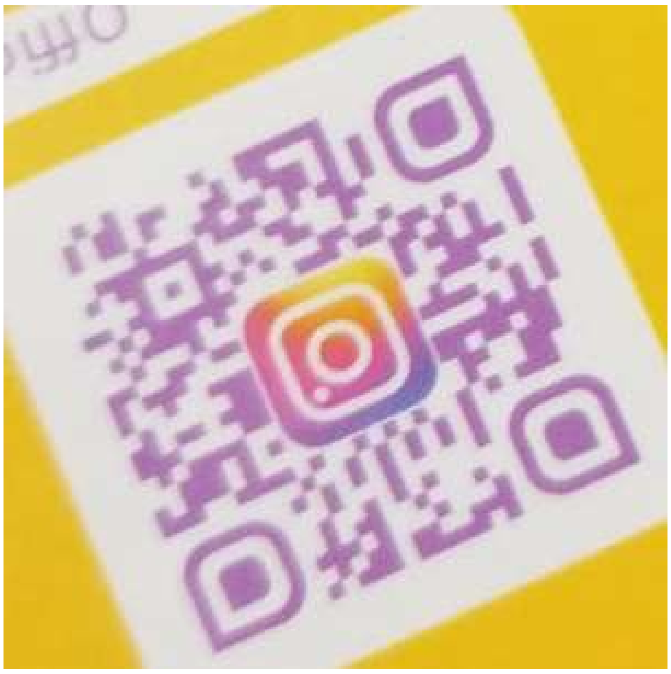

(a) (b)

Figure 5.5: Binarization of image containing a bright spot. (a) showing the result of global mean thresholding and (b) the result of using Otsu's method of creating the threshold value.

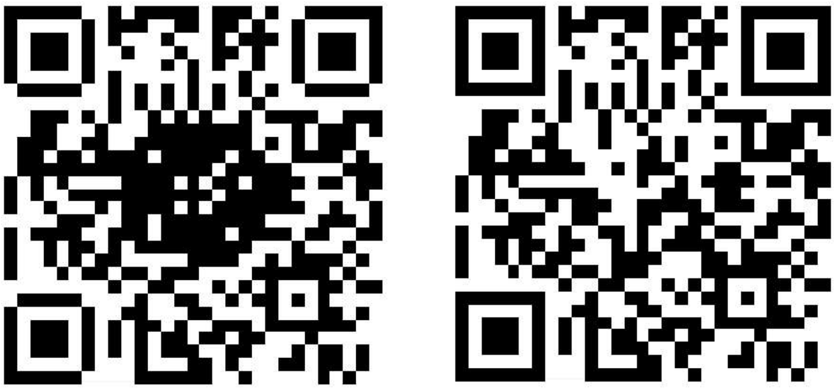
This can be supported by comparing the positions of the found patterns to ensure no false positives.

5.4 Design

QR codes are used a lot in promotional material and as such are often presented with parts or the whole in color. Additionally, certain liberty is taken with the shape and content of the code. In the following some examples of QR codes, deliberately breaking the standard are tested. The first image serves as an example of how codes might be broken. It combines multiple levels of non-compliance with the standard. Figure 5.6 shows the potential of using the error correction capabilities to embed symbols in the code. The shape of the position detection patterns may fail for certain implementations, as the pattern is not consistent over the diagonal. Further, the color choice of the "dark" modules is significantly lighter than recommended. The successful decoding of this code is only possible because the code uses the highest error correction level H, allowing for a quarter of the encoding area to be obstructed. Applying the error correction, encoding, and remasking to the resulting data the originally intended code can be restored. Should the code contain more errors than the intentionally placed ones, it is quite feasible that the error correction may not be successful. During the next testing of colored QR codes this same code was tested and the design in the center was detected as a position detection pattern multiple times, which may inhibit the detection process of poorly implemented code detection systems. The discussion of the previous image eluded to the color of the dark modules, potentially causing problems. The following test compares three grayscaling methods to find their influence on the contrast between light and dark modules, specifically for codes with colored modules. The tested images contain 9 images of either colored QR codes, colored parts of QR codes or black codes on colored backgrounds. The intention of the test is the comparison of grayscaling methods: *Intensity*, *Luma*, using a γ of $\frac{1}{2.2}$, and *Luma* using a $\gamma = 2$. Gamma correction is necessary because video capture devices, as well as displays, do not correctly display



(a)



(b)

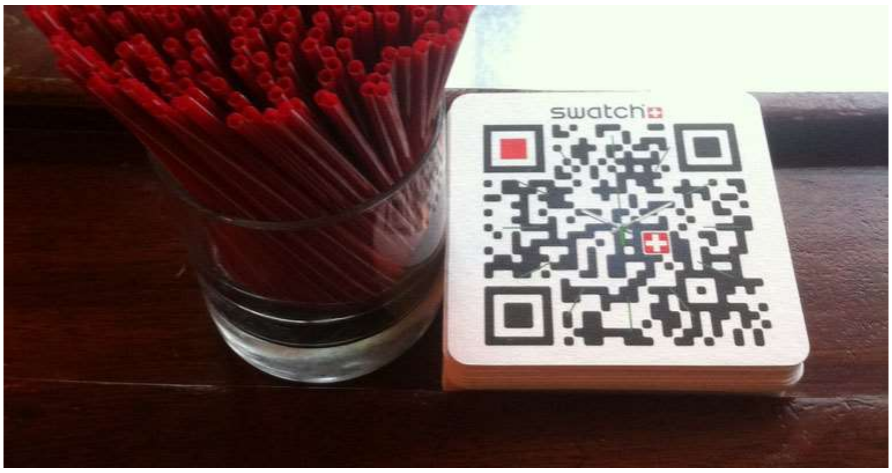
Figure 5.6: Example of non compliant QR code. (a) is the original image which was scanned. (b) shows side by side the binary matrix which was extracted from the image on the right, and the error corrected and remasked version on the left.

luminance. The values selected here roughly correspond to what would be used for gamma correction for a CRT Monitor, as well as its inverse used to revert the process. Lastly *Value* will be tested, as its achromatic nature may influence the detection of color codes differently from other methods. The test results are also judged on the amount of position detection pattern found, therefore require binarization to take place. For this local mean thresholding is used as it proved to be the most consistent over the previous test. The testing revealed that certain color values result in significantly lower contrast

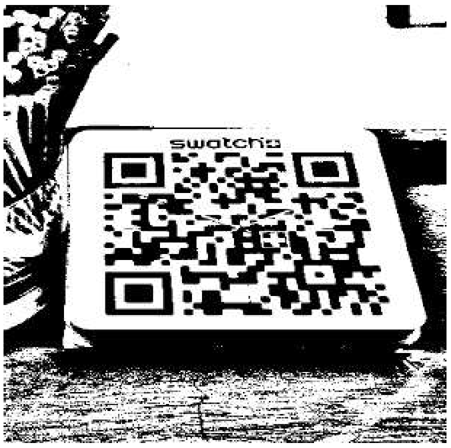
	Intensity	Luma (12.2)	Luma (2)	Value
All Found	5	4	6	3
2 found	0	1	1	1
1 found	3	1	1	0
Failure	1	3	1	5

Table 5.5: Results of testing images of varying brightness using different grayscaling methods and different values for gamma.

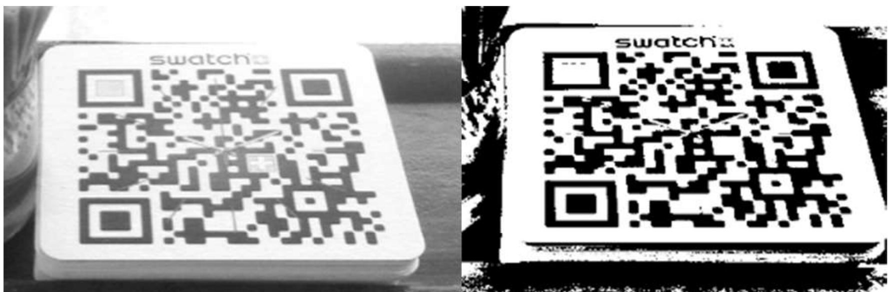
to the surrounding area than required for flawless detection. This is well visualized in figure 5.7 showing the results of using different grayscaling methods on a QR code with a colored position detection pattern. Using *Intensity* grayscaling the binarization can function properly, using *Value* however a very light gray square remains of the red pattern causing the binarization to be unsuccessful. The relevance of correct gamma correction is another aspect of this test, which yielded interesting results. Figure 5.8 shows that the clear improvement of the binary image resulting from it. Application of any gamma correction changes the linear brightness function to a quadratic one, changing the range in which certain brightness values lie. Figure 5.9 demonstrates how this can cause the binarization to fail if the code modules are close to the background in brightness.



(a)

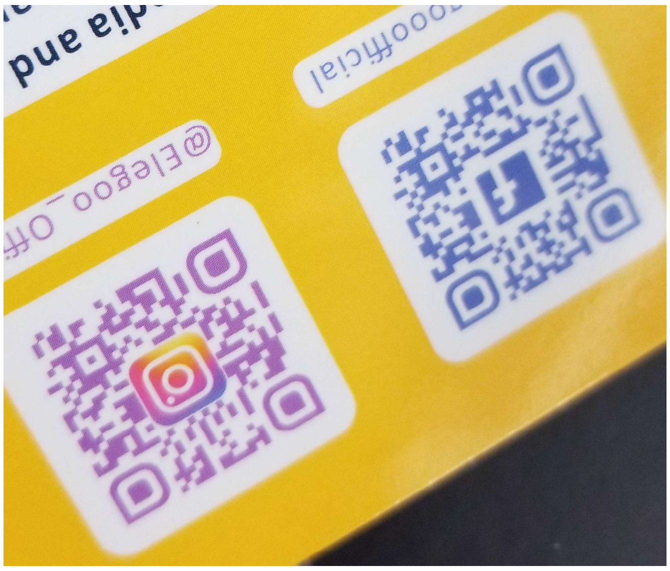


(b)

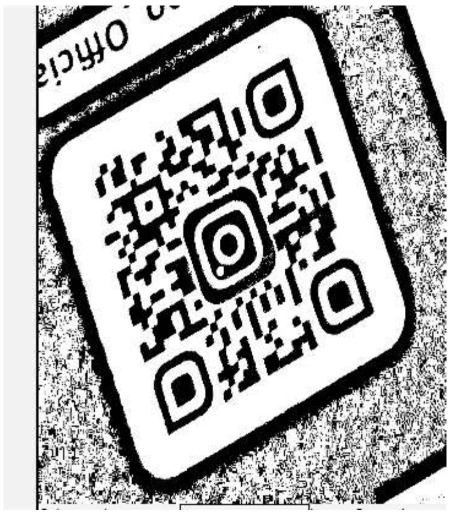


(c)

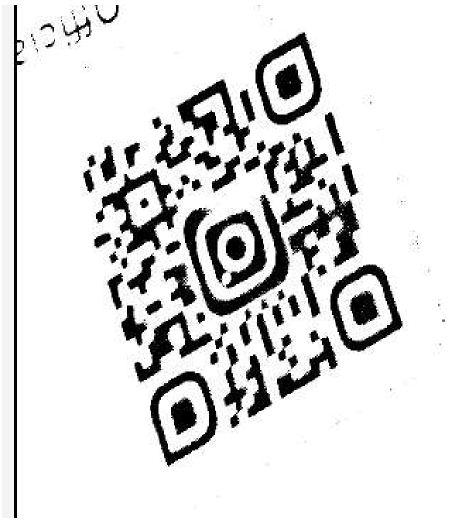
Figure 5.7: Example of the grayscaling testing done. (a) shows the original image. (b) contains the binarized image after *Intensity* grayscaling which succeeded. (c) shows side by side the result of *Value* grayscaling and the resulting binary image.



(a)

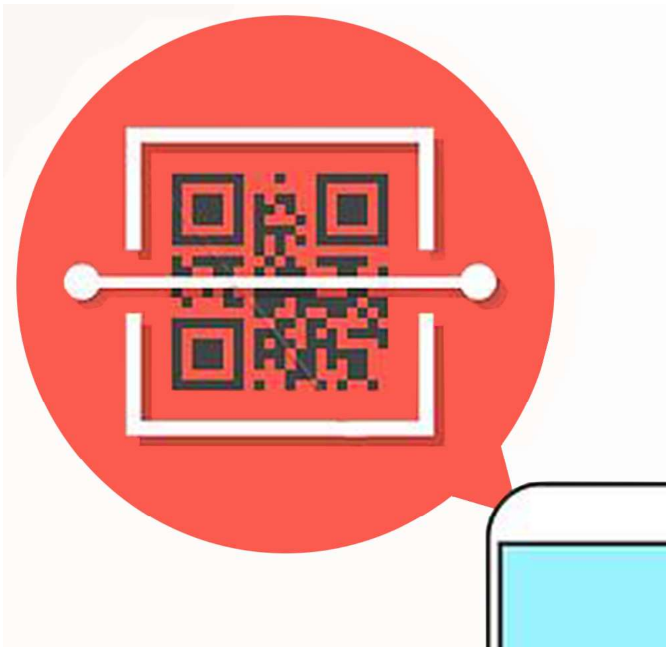


(b)

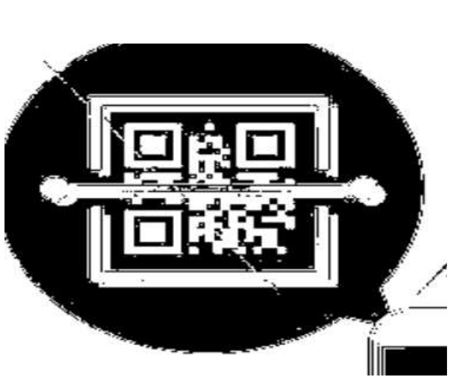


(c)

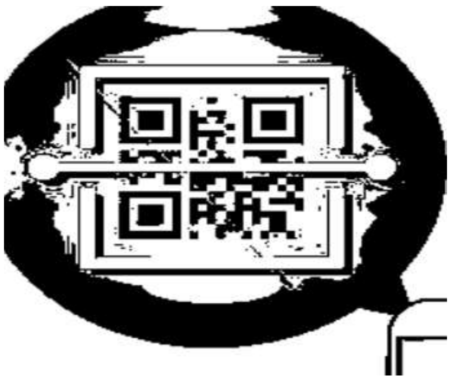
Figure 5.8: Example of the testing with different values for the gamma correction. (a) being the original image, (b) using $\gamma = 2$ and (c) $\gamma = \frac{1}{2.2}$. It clearly shows how the contrast suffers from incorrect gamma correction.



(a)



(b)



(c)

Figure 5.9: Example of a gamma correction catastrophe. The first figure (a) shows the original image. By applying different values for gamma (b) $\gamma = 2$ and (c) $\gamma = \frac{1}{2.2}$ the grayscaling process disrupted the content so much, that no successful binarization was possible for figure (b) anymore.

Chapter 6

Conclusion

The goal of this project was the determination of how image quality affects the detection process of QR codes and general visual AR markers. The initial goal of creating a program to “judge” image quality immediately proved problematic to solve as any rating process arrived at with classical means would be arbitrary at best and flawed at worst, as the number of potential parameters and their influence on each other is not possible to evaluate. The structure of the data does open the door for another solution though. Using QR codes as an example, a detection system could be used to annotate the images in order to train a neural network on the aspects, which make QR codes undetectable. Particularly the comparisons between different binarization techniques and grayscaling techniques may provide enough data to objectively grade images on their contrast and composition via a neural network to quickly determine detection chances. It appears reasonable to assume that such a grading system could then be applied to other AR systems that share features with QR codes, and provide correct or at least usable data for these systems as well. Particularly the comparison with AR Toolkit and other square binary marker systems seems possible. Additional insights gathered from the work on this project address flaws in current detection methods. The given goal of the project being about gathering data, a system was needed which continues computations on the aspects of the code, even if the detection process already failed at some point, e.g. if fewer than three position detection patterns were found. The QR detection systems used to learn how the process works could only guide the development so far, as the result they provide exists in a binary state, of the code being found or not found. Understanding and analyzing the data which caused the failure, allowed for the postulating of theories on how the process may be improved to allow for a higher rate detection in images, which currently have zero chance of yielding pattern position. One of the theories was the result of researching the iQR code system, which improves upon QR code by increasing the variety of shapes and sizes they can be displayed as, while also removing two of the position detection pattern. The exact detection manner is not part of the public domain at this point, it can, however, be assumed that the combination of the one existing positional pattern and the black-white iterating border on two sides of the code, aids the location process of the code. This structure is similar to that of regular QR codes, in how the position detection patterns are connected by the black and white alternating timing pattern. It appears feasible to use the timing patterns in connection with one or two position detection patterns in order to find or estimate missing positional patterns.

This is obviously just a theory which would require time not allotted for this project to test. Unrelated to the possibility of this being functional, it might not be relevant to the way QR codes are used. This sentiment of not being relevant to the common usage of QR codes extends to this entire project. The common use cases for QR codes do not include the detection in a single image but in a stream of images retrieved from the camera. In this way, only a single successful detection is necessary to succeed. As a result, the method by which the testing was done might be flawed from the beginning. Determining the quality of the images received from the camera of a phone would require a different setup to test and would likely have different requirements in its testing, i.e. results being available in close to real-time, preventing some approaches tested in this project. However with the goal of creating a tool for developers and not end-users, this criticism falls flat, as the goal for the developer will not be the quick, but accurate evaluation of scenarios. Additionally, an individual photo may reveal problems with the image acquisition that an interpolated image stream may never be subject to. The wide field of both phone hardware and Marker systems, with their respective detection processes, results in a wide range of possible points of failure, meaning that problems with outdated hardware or software may not be immediately apparent. A tool capable of anticipating problems with poorly implemented marker detection software would definitely provide some value to developers and allow for the creation of a more consistent application, and as such a better user experience. Personally, the project required me to learn new information in fields I did not expect at the start. In particular, the math involved at every step of the detection and decoding processes required a lot of work, with the error correction calculations especially, as the mathematical concept of fields, was not a topic in any project before. The perspective transformation proved to be an interesting use case of something which was learned for a different context in the past. On the topic of the project structure, the goal being set rather loosely, proved to be a blessing and a curse, as the finding of specific aspects to prove or disprove was virtually impossible. This led to no solution for this project and problem ever being complete as some new combination of parameters is always possible and reasonable to test, leaving a feeling of something always missing from the project. On the positive side, this allowed for a lot more testing methods to be possible, as few things were beyond the scope of the project in that area.

Appendix A

CD-ROM/DVD Contents

The accompanying DVD contains this paper in both PDF form as well as the full LaTeXsource code. Additionally the source code of both programs created for the purpose of testing is included. This serves only for completion sake and not to be used for additional testing as neither usability nor the creation of executable software was the goal of the project. Lastly the images used for the evaluation in chapter 5 are included on the DVD.

References

Literature

[1] Clemens Arth et al. *The History of Mobile Augmented Reality.* Tech. rep. Institute for Computer Graphics and Vision Graz University of Technology, Austria, Nov. 2015 (cit. on p. 5).

[2] Luiz Belussi and Nina Hirata. “Fast QR Code Detection in Arbitrarily Acquired Images”. In: *Proceedings of the 24th SIBGRAPI Conference on Graphics, Patterns and Images* (Algoas, Maceia, Brazil). São Paulo, Brazil: IEEE Computer Society, Aug. 2011, pp. 281–288 (cit. on pp. 9, 26).

[3] Oliver Bimber and Ramesh Raskar. *Spatial Augmented Reality. Merging Real and Virtual Worlds.* A. K. Peters/CRC Press, Aug. 2005 (cit. on pp. 1, 2).

[4] Wilhelm Burger and Mark J. Burge. *Digital Image Processing. An Algorithmic Introduction Using Java.* 2nd ed. Springer Publishing Company, Incorporated, 2016 (cit. on p. 31).

[5] Can Eyupoglu. “Implementation of Bernsen’s locally adaptive binarization method for gray scale images”. *The Online Journal of Science and Technology* 7 (2017), pp. 68–72 (cit. on p. 25).

[6] Mark Grundland and Neil A. Dodgson. *The decolorize algorithm for contrast enhancing, color to grayscale conversion.* Tech. rep. UCAM-CL-TR-649. University of Cambridge, Computer Laboratory, Oct. 2005. URL: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-649.pdf (cit. on p. 21).

[7] Bernsen J. “Dynamic thresholding of grey-level images”. In: *Proceedings of the International Conference on Pattern Recognitions(ICPR)* (Paris). IEEE Computer Society, Oct. 1986, pp.1251–1255 (cit. on p. 25).

[8] Christopher Kanan and Garrison W. Cottrell. “Color-to-Grayscale: Does the Method Matter in Image Recognition?” *PLoS ONE* 7.1 (2012), pp. 1–7 (cit. on p. 19).

[9] H. Kato and M. Billinghurst. “Marker tracking and HMD calibration for a video-based augmented reality conferencing system”. In: *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR’99).* IEEE Computer Society, 1999, pp. 85–94 (cit. on p. 5).

[10] Nobuyuki Otsu. “A Threshold Selection Method from Gray-Level Histograms”. *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66 (cit. on pp. 23, 24, 32).

[11] Simon Perreault and Patrick Hebert. “Median Filtering in Constant Time”. *IEEE Transactions on Image Processing* 16.9 (Sept. 2007), pp. 2389–2394 (cit. on p. 32).

[12] Ivan E. Sutherland. “A head-mounted three dimensional display”. In: *Proceedings of the AFIPS ’68 Fall Joint Computer Conference.* ACM Press, Dec. 1968, pp. 757–764 (cit. on p. 4).

[13] István Szentandrási, Adam Herout, and Markéta Dubská. “Fast Detection and Recognition of QR Codes in High-resolution Images”. In: *Proceedings of the 28th Spring Conference on Computer Graphics.* SCCG ’12. Budmerice, Slovakia: ACM, 2012, pp. 129–136 (cit. on p. 26).

Online sources

[14] Peter Abeles. *Study of QR Code Scanning Performance in Different Environments. V3.* 2019. URL: https://boofcv.org/index.php?title=Performance:QrCode (cit. on p. 37).

[15] Denso-wave. *Bar code symbology - QR Code.* URL: https://www.swisseduc.ch/informatik/theoretische_informatik/qr_codes/docs/qr_standard.pdf (cit. on pp. 7–10, 12, 13, 26).

Check Final Print Size

— Check final print size! —

width = 100mm
height = 50mm

— Remove this page after printing! —