Robust Patient Pseudonym Extraction during Automatic Calibration in Low-Cost Photogrammetry

Student Research Project Paper

Benedikt Christoph Wolters

RWTH Aachen University, Germany
Department of Medical Informatics

Advisors:

Prof. Dr. rer. nat. Dipl.-Ing.	Thomas Deserno (geb. Lehmann)
Dipl.-Inform.	Daniel Haak
M.Sc.	Ekaterina Sirazitdinova

Submission Date:
2016-04-04

Abstract

Photographic documentation is frequently performed in longitudinal clinical trials e.g., to support quantitative wound assessment. Conventional methods typically require specialized expensive calibrated photographic equipment. The emergence of inexpensive high-resolution general-purpose cameras in modern smart phones or tablets has paved the way for novel, more economic, and integrated ways to perform photographic documentation. However, using low-cost hardware, the calibration of the photographs now becomes a post-processing step as it is an essential task to achieve comparability. Additionally, using Internet-connected consumer equipment such as smart phones enables the entire documentation process to be embedded onto the consumer device e.g., through a mobile application. Amongst others, this process involves a robust patient (re-)identification to prevent documentation errors. In this paper we will combine the required image calibration step with a mean to extract the subject's pseudonym using a special subject-tailored Macbeth color reference card. Initially, the primary goal of the card is to serve as a reference for color- and perspective-corrections of the image. However, equipped with both a textual and a bar-code representation of the subject's pseudonym, the card also allows us to apply and combine different approaches of image recognition to extract the subject's pseudonym. In this paper we will introduce different algorithms to robustly localize the Macbeth color reference card in a medical image. Subsequently, we discuss and evaluate different approaches of fine-tuning and combining different stages of the algorithms on 208 images. Finally, we conclude the paper with an outlook and a comparison of our methods.

Contents

1 Introduction

1.1 Photographic Documentation in Clinical Trials using Smart Phones

Longitudinal quantitative monitoring of the development of visible outspread of a disease, especially a wound or a skin lesion is of essential importance in clinical trials. For this, photographic documentation plays an important role in clinical trials especially for objective wound evaluation. However, lack of standardization in protocol has hindered the wide adoption of a unified documentation process [1]. Photographic documentation suffers from perspective distortions due to the use of optical lenses as well as color shifts due to different illumination and ambiance [2]. Hence current photographic documentation techniques require high-quality, expensive and special calibrated hardware [3]. Additionally, calibrating the equipment might be a tedious task that requires special knowledge. In [1, 4, 5] a imaging processing framework based on consumer-grade smart phone equipment has been proposed. Instead of performing the picture calibration before shooting a picture, now the picture calibration becomes a post-processing step by using a color reference card. This card both standardized in size measurements as well as colors can be placed next to the region of interest to be documented in the image. Using the color card as a reference the image then can be corrected. To increase standardization amongst the images taken in possibly multi-centered locations by different study nurses a written standard operation procedure (SOP) is handed to the study nurses, who capture the images. The SOA describes the parameters and basic conditions of the photograph, positioning of the reference card and region of interest as well as parameters of the smart phone camera [1].

1.2 Combining Color Cards with Subject-Pseudonyms

Using only smart phone equipment now the whole documentation process can be integrated automatically with electronic data capture (EDC) systems [5]. For this the subject needs to be associated with the image taken.

While the subject's pseudonym could also be entered manually in the smart phone app before or after taking the picture there are several reasons to embed the pseudonym within the image itself and combine this two steps:

- The smart device might already contain multiple very similar images of different test subjects, which makes it hard for the operator to distinguish different subjects from each other, especially on small screen sizes, similiar subject's characteristics or due to the sheer amount of images.

- The operator might mistype or confuse subject's pseudonyms, when entering the subject's data manually at worst resulting in a wrong association, or corrupt study data.
- The operator might also aswell enter the subject manually, having the pseudonym in the image itself can be used for data validation and will avoid association errors.
- In case of a partial meta data loss (e.g. a corrupt or missing subject-to-image database association) the pseudonym is still extractable from the raw images.
- The successful pseudonym extraction can be used as a mean of quality assurance: If the pseudonym embedded into the image turns out to be extractable, chances that the image is not unsharp, jittery, blurred or otherwise unusable in latter analysis are increased. On the other hand images of poor quality can be automatically declined.

Therefore, in this paper we extend and improve the architecture presented in [1, 3] by introducing special individual subject-tailored color reference cards containing the study subject's pseudonym. In this paper we specifically aim at (i) locate the color reference card automatically within the image, (ii) extract the card from the image, and (iii) read the subject's pseudonym from the special color reference card.

1.3 Related Work

The use of color cards to perform geometry and color correction was originally presented in [2]. However, the initial approach requires manual selection of the color card corners which are then refined using template matching.

A major part of our approach concentrates on the automatic localization of a special color reference card used for image calibration. Several approaches have been introduced in the past for the localization of color cards: [6] uses adaptive thresholding against the RGB color channel in combination with contour finding with heuristics to filter the color squares, Bianco et al. in [7] introduced SIFT feature matching, followed by clustering the matched features to be fed into a pose selection and an appearance validation algorithm. In [1] Deserno et al. leverage SIFT feature matching and Macbeth color cards in medical images to perform perspective transforms in combination with quantitative assessment of medical images. Brunner et al. in [8] propose a scan-line based method to fit a known color reference chart. The authors of [9] describe the extraction of polygonal image regions and apply a cost function to check their adaption to a color card. [10] uses color quantization and binarization combined with a refinement step to obtain the color card values. Finally, in [11] Wang et al. introduce a per-channel feature extraction with a sliding rectangle in a rough detection step combined by a more precise fine-adjustment step.

While we acknowledged the existence of a myriad of various approaches that cover color card detection, to the best of our knowledge none of them have dealt with extracting additional information from the color card apart from the sole purpose of image and color calibration. In our context not only a color value of the color patch is of interest, but also a very precise localization of the color card to extract the regions of the color card that contain the subject's pseudonym. Thus a precise localization of the card's geometry is essential for the correct extraction of the subject's pseudonym.

Hence and due to the fact that our scenario is similarly targeted to clinical trials, we base our work on Jose el al. [3] which utilizes an algorithm by Park et al. [12] to detect deformed lattices in real world images, which has proven to outperform the SIFT approach in [1].

To summarize the goal of this paper is to extend and optimize the work in [3] both with respect to runtime, robustness, and accuracy. Also we adapt this approach to a new specialized color reference card containing two representations of the study subject's identifier.

1.4 Outline

The rest of the paper is structured as follows: Section 2 will introduce preliminaries and building blocks of of our architecture. Section 3 describes our card-detection algorithm as well as various algorithm variants. Subsequently, in Section 4 we evaluate our algorithm, followed by an outlook in Section 5.Finally, the paper is concluded in Section 6.

2 Background

In the following section we will describe background information and preliminaries needed for the comprehension our card extraction algorithm.

2.1 Macbeth ColorChecker Cards

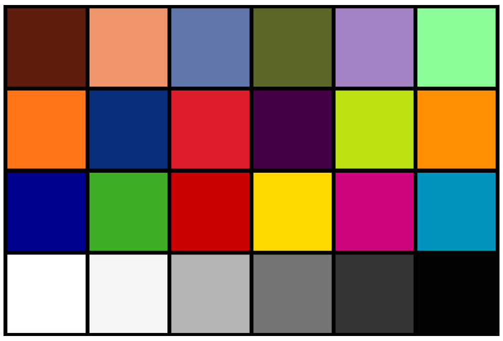


Figure 1: The original Macbeth ColorChecker card

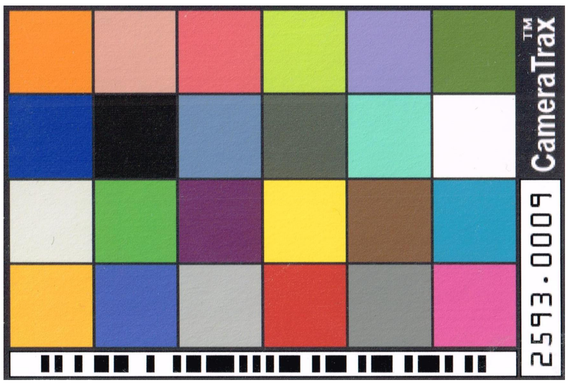


Figure 2: The color card used in this study. It is comprised of a 6x4 color patch grid, a bar code and a textual representation of the study subject's pseudonym

Property	Value	
Print Resolution	150 ppi	(0.17mm/pix)
Height	300 pix	(51 mm)
Width	447 pix	(76 mm)
Color Patch Height/Width	64 pix	(10.84 mm)
Barcode Width	382 pix	(64.69 mm)
Barcode Height	18 pix	(3.05 mm)
Text Width	148 pix	(25.06 mm)
Text Height	30 pix	(5.08 mm)

Table 1: Custom color card specifications

No.	Name	Red	Green	Blue	No.	Name	Red	Green	Blue
01	Orange	223	121	053	13	Light-Grey	202	203	202
02	Light-Tone	193	145	130	14	Green	075	153	077
03	Light-Red	195	082	097	15	Purple	093	061	104
04	Yellow-Green	162	189	064	16	Yellow	241	201	042
05	Light-Blue	129	128	175	17	Dark-Tone	116	084	073
06	Tree-Green	093	107	069	18	Cyan	000	135	166
07	Blue	066	067	152	19	Orange-Yellow	229	158	048
08	Black	051	052	053	20	Median-Blue	071	090	173
09	Sky-Blue	091	121	156	21	Grey	161	162	163
10	Charcoal	084	085	085	22	Red	172	059	062
11	Blue-Green	097	189	170	23	Dark-Grey	119	120	121
12	White	247	244	243	24	Magenta	190	087	151

Table 2: sRGB color values for custom color card (top left to right)

In our scenario a customized version of the Macbeth ColorChecker Card [13] is being utilized. The Macbeth ColorChecker Card (depicted in Figure 1) is an arrangement of 24 squares (color patches) in a 4x6 grid filled with sample colors. The color patches of the card have been chosen to have spectral reflectance of as natural objects such as human skin. Furthermore, the matte painting of the card has been chosen to have consistent appearance under a variety of lighting conditions and to keep the same color appearance over time (stableness).

The colors chosen for the card contain six uniform gray color patches, the six primary colors typically encountered in chemical photographic processes (red, green, blue, cyan, magenta, and yellow) as well as a collection of approximations of medium light and medium dark human skin, blue sky, the front of a leaf and a blue chicory flower. Furthermore the card includes a yellow and an orange similar to the colors of oranges and lemons. The remainder of colors have been chosen arbitrarily to represent a good mix of colors [13].

Photographs containing the Macbeth ColorChecker Card can be compared to the original card and the card's specifications and thus allow harmonization and finally the comparison of different images taken with different photographic equipment or in different settings.

In our scenario, we have extended and adapted the Macbeth ColorChecker Card in order to (i) incorporate two representations of the subject's identifier, and (ii) rearranged the color patches to increase the color difference between two pairwise-different neighboring color tiles. The patient's identifier is an eight digit numeric number. The patient's identifier is embedded twice in different encodings on the color reference card. On the one hand this is done to ensure that both humans as well as machines have a preferable representation: The bar code is printed in normal mono space machine letters as well as in a bar code version. Details of the bar code are described in the next subsection. On the other hand this redundancy grants additional robustness in the event that one of the two representations is (partly) occluded or otherwise illegible e.g., by a finger holding the color reference card. The adopted Macbeth ColorChecker card is depicted in Figure 2. The adapted card specifications are summarized in Table 1, while the used colors are listed in Table 2.

2.2 Bar Codes and EAN-8

In order to obtain a machine-readable representation of the study subject's identifier we deployed a bar code to the color card. In general a bar code is a machine-readable representation of data relating to the object to which it is attached. Bar codes are widely used in the commercial domain at checkout counters as well as transportation and have been adopted in the medical field for example in the domain of medication safety [14,15]. There exists are multitude of different bar code types: There are one dimensional bar codes, where the data to be recognized is depicted by an alphabet consisting of varying widths of spacings of parallel lines. Furthermore, there are two-dimensional encodings that use rectangles, dots, hexagons and other geometric shapes to encode the data. While bar codes originally were developed to be scanned by special hardware (known as bar code readers), software applications are available that allow the processing of bar codes within devices that simply take an image of the bar code such as smart phones.

We evaluated different one dimensional bar codes for the use within our setup under the aspects of (i) robustness against errors, (ii) required size/space, (iii) available open-source bar code reader software. We found the one dimensional bar code EAN-8 to be fitting due to its wide usage in commercial domain as well as adoption in software scanners and it support for checksums. The EAN-8 bar code is a derivation from the longer European Article Number (EAN-13) code. Both bar codes are standardized in ISO/IEC 15420:2009 [16]. The purpose of introducing EAN-8 was for use on small packages where an EAN-13 bar code would not fit; for example on cigarettes, pencils, or chewing gum packets. The encoding is almost identically to the 12 digits of the UPC-A bar code.

EAN-8 bar codes may originally be used to encode GTIN-8s which is a set of product identifiers from the GS1 System. The GTIN-8 begins with a 2- or 3-digit GS1 prefix (which is assigned to each national GS1 authority) 5-

or 4-digit item reference element depending on the length of the GS1 prefix), and finally a checksum digit.

The GS1 authority ensures the international standardization of product identifiers. However, if companies want to use the bar code to encode own-brand products sold only in their stores, they may use a RCN-8s (8-digit Restricted Circulation Numbers) which is of the pattern `02xx xxxx`, `04xx xxxx` or `2xxx xxxx`, where `x` is a placeholder for arbitrary values that are assigned internally.

The construction of the EAN-8 bar code is straight-forward. The code is comprised of 8 digits. The resulting bar code is structured into a left part of 4 digits and a right part of 4 digits. The left part and the right part have different alphabets as shown in Table 3.

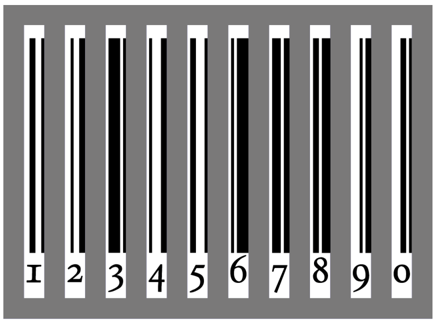
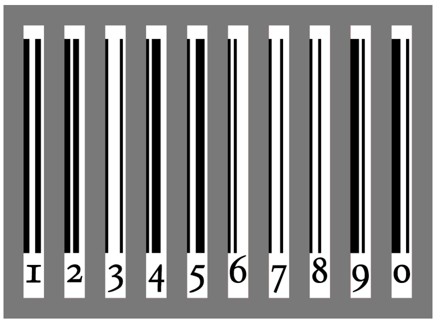
First group of 4 digits	Last group of 4 digits
`LLLL`	`RRRR`
Alphabet used for L part 	Alphabet used for R part 

Table 3: Structure and alphabet of EAN-8 encoding

The alphabet for each digit can also be described using a bit representation, where `1` denotes a black line and `0` a white line (cf. Table 4). Note that the R code is the bitwise-complement of the L code and vice versa.

The checksum calculation of EAN-8 is also very straightforward. The code has 7 data digits $\varrho_i, i \in 1 \dots 7$ and one checksum digit ϱ_8 Each of the positions of the number to be encoded $i \in 1 \dots 7$ is assigned a weight ω_i. The weight is then multiplied with the data digit ϱ_i at position i. Finally, the checksum digit is then calculated by

$$\varrho_8 = 10 - \sum_1^7 \omega_i \varrho_i \mod 10.$$

The weights ω_i as well as an example are depicted in Table 5.

Several programming libraries exist to read a given EAN-8 bar code from an image. In this project we chose the Java-based Zebra Crossing (ZXing)

Digit	L-code	R-code
0	0001101	1110010
1	0011001	1100110
2	0010011	1101100
3	0111101	1000010
4	0100011	1011100
5	0110001	1001110
6	0101111	1010000
7	0111011	1000100
8	0110111	1001000
9	0001011	1110100

Table 4: EAN-8 L- and R-code alphabet as bitwise-representation

Position i	1	2	3	4	5	6	7	
Weight ω_i	3	1	3	1	3	1	3	
Example Code ϱ_i	2	5	9	3	0	0	0	
Example Checksum	$\begin{aligned}\omega_8 &= 10 - ((2 \cdot 3 + 5 \cdot 1 + 9 \cdot 3 + 3 \cdot 1 + 0 \cdot 3 + 0 \cdot 1 + 0 \cdot 3) \mod 10) \\ &= 10 - (41 \mod 10) \\ &= 9\end{aligned}$							

Table 5: EAN-8 positional weights and example calculation of an EAN-8 checksum

[17] due to its widespread use, speed, and available support. A in-depth description of the algorithm to read a bar code from a captured image exceeds the scope of this paper. However, Algorithm 1 gives a concise description of the algorithm that is being used in our scenario.

2.3 Optical Character Recognition (OCR)

Optical character recognition (OCR) is the electronic conversion of images of printed text into machine-encoded text. OCR has been widely used as a form of data entry from printed paper data records, such as bank statements, or computerized receipts. OCR is an active field of research in pattern recognition, artificial intelligence and computer vision.

OCR accuracy can be increased if the output is constrained by a lexicon – a list of words that are allowed to occur in a document. This might be, for example, all the words in the English language, or a more technical lexicon for a specific field. This technique can be problematic if the document contains

Algorithm 1 Generic 1D-Bar Code Reader Algorithm

```
(1) Load the RGB Image and convert it to greyscale
(2) Convert the RGB Image to Greyscale
(3) Binarize the image using a global histogram approach
(4) Compute two 2D-points in the image based on some heurstic
(5) Trace the binary path between both points
(6) Scale the measured binary path interval down to a fixed length interval
(7) Perform pattern matching to the given reference bar code alphabet
(8) Check if checksum is correct and if yes output mapped code,
    otherwise compute another set of points using the heuristic and continue
    with (5) (or abort)
```

words not in the lexicon, like proper nouns. However, in our scenario the input is only comprised of numerical letters and a dot sign.

In the hope to further assist the accuracy of the OCR tools the OCR-A is a being utilized on the Color Checker Card. This font arose in the early days of computer optical character recognition when there was a need for a font that could be recognized not only by the computers of that day, but also by humans. OCR-A uses simple, thick strokes to form recognizable characters. The OCR-A font is monospaced i.e. each character has a fixed fixed-width.

In this project we use a combination of the following three OCR reader:

- Tesseract [18] is a OCR library originally written at Hewlett-Packard between 1985 and 1995. After Hewlett-Packard abandoned the OCR market the code was adopted by Google and was put under open-source. Tesseract is highly configurable, supports over 100 languages including Arabic and Hebrew.
- Ocrad [19] is an optical character recognition program, and part of the GNU Project. It is free software, and is licensed under the GNU GPL. It has been in development since 2003. It is based on character feature extraction.
- Our image processing and evaluation framework uses Matlab. The Computer Vision Toolbox of Matlab features an OCR Reader [20]

that we also use.

2.4 Deformed Lattice Detection Algorithm

An essential building block of our card extraction algorithm is the deformed lattice detection algorithm proposed by Park et al. [12]. The algorithm's purpose is to find anchor points in near-regular patterns that are found nature in an digital image.

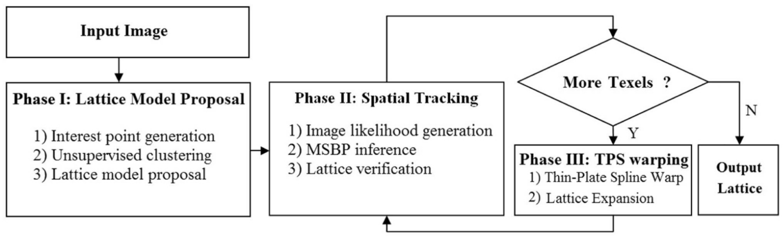

Input Image
Phase I: Lattice Model Proposal
1) Interest point generation
2) Unsupervised clustering
3) Lattice model proposal
Phase II: Spatial Tracking
1) Image likelihood generation
2) MSBP inference
3) Lattice verification
More Texels ?
Y
N
Phase III: TPS warping
1) Thin-Plate Spline Warp
2) Lattice Expansion
Output Lattice

Figure 3: Flowchart of the phases of the proposed algorithm by Park et al. [12]

The algorithm is comprised in three phases as depicted in Figure 3. In the following we summarize the phases of the lattice detection algorithm:

Phase I: Lattice Model Proposal This phase can be described as a discovery process to find a lattice proposal. The phase start with an input image I, and outputs a lattice proposal consisting of a vector pair (t_1, t_2) of two linear independent vectors, a initial texel estimate (a texel is a set of points belonging to a lattice) and an appearance template T_0. The input image is subdivided into blocks of size 50x50 pixel. On each of the partitions the KLT corner extractor [12] is invoked and yields a candidate interest point set of 2D coordinates of at most $N_s = 30$ per 50x50 pixel partition. The final set of all feature points is called S_{klt}. For each of the interest points a centered region of 11x11 pixel is considered and all points in this set are clustered using Mean-Shift Clustering by their regional template similarity. Having the interest points clustered into different groups, then a voting mechanism (RANSAC) is used to find a generator pair for a lattice pattern (t_1, t_2): For each clustered group three points $\{a, b, c\}$ are sampled, where a is chosen at random and b and c are chosen as the closest points in the vicinity to a. An affine transformation t is computed that maps $\{a, b, c\}$ to the lattice basis $\{(0,0), (1,0), (0,1)\}$. The remainder of the points in the clustered group is subsequently transformed to lattice space using this transform

t and the number of points which are within a given threshold to a position in lattice space $(i, j), i, j \in \mathbb{N}$ is counted/voted. These points are called inlier. The random selection of the three points is repeated multiple times and the (t_1, t_2) vector pair with the highest number of counts is chosen as the lattice generator. Thus the output is the best (t_1, t_2) pair, the inlier as a texel estimate (a set of 2D points), and the template T_0 which centered at the point spanning t_1, and t_2.

Phase II: Lattice Expansion Since the lattice present in the image to detect may be geometrically distorted it cannot be assumed that Phase I will detect the entire lattice. Therefore an initial small seed lattice is predicted, refined, and gradually developed into into a larger and larger lattice, while the image is progressively unwarped (Phase III) to remove geometric deformations.

Details of this Phase are beyond the the scope of this paper. However, a Markov Random Field (MRF) model is being utilized here to infer further lattic texel locations. This model both incorporates spatial constraints (dependence on t_1, t_2 generator of lattice points) as well as image similarity (T_0). The inference process is realized using Mean Shift Belief Propagation.

Phase III: Regularized Thin-Plate Spline Warping After Phase II has inferred some further points of the lattice, the found lattice related to its regular origin. This is done by an unwarping step using regularized thin-plate spline warping [12]. This phase is ensures the stability of the lattice model throughout the entire iterative procedure of alternating Phase II and III until no more lattice points can be developed. Phase II and III are executed unless no more texels are found.

An example of the lattice detection algorithm is shown in Figure 4.

2.5 Communication Architecture

In this section we will elucidate the framework in which our card extraction algorithm is run. Our algorithm is comprised of several building blocks and operates in a pipelined manner.

The communication architecture (depicted in Figure 5) follows a star topology where a central component is the communication server. In our framework the communication server implements the mediator pattern: The mobile phone is connected to the communication server and invokes a certain workflow. In our scenario the workflow is the execution of multiple execution

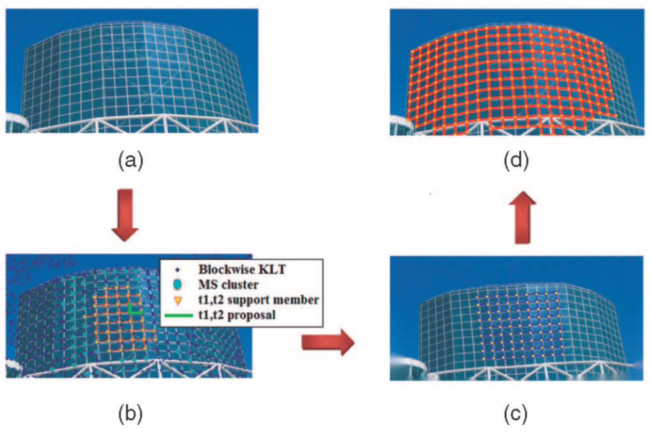


Figure 4: Steps of the Lattice Detection Algorithm: (a) the input image, (b) the lattice proposal after Phase I, (c) the partially developed lattice (note that the image is warped) after some iterations of Phase II/III, (d) the final lattice [12]

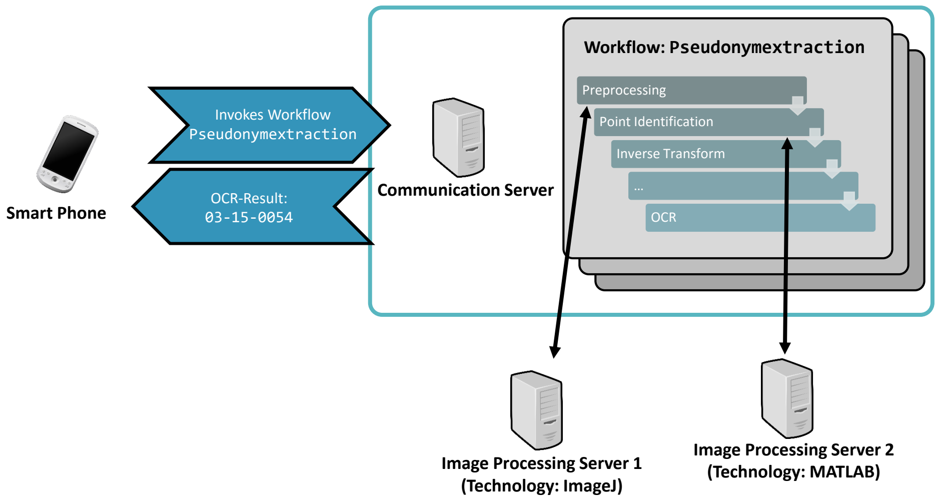

Smart Phone
Invokes Workflow Pseudonymextraction
OCR-Result: 03-15-0054
Communication Server
Workflow: Pseudonymextraction
Preprocessing
Point Identification
Inverse Transform
...
OCR
Image Processing Server 1 (Technology: ImageJ)
Image Processing Server 2 (Technology: MATLAB)

Figure 5: The communication server acts as a mediator between the requesting entity (smart phone) and the workflow specific worker servers (e.g., Image Processing Servers)

steps in image processing and computer vision. The communication server schedules and takes care of the execution of steps in the workflow on dedicated execution servers (image processing servers). One the one hand those execution servers can run different technology stacks or operating systems (Windows/Linux, MATLAB/ImageJ/...), on the other hand the centralized communication server can perform agnostic failure handling as well as load balacing. A workflow can be thought of as a recipe to perform a certain goal. The specification of a workflow includes the specification of the required parameters that are needed to be passed to the execution server, such that the single workflow step can be executed. Furthermore, the communication server can identify workflow items that can be executed in parallel. A single step in the workflow could be the execution of a single pipeline step in our algorithm. For example, a single workflow item in our scenario is the

(i) resizing of an input image to a fixed width/height,

(ii) the calculation of an edgemap,

(iii) an invocation of the aforementioned lattice detection algorithm,

(iv) the invocation of the card extraction algorithm,

(v) the invocation of an OCR software,

(vi) the invocation of a bar code reader algorithm,

(vii) post processing and validation of OCR and bar code readout, or

(viii) the subsequent image calibration.

Therefore, we obtain the freedom to be technology independent as long as there is a technology-specific binding implemented between the execution server and the communication server. Additionally, the potentially resource constrained client of the communication server (e.g.,the mobile phone) is not bothered with the complex inner-workings of the underlying infrastructure.

3 Card Extraction

In this section we will describe our proposed card extraction algorithm.

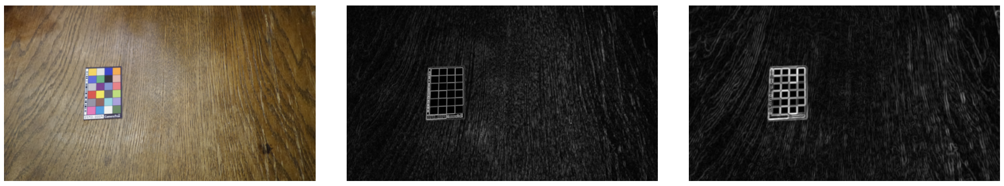
3.1 Outline

The input parameters of our algorithm is a RGB image I_{RGB} of arbitrary resolution (we assume the image is larger than 1000x1000 pixel) containing a color reference card. The output of the algorithm are the either the bar code/ocr readouts or an error indication if the algorithm was unable to either detect the card or some step (e.g., the barcode reader) encontered an error condition.

3.2 Edgemap Calculation

The first phase of the algorithm is a preprocessing step for the subsequent lattice detection phase. The goal of this phase is to emphasize the edges of the color checker card placed within the image as a good starting point for the lattice detection. Simultaneously, in this phase we resize the image obtain a smaller input image in the subsequent lattice detection phase.

We propose two variants of the edgemap calculation, which are heuristic approaches to yield an image that contains good contour traces of the color card patches. Both variants differ in their level of blurriness (c.f. Figure 6).



(a) Resized RGB Image (b) Variant A (c) Variant B

Figure 6: Original image (a) and different edgemap variants (b-c)

Variant A

Given a maximum size $\delta_{\max}$, a blur strength δ_{blur} the steps of the edgemap calculation phase are as follows:

1. Resize the I_{RGB} to I'_{RGB} such that the height/width of I'_{RGB} are bounded by $2\delta_{\max}$.

2. Transform the I'_{RGB} to a greyscale image I'_{grey}.

3. Compute and normalize the Prewitt gradient operator g_{grey} magnitude for I'_{grey}. Also compute the same operator $g_{\text{c}}, c \in \{R, G, B\}$ for each of the RGB color channels of I'_{RGB} separately.

4. Compute the edgemap E image as follows:
$$\begin{aligned} E(i,j) \quad = \max(\quad & g_{\text{grey}}(i,j), \\ & g_{\text{R}}(i,j), g_{\text{G}}(i,j), g_{\text{B}}(i,j), \\ & \tfrac{2g_{\text{B}}(i,j)+g_{\text{G}}(i,j)}{3}, \tfrac{2g_{\text{R}}(i,j)+g_{\text{B}}(i,j)}{3}, \tfrac{2g_{G}(i,j)+g_{\text{R}}(i,j)}{3}), E(i,j) \leqslant 1 \end{aligned} .$$

5. Resize E to E' such that the height/width of E' are bounded by $\delta_{\max}$.

6. Finally, perform a Gaussian blur of strength δ_{blur} on the final edgemap E''.

Variant B

Variant B follows is almost the same procedure as in Variant A. However, here we introduce another Gaussian blur of strength δ_{blur2} after step 1. The following gradient operators (Step 2 and following) are then computed on this blurred image instead of the resized original image.

3.3 Lattice Detection Algorithm

Having computed the edgemap in the previous phase we invoke the lattice detection algorithm of Park et al. [12] on the resulting edgemap image E''. The lattice detection algorithm yields a point cloud of of 2D-coordinates of the grid points. In our scenario we expect the grid points to be the color card coordinates due to the specific nature of our edgemap. Since the edgemap input image was calculated on a resized image, we upscale coordinates and obtain the set L of the lattice coordinates in our original image. However, the lattice detection algorithm only gives a coarse and potential incomplete list of grid points in the image (cf. Figure 7).

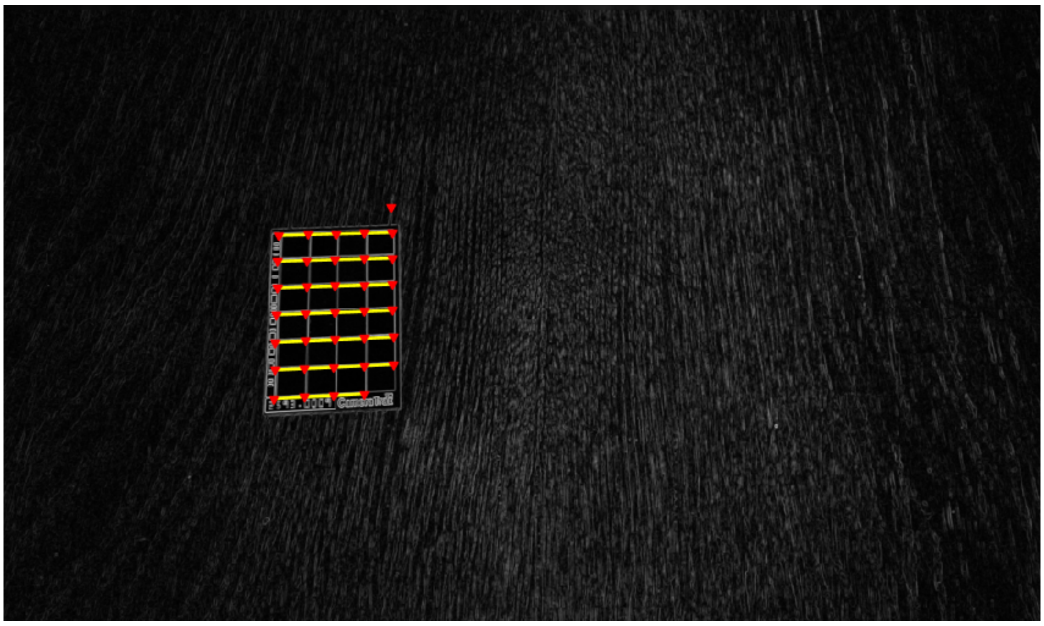


Figure 7: Result of the Lattice Detection algorithm. Lattice Points (red) on a grid (yellow) were detected. The detected points may be coarse, not complete, or incorrect.

3.4 Point Labeling

Due to rotation and perspective distortions the sole knowledge of some point coordinates within the card is not sufficient to identify the regions for the card patches. First, as previously stated, the list L of lattice coordinates might be incomplete or contains false positives. Secondly, the points are not yet associated to the card's topology.
In order to obtain a mapping of the points to the card's topology and to filter out false positives, we compute a feature vector for each of the points in L. The feature vector should incorporates the color information of the neighboring color patches as shown in Figure 8.

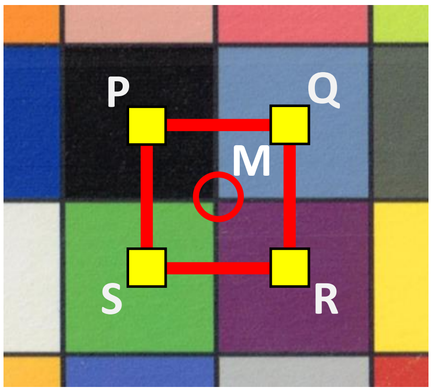


Figure 8: The feature vector is computed using the averaged color of four points within the quadratic surrounding of the point M.

First we compute the average distance d for each point to the closest neighboring point:

$$\alpha = \frac{\sum_{i \in L} \min_{j \in L, i \neq j} \{\|\mathbf{i} - \mathbf{j}\|_2\}}{|L|}.$$

We can then define a offset o and a region size u as follows:

$$\gamma = \max(30, \frac{d}{2}),$$

$$\beta = \max(3, \sqrt{\frac{d}{4}})$$

Now given a point $M = (m_1, m_2) \in L$, we then compute four points p, q, r, s in rectangular neighborhood of S such that:

$$p(m_1, m_2) = (m_1 - \gamma, m_2 - \gamma)$$

$$q(m_1, m_2) = (m_1 + \gamma, m_2 - \gamma)$$
$$r(m_1, m_2) = (m_1 + \gamma, m_2 + \gamma)$$
$$s(m_1, m_2) = (m_1 - \gamma, m_2 + \gamma)$$

For $i \in \{p, q, r, s\}$ we compute the mean RGB values (R_i, G_i, B_i) as well as $A_i = R_i + G_i + B_i$ within the neighboring region of i spanning β pixels. Finally, we can compute the feature C vector for M. We abbreviate p for $p(m_1, m_2)$ here etc.:

$$\begin{aligned} C(m_1, m_2) \quad = (\quad & R_p/A_p, G_p/A_p, B_p/A_p, \\ & R_q/A_q, G_q/A_q, B_q/A_q, \\ & R_r/A_r, G_r/A_r, B_r/A_r, \\ & R_s/A_s, G_s/A_s, B_s/A_s) \end{aligned}$$

Knowing the RGB values for our color checker card we can also compute the same feature vector for fixed points L' between color patches in our color checker card. Subsequently, we can then compute a mapping between L and L' using our feature vector. For each point in $l' \in L'$ we compute a mapping $m : L' \to L$ that minimizes the Euclidian distance between the two obtained feature vectors. In order to be able to handle rotations of the card, we repeat the computation of the feature vector and shift the computation of the points p, q, r, s by 90, 180, and 270 degree. Finally, we choose the mapping which has the least accumulated error amongst all rotations (cf. 9).

This mapping yields to a labeling of lattice points detected in the image to inner grid points of the points of the color card. However, this mapping can still be erroneous for some points i.e. some points within the lattice are assigned to wrong topology points of the card. Moreover, as seen in 9 due to the design of the lattice detection algorithm, the obtained points may not perfectly resemble the intersections of neighboring color patches.

3.5 Fine Adjustment

In order to obtain a more accurate match of the intersections, we propose a fine adjustment step that compensates the inaccuracies of the lattice detection. The overall assumption here is that the point labeling step has already chosen points that are present at such a intersection but that are affected by an error i.e. the real intersection is in a certain vicinity from the original point. Our approach is derived from Wang et al. [11] but was adapted to our scenario.

The idea here, again, is to only consider the surrounding region of the point under examination: We again use the same points that were previously

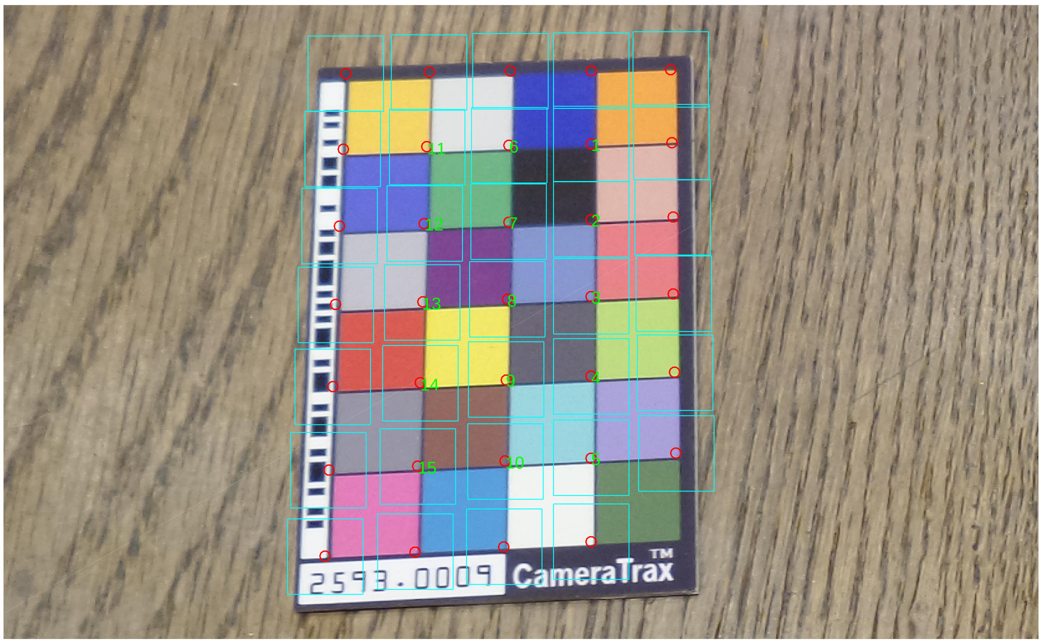


Figure 9: The feature vector is computed for all lattice points and matched to known feature vectors of the inner color patches of the color checker card's specification

used to compute the feature vector (see subsection 3.4), but now we consider color gradient amongst a line between two of those points at the regions boundary. Our observation is that the intersection point is always on a black line between two color levels.

In the following to simplify we focus on the one dimensional case for one color level. In reality, we deal with three color levels and two dimensions.

For two given discrete one dimensional points at position l and r,$l < r$, let C_l and C_r denote their color intensity level.

For discrete points between l and r at discrete position i we define:

$$d(i) = \min(|C_i - C_l|, |C_i - C_r|)$$

Additionally, we define the color derivative as follows:

$$\Delta d(i) = \begin{cases} 0 & i = l \\ |d(i-1) - d(i)| & \textit{otherwise} \end{cases}$$

That means $\Delta d(i)$ denotes the absolute change in color. Because we assume that the color changes from one color level to another, we can approximate

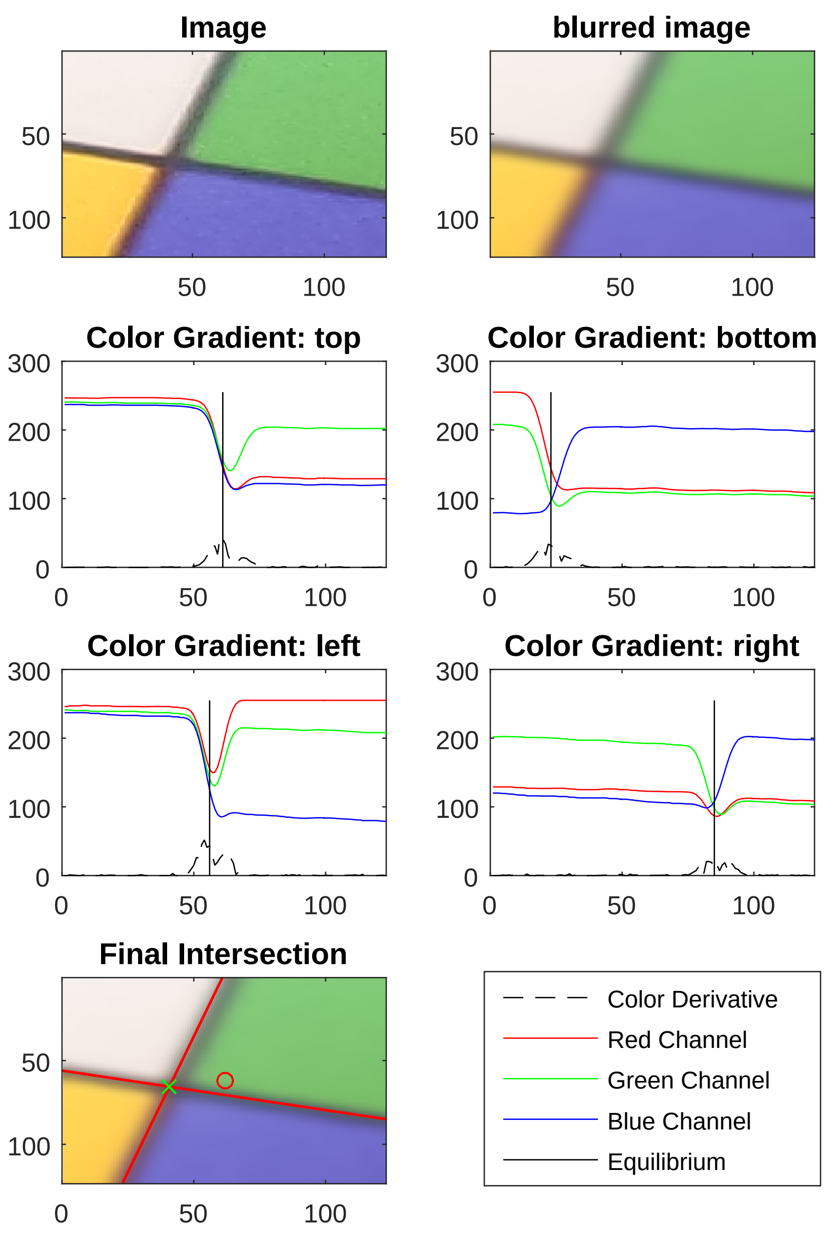


Figure 10: Example of a single fine adjustment step. Based on an initial starting point (red), we consider an excerpt from the image and blur it.Then the color derivatives are calculated for the outermost top/bottom/left and right segments of this excerpt, which yields to an approximation for the intersection point (green)

the intersection point c with the following equitation:

$$\int_{l}^{c} \Delta d(x)dx \approx \int_{c}^{r} \Delta d(x)dx$$

We extend this approach to incorporate all three color channels in our scenario by computing the color derivatives for each color channel separately and finally adding up their color derivatives. The intersection point is then computed over the accumulated color derivative.

Using this method we obtain an approximate location for center of a black line, when drawing a line between two color patches. We can now apply this method to find out the position of the black color line at the top, the bottom, left and right of the point of interest (cf. Figure 10). Having those four positions, we can compute a linear intersection, which is our final fine adjustment. It is noteworthy that since the computation for each point is independent of one another, the fine adjustment computation for each point can be computed in parallel.

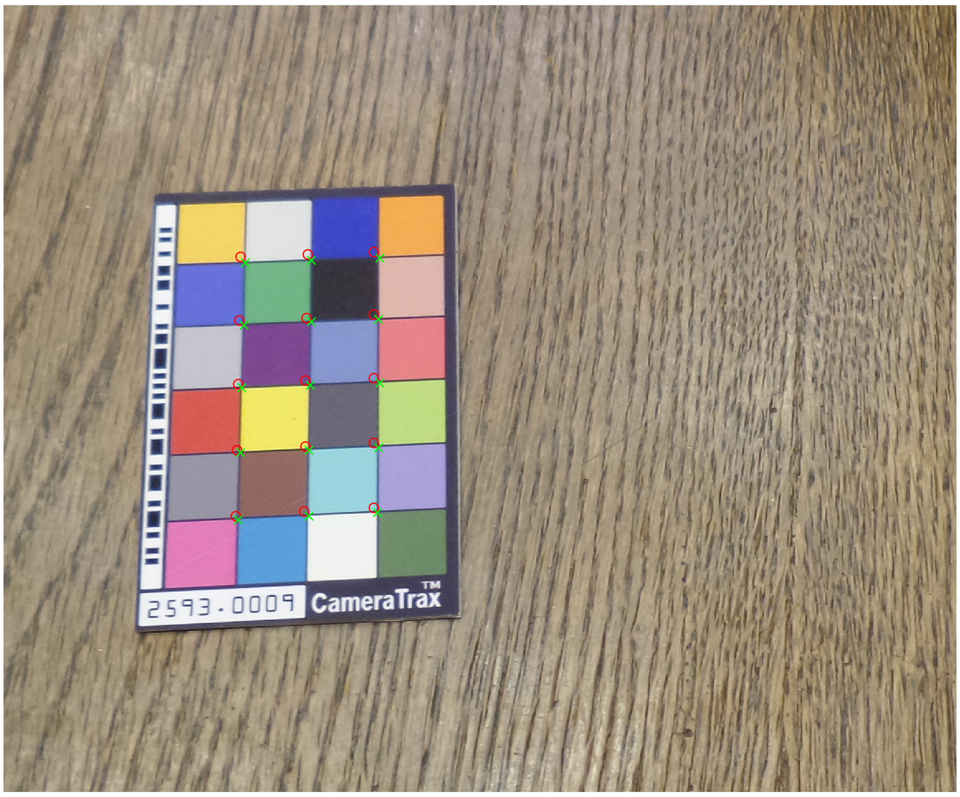


Figure 11: Result after Fine Adjustment step, the green points are now centered at the intersections

3.6 Perspective Transform

After we have filtered, labeled and fine-adjusted the detected lattice points the next goal is to perform a perspective transform on the image such that the perspective distortion of the card in the image plane removed. Traditionally, four points suffice to calculate a perspective transform. However, the point labeling step may labeled some points wrong. Furthermore, even after the fine-adjustment step there might be some points that do not perfectly fit. Also since we do have more than four points we want to pick the best combination of those to ensure a very precise perspective transformation.

To do this we use the M-estimator SAmple Consensus (MSAC) algorithm, which is a variant of the Random Sample Consensus (RANSAC) algorithm.

Given a distance threshold d, a set of fine-tuned lattice coordinates L, a set of coordinates on the actual card L' and a point labeling mapping $m : L \to L'$, the steps of the MSAC algorithm are as follows:

1. From the set of fine-tuned lattice coordinates L pick a set of four points $S \subseteq L$ at random and compute a perspective transformation T of those points to their labeled card coordinates.

2. From each of the remaining points $p \in L \setminus S$ of the set of fine-tuned lattice coordinates that have not been used in the previous transform yet, compute the projection of the transformation $T(p)$. If the distance between the transformation result and coordinates of the mapping of the point is less than the threshold d, than include this point into a *Consensus Set* C, otherwise this point is discarded out as an outlier.

3. Evaluate the current found transform i.e.: $c = \sum_{i \in L} p(i)$, where

$$p(i) = \begin{cases} \|T(i) - m(i)\|_2 & \text{if } \|T(i) - m(i)\|_2 \leqslant d \\ k & \text{otherwise, where } k \ll d \text{ is a constant} \end{cases}$$

   If the current c is smaller than the best previous c we take the save the current transform T as the current best transform T_{best}.

4. Repeat steps 1 - 3 until the number of iterations n exceeds the following inequality:

$$n > \frac{\log(1-p)}{\log\left(1-(1-\epsilon)^4\right)}$$

   where p is the confidence level to include all inliers and ϵ is an estimate of the relative amount of outliers in the given data set. Since ϵ is unknown, ϵ can be estimated on-the-fly with:

$$\epsilon \approx \left(\frac{|C|}{|L|}\right)^4$$

5. For the final transformation compute a final perspective transform T_{final} using all inliers of the best consensus set C by using the least square fitting approach.

Finally, we after having obtained T_{final}, we can apply the transformation to the input image and obtain an image where the color checker cards coordinates are normalized an example of a perspective transform is depicted in Figure 12 and Figure 13.

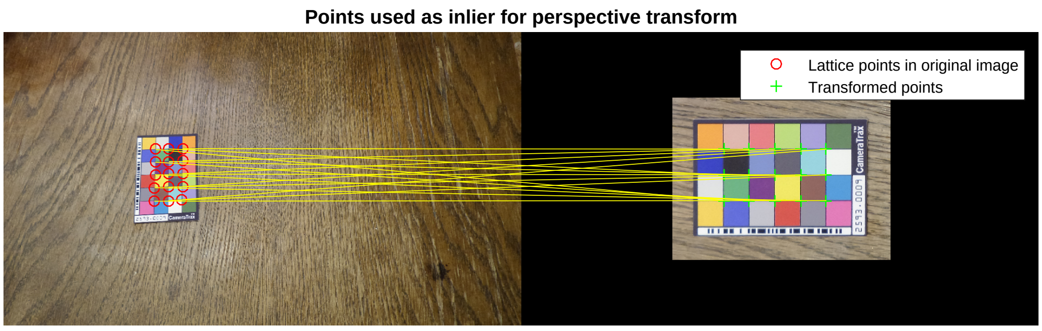

Points used as inlier for perspective transform
○ Lattice points in original image
+ Transformed points
CameraTrax
2593-0009

Figure 12: Image before and after computing the perspective transform. All lattice coordinates are used as inliers.

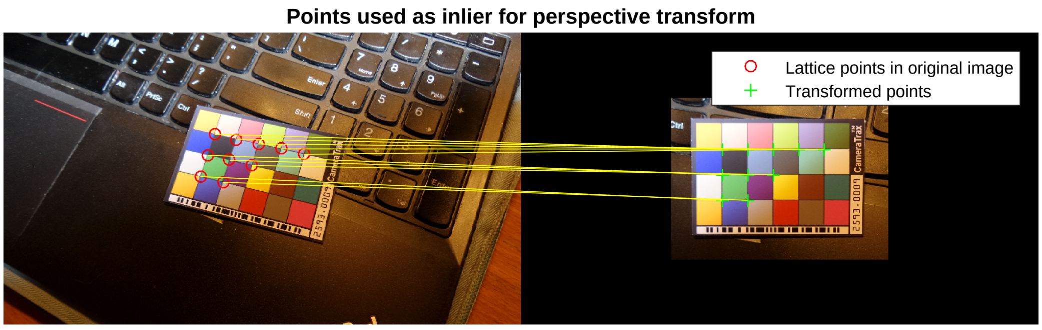

Points used as inlier for perspective transform
○ Lattice points in original image
+ Transformed points
CameraTrax
2593-0009

Figure 13: Image before and after computing the perspective transform Only some lattice coordinates are used for the transform.

3.7 Bar code readout and OCR readout

Having the image perspectively transformed we are able to segment the parts of the image where the bar code and the textual representation of the study subject's identifier is located, since we know the positions of the respective areas from the color cards specifications. This allows us to invoke a bar code reader on the extracted bar code patch as well as OCR readers on the text patch.

3.8 Algorithm Variants

In the previous section we described the basic building blocks of our algorithm. However we have implemented and evaluated several variants of the algorithm that we summarize in this section.

3.8.1 Different Edgemaps

As already described in subsection 3.2 we implemented two different versions of the edgemap, which is being used as a starting point for the lattice detection algorithm.

3.8.2 Variant: Single Pass vs. Double Pass

Apart from using different edgemap another variant that we propose is performing the perspective transform and the preceded fine-tuning step twice or not. That means in the *Single Pass* variant there is only one iteration of point labeling, followed by a fine adjustment step and finally the perspective transform is computed, while in the *Double Pass* variant we perform an additional second perspective transform on the basis of the result of the first transformation. That means after we have performed the first perspective transformation we invoke the fine adjustment again on the transformed coordinates and calculate a second perspective transform as described in subsection 3.6. The doubly transformed image is then used for OCR/Bar code recognition.

3.8.3 Variant: Full Lattice Detection vs. Phase I only

During our evaluation we observed that the a large portion of the overall runtime of our algorithm is spend in the lattice detection algorithm. Moreover, the most expensive part of the lattice detection algorithm are Phases II and III. In the *Full Lattice Detection* variant we invoke the entire lattice detection.

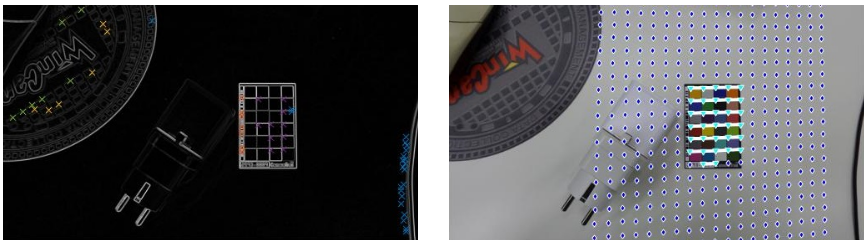

(a) Result after Phase I (b) Fully developed Lattice

Figure 14: The points including different lattice proposals (a) and a fully developed lattice (b).

An example of the differences between the results after Phase I and the final Lattice is depicted in Figure 14.

However, we observed that the initial lattice candidate already yields a good estimate and due to this in the in the *Phase I* only variant we stop the lattice detection algorithm after Phase I and use the lattice candidate set as an direct input for the subsequent point labeling algorithm instead of fully developing the entire lattice.

3.8.4 Variant: Alladjust vs. Only Adjust Point Labeling

In the previously Phase I only variant, we furthermore distinguish between the *Alladjust* and *Only Adjust Point Labeling* variant. The former is invoking the fine adjustment before labeling the points. The latter is only performing the fine adjustment after labeling the points.

3.8.5 Variant: Equidistant Sampling as Baseline

In order to examine the significance of the effect of the lattice detection algorithm, we also implemented a baseline variant where instead of obtaining the points from the lattice detection algorithm we used a systematic uniform sampling approach. That means we choose equidistant sampling points across the entire height/width of the image and perform fine adjustment on each of this points and then perform point labeling (see Figure 15).

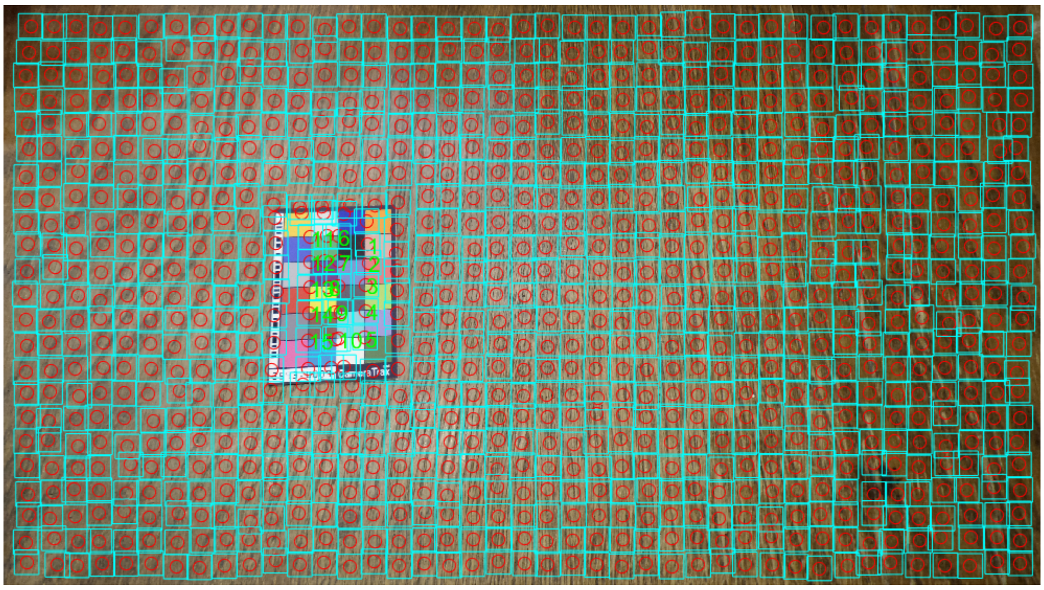


Figure 15: Equidistant Variant without lattice detection: Uniform equidistant samples across the entire length of the image. First the fine adjustment step is applied and then the point labelling is performed

4 Experimental Results

4.1 Setup and Dataset

The following results were performed on a total of 208 images. All images have been taken by the same photographer using different motives, arbitrary card rotations and arbitrary camera-to-card distances. The camera in use was a Samsung Galaxy K with a resolution of 5184x2916 pixel. This set has not been preprocessed or filtered in any kind, and contains some images that are too unsharp, jittery or blurred to detect the bar code of the card properly. Also in some images the card to be detected is semi-covered. All images contain a card. The algorithm was executed on a Lenovo ThinkPad W510 with an Intel Core i7 820QM Quad Core Processor and 12GB of main memory.

4.2 Evaluation Criteria

Our evaluation considers the overall runtime of the proposed variants. Furthermore, we use the readout success to evaluate accurracy of the approaches under consideration. We obtain ground-truth by always using the same card in every picture containing the same identifier code.

4.3 Baseline

We use the equidistant sampling approach as a baseline to determine worst-case runtime as well as a baseline success rate. We vary the amount of samples taken from one sample per 250 pixel (≘ 241 samples) up to one sample per 100 pixel (≘ 1510 samples). The runtime is depicted in Figure 16. Further-

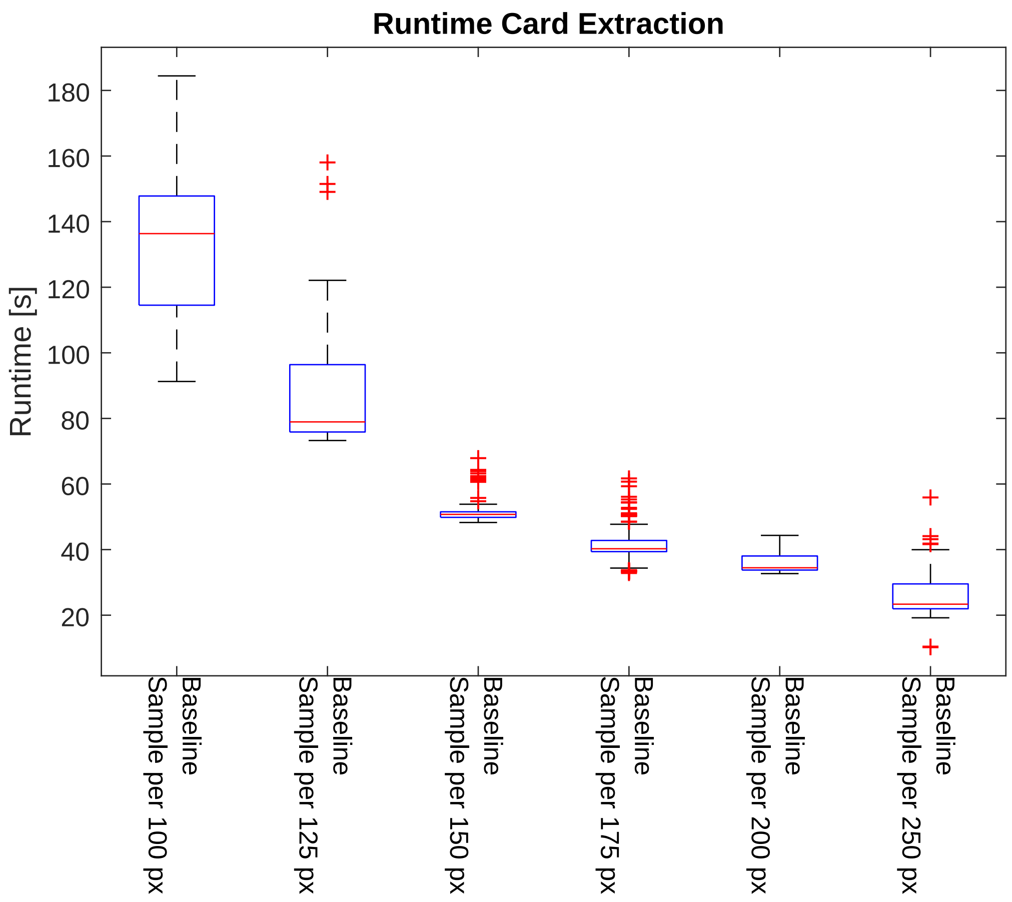


Figure 16: Total runtime for different baseline variants

more, we summarize the readout success for the different baseline variants in Figure 17.

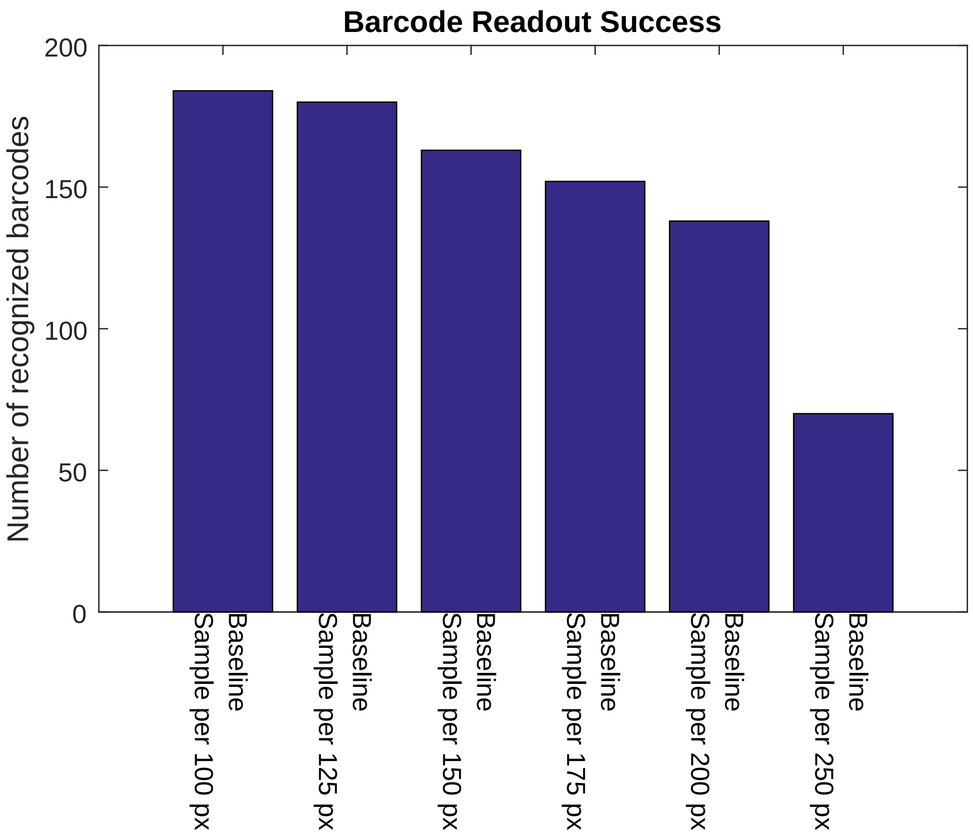
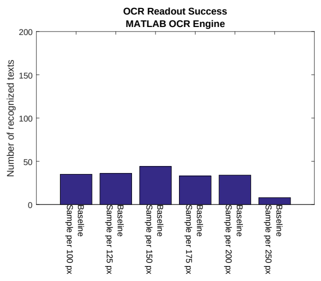
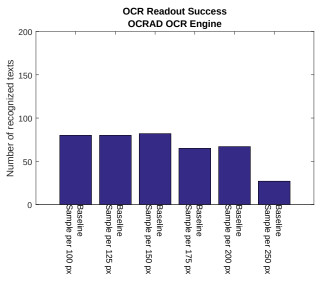


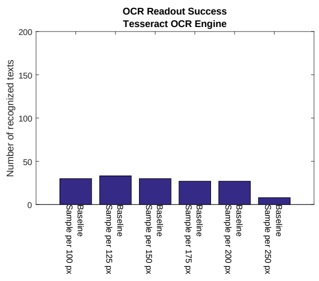
Figure 17: Barcode Readout Success for different baseline variant



(a) Matlab



(b) OCRAD



(c) Tesseract

Figure 18

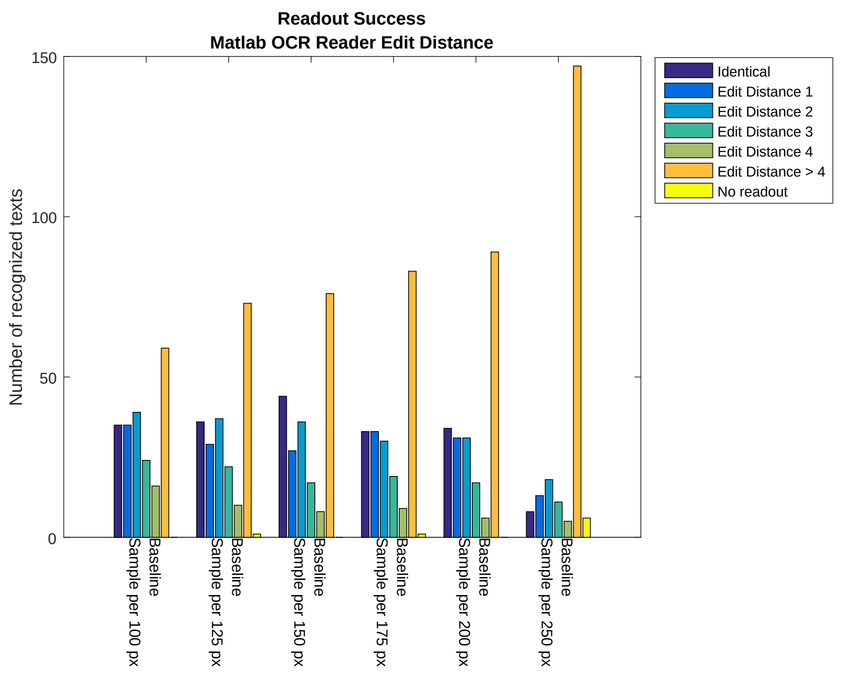


Figure 19

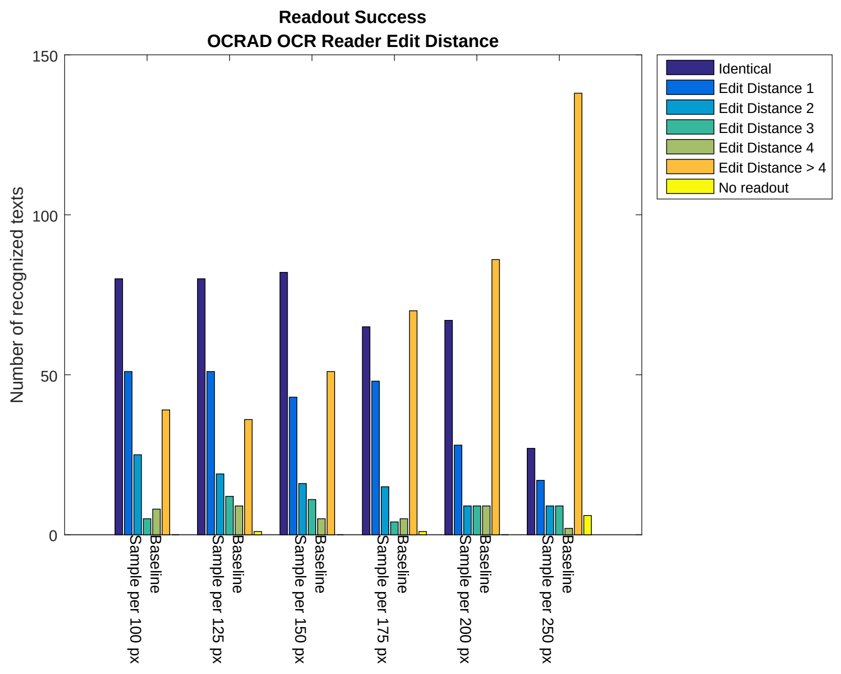


Figure 20

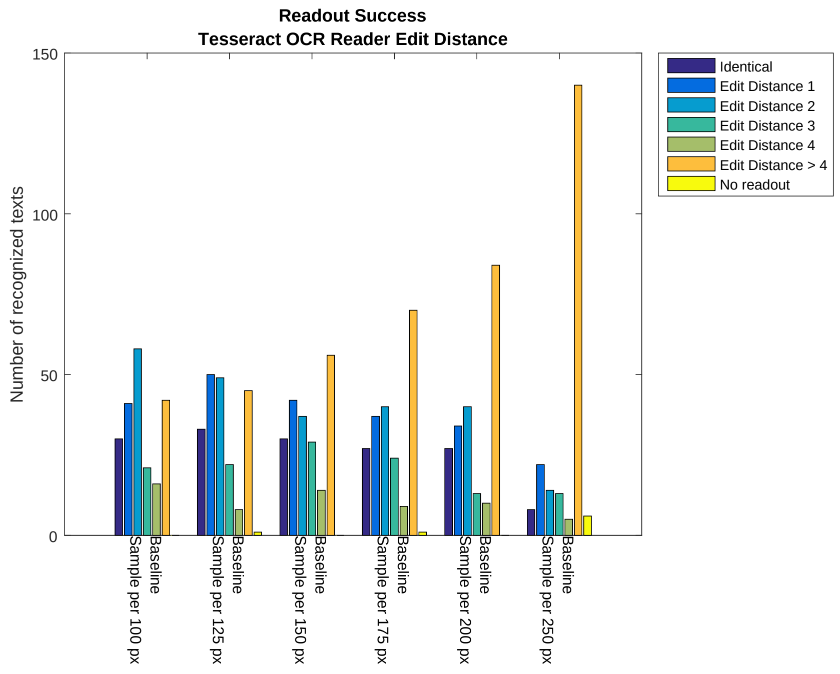
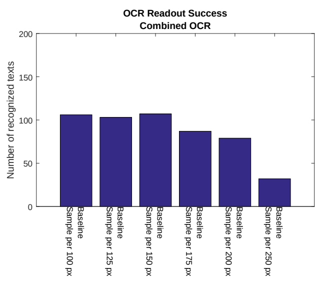
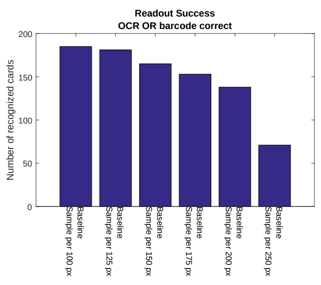


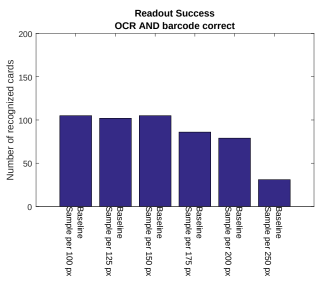
Figure 21



(a) All OCR Readers



(b) OCR OR Bar Code



(c) OCR AND Bar Code

Figure 22

Summary

We observe that in general the bar code readout is much more robust than the OCR readout. In the best case the our baseline approach detects the bar code in 184 out of 208 images ($\widehat{=}$88%) at the expense of a long runtime. If we reduce the number of samples to obtain a reasonable runtime the number of recognized bar codes as well as the OCR readout success rate drops. Note that all baseline variants take longer than 20 seconds.

4.4 Results for Edgemap Variant A

In this section we present the evaluation results for our algorithm variant using the lattice detection algorithm and using an edgemap type of Variant A (see subsection 3.2).

Timing

We measure the total runtime of the algorithmic variants per image (Runtime Total). The total runtime consists of the runtime of the lattice detection algorithm and the card-extraction algorithm. When only using the Phase I of the lattice detection algorithm the runtime decreases significantly.

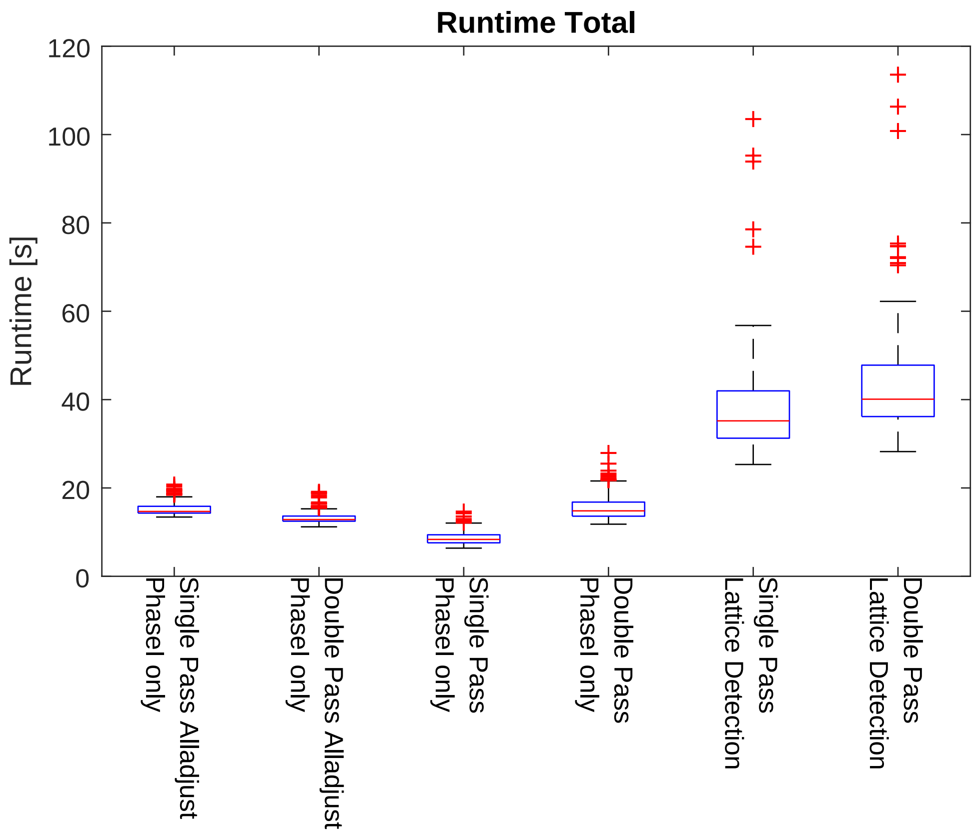


Figure 23

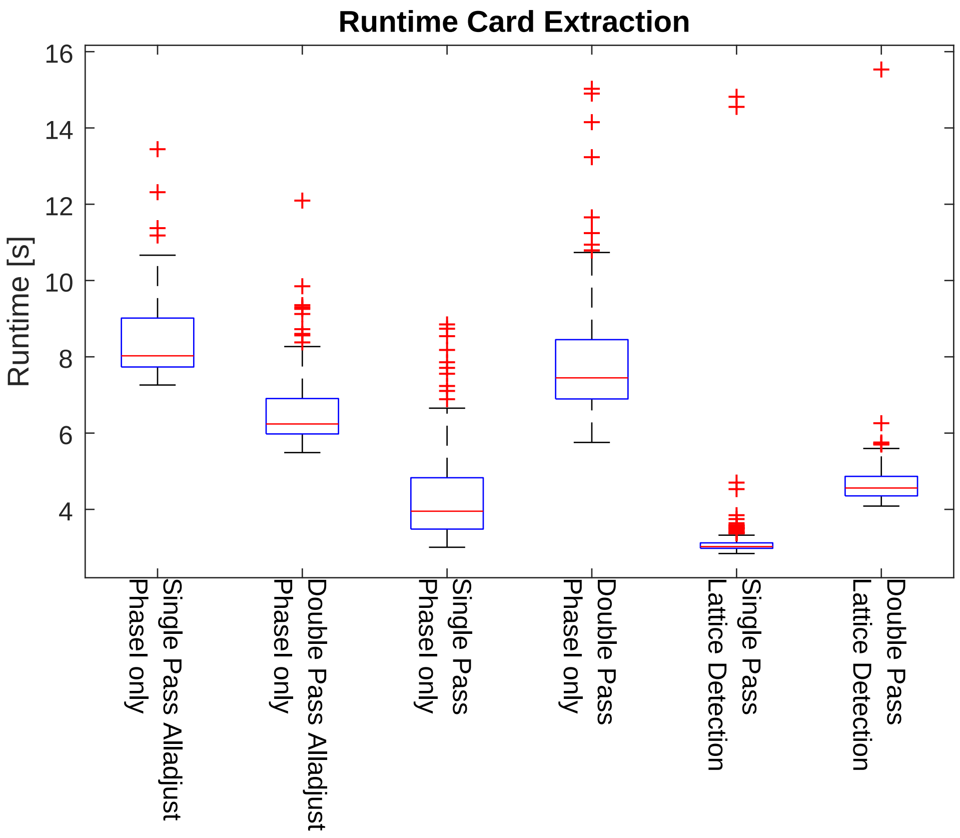

Runtime Card Extraction
Runtime [s]
16
14
12
10
8
6
4
Single Pass Alladjust Phasel only
Double Pass Alladjust Phasel only
Single Pass Phasel only
Double Pass Phasel only
Single Pass Lattice Detection
Double Pass Lattice Detection

Figure 24

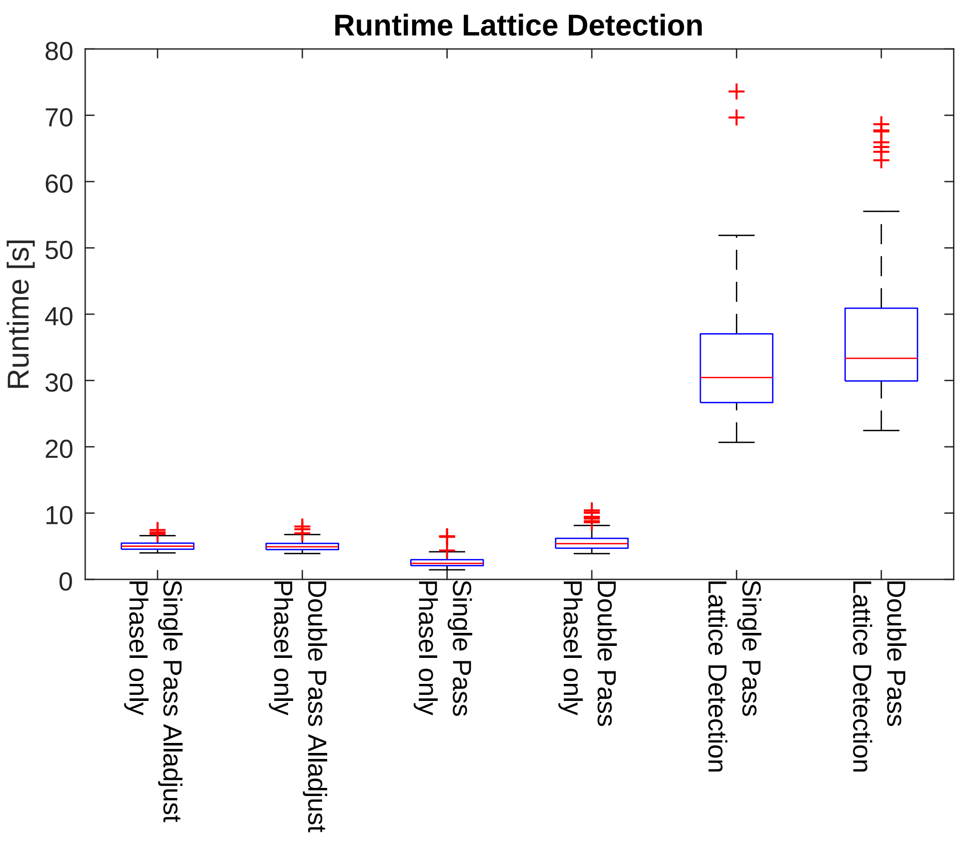

Runtime Lattice Detection
Runtime [s]
0
10
20
30
40
50
60
70
80
Single Pass Alladjust Phasel only
Double Pass Alladjust Phasel only
Single Pass Phasel only
Double Pass Phasel only
Single Pass Lattice Detection
Double Pass Lattice Detection

Figure 25

Readout Success

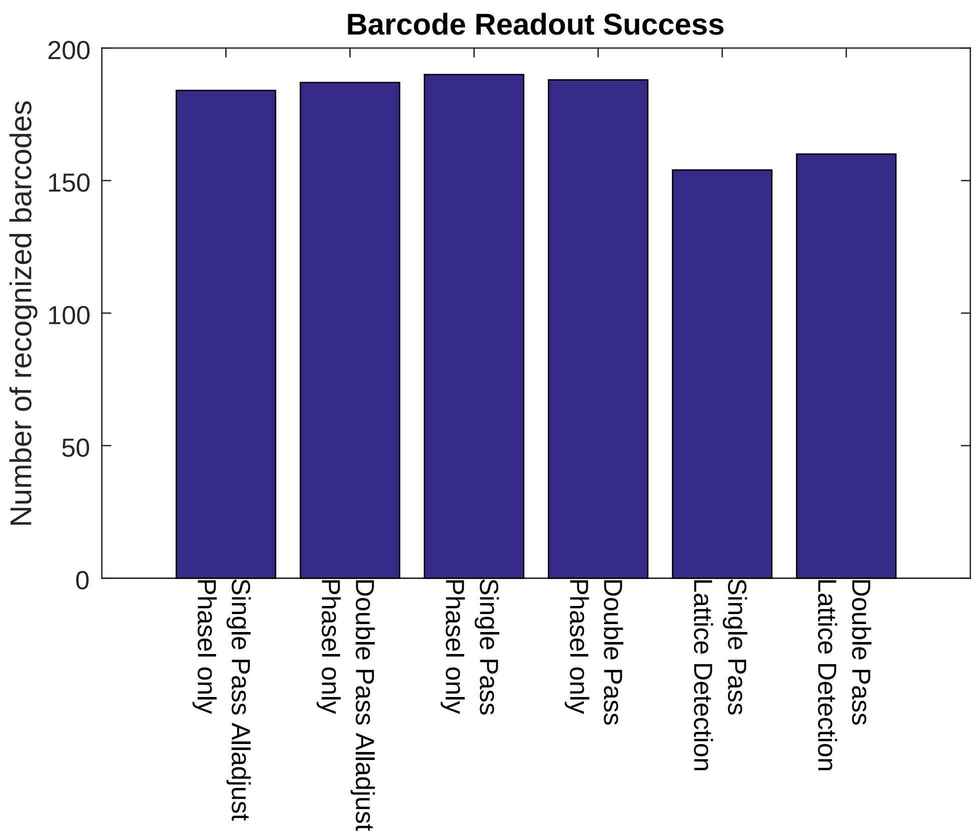
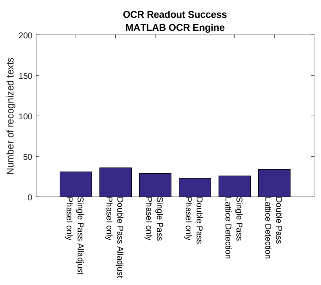
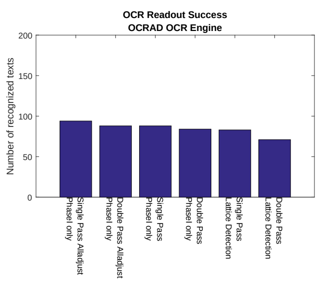


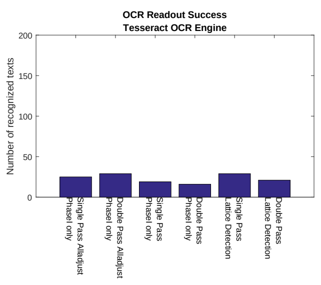
Figure 26



(a) Matlab



(b) OCRAD



(c) Tesseract

Figure 27

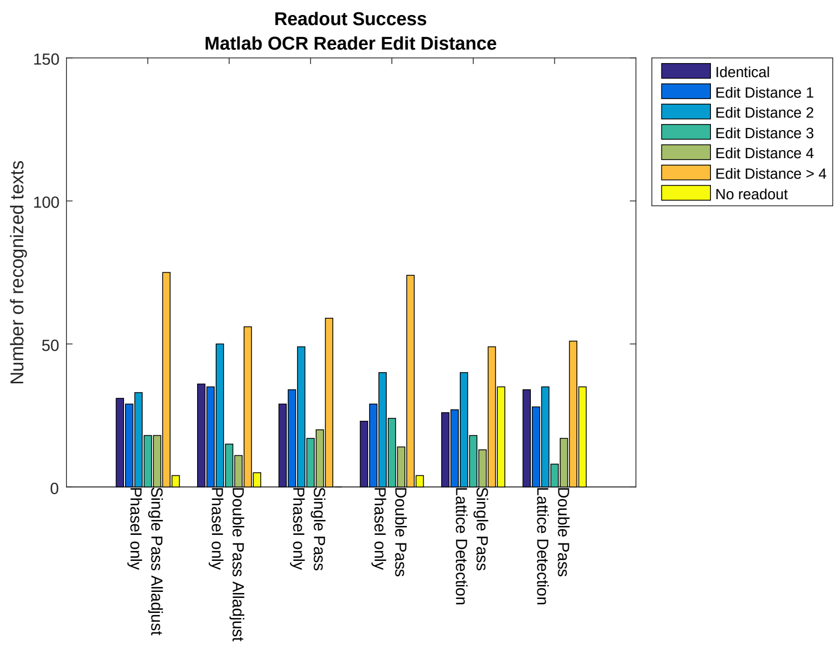


Figure 28

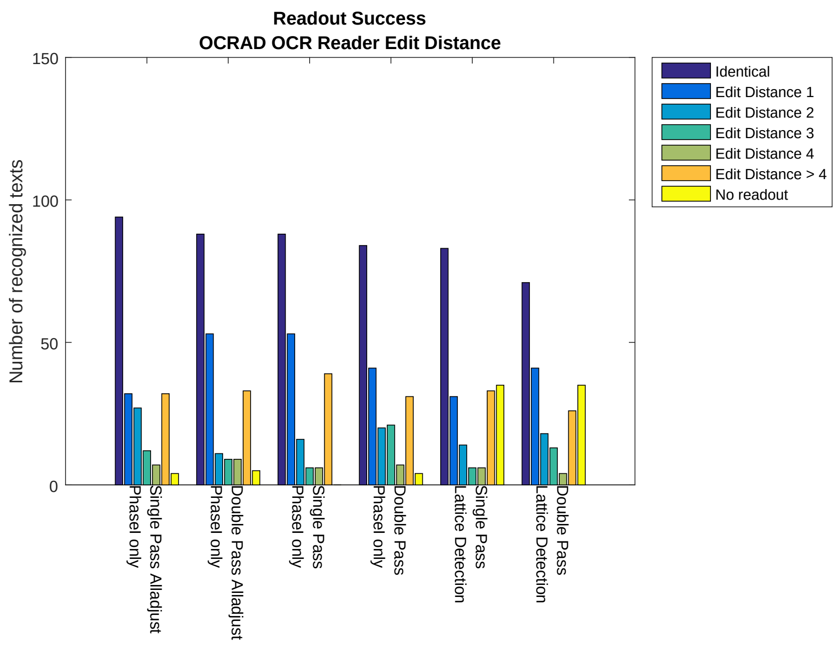


Figure 29

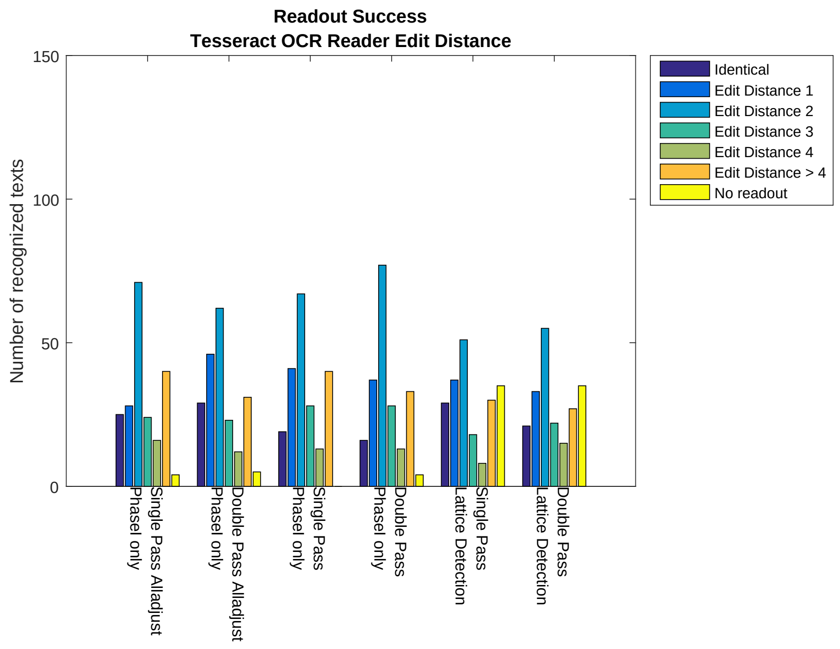
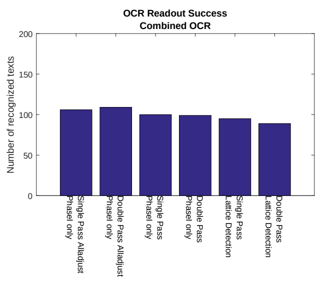
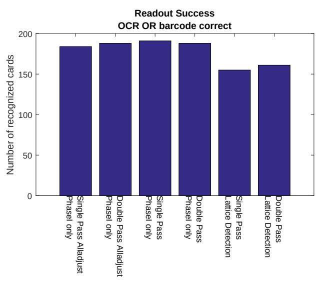


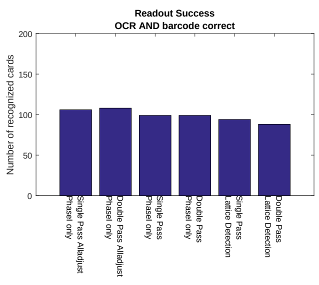
Figure 30



(a) All OCR Readers



(b) OCR OR Bar Code



(c) OCR AND Bar Code

Figure 31

4.5 Results for Edgemap Variant B

Timing

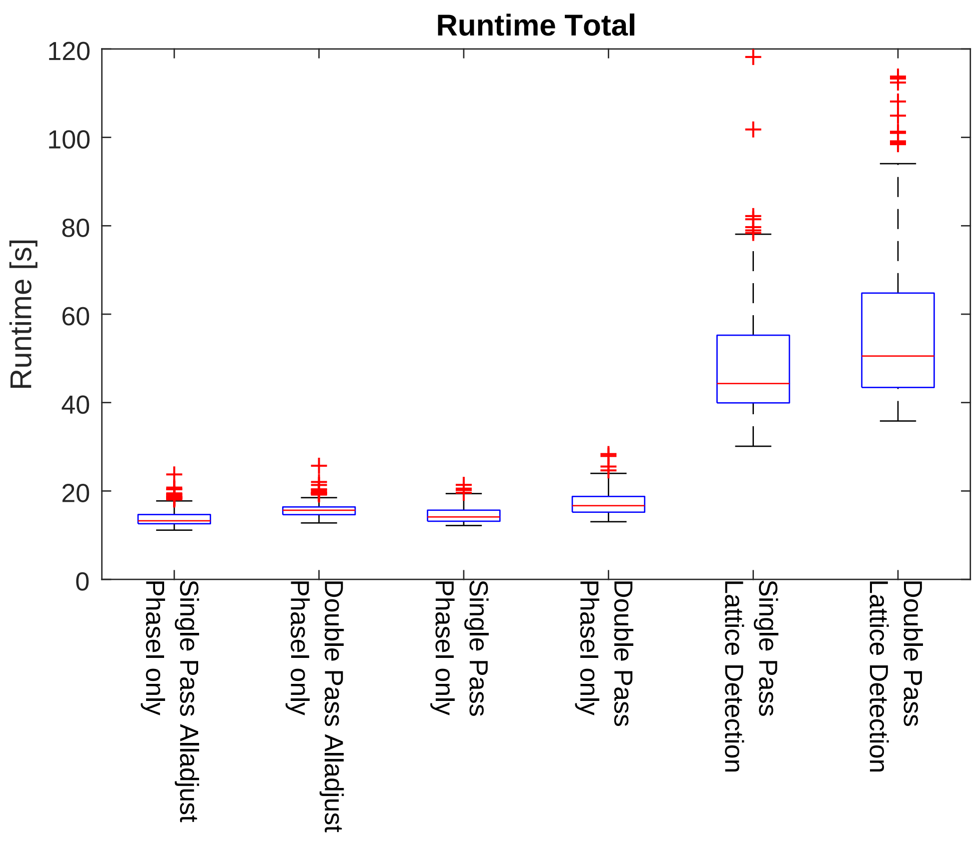

Runtime Total
Runtime [s]
0
20
40
60
80
100
120
Single Pass Alladjust Phasel only
Double Pass Alladjust Phasel only
Single Pass Phasel only
Double Pass Phasel only
Single Pass Lattice Detection
Double Pass Lattice Detection

Figure 32

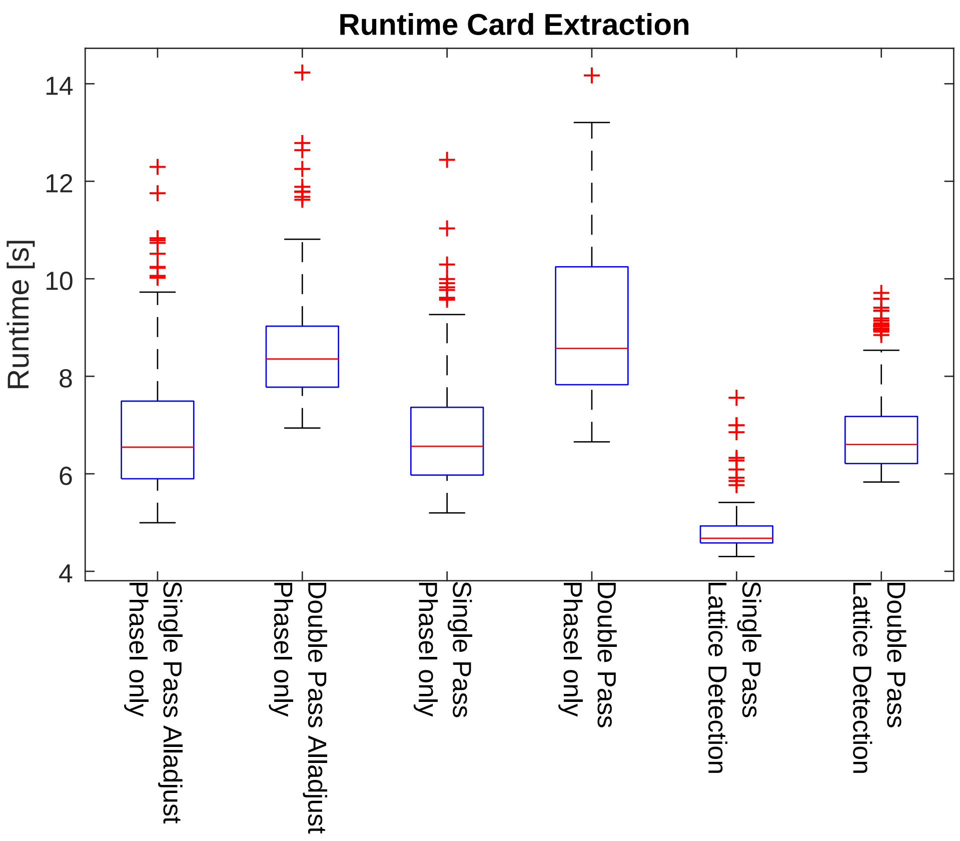

Runtime Card Extraction
Runtime [s]
14
12
10
8
6
4
Single Pass Alladjust Phasel only
Double Pass Alladjust Phasel only
Single Pass Phasel only
Double Pass Phasel only
Single Pass Lattice Detection
Double Pass Lattice Detection

Figure 33

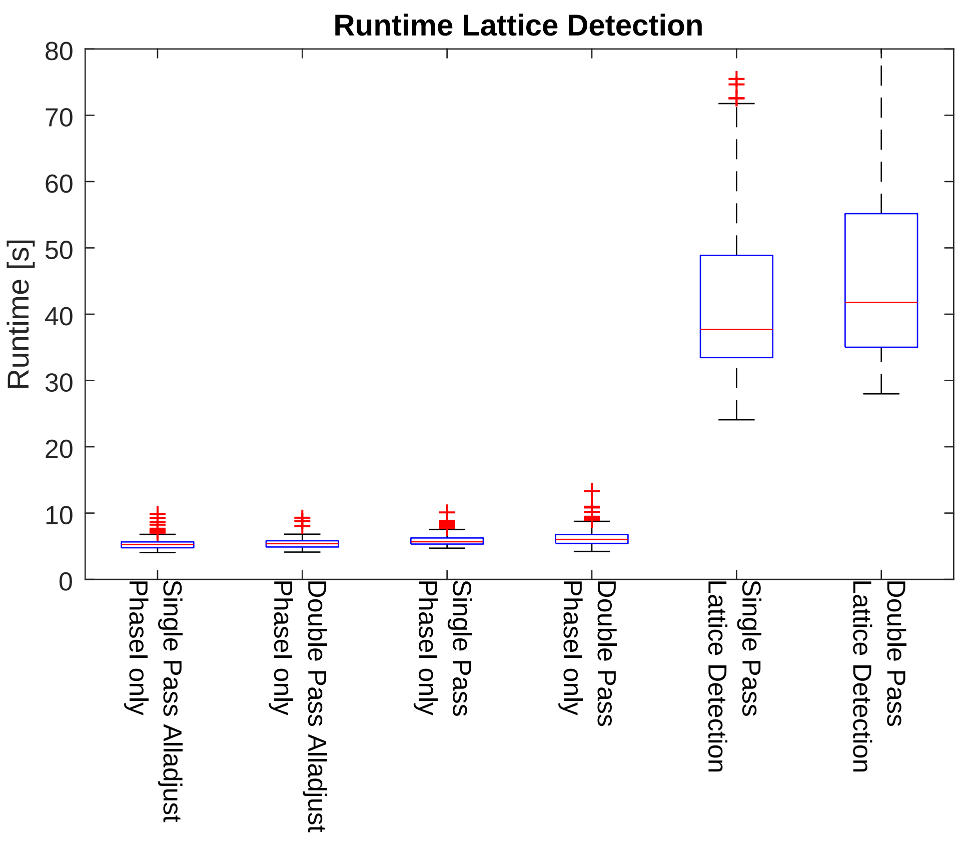

Runtime Lattice Detection
Runtime [s]
0
10
20
30
40
50
60
70
80
Single Pass Alladjust Phasel only
Double Pass Alladjust Phasel only
Single Pass Phasel only
Double Pass Phasel only
Single Pass Lattice Detection
Double Pass Lattice Detection

Figure 34

Readout Success

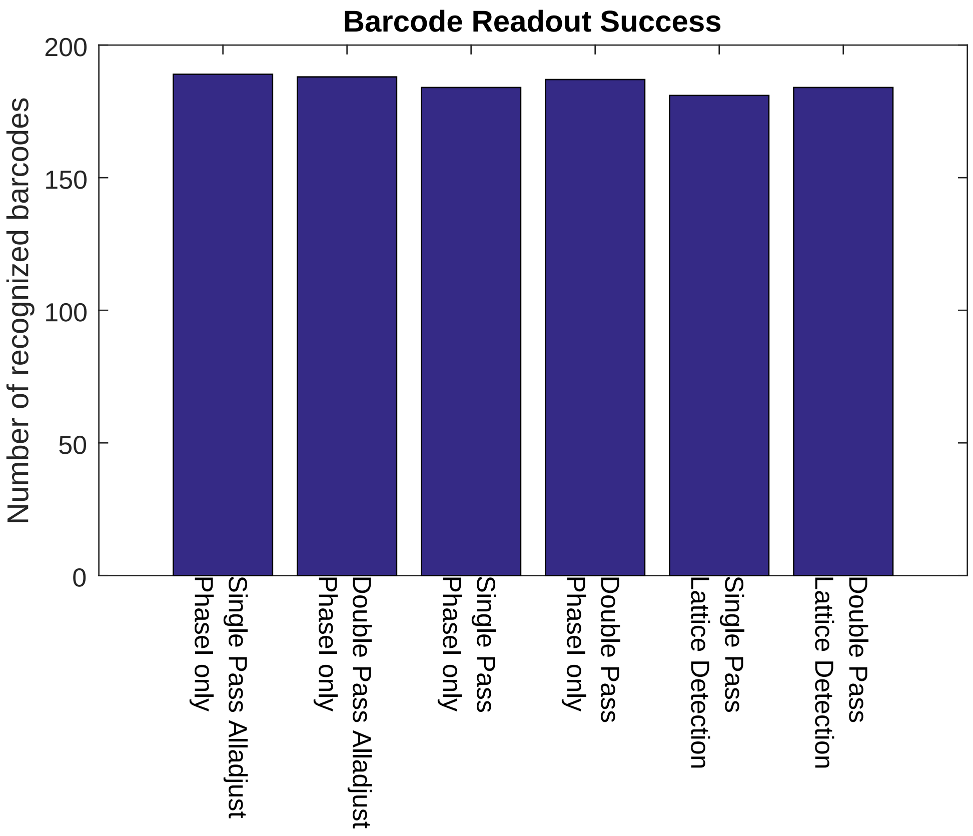
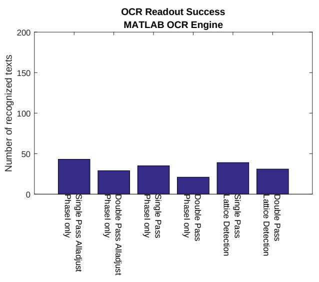
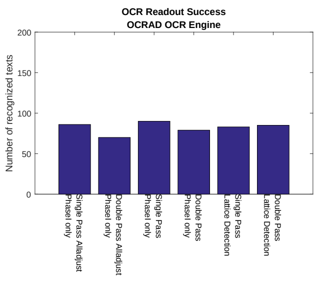


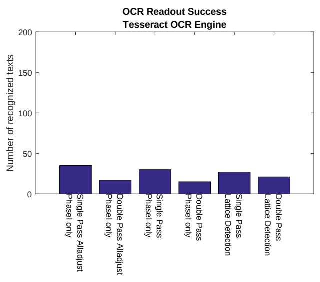
Figure 35



(a) Matlab



(b) OCRAD



(c) Tesseract

Figure 36

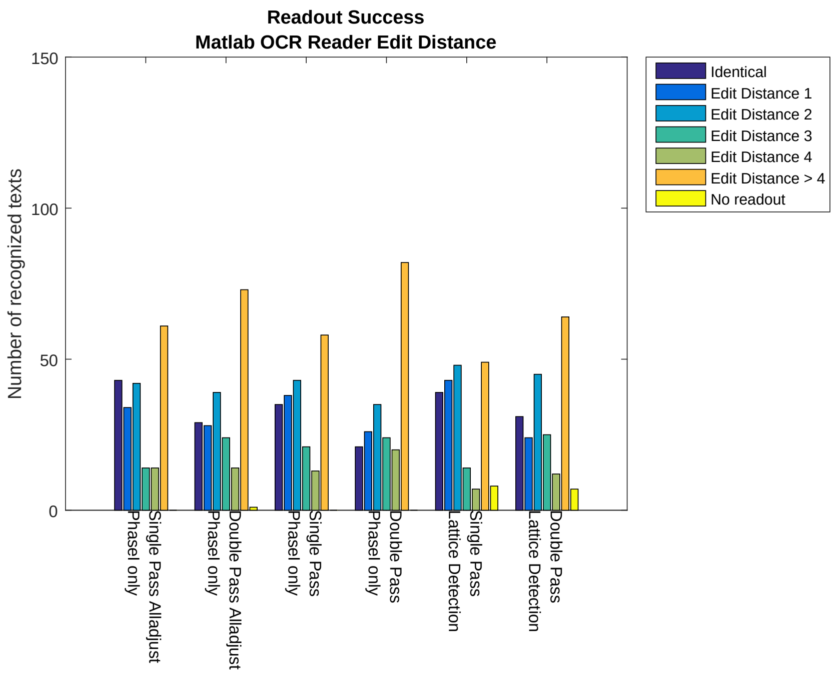


Figure 37

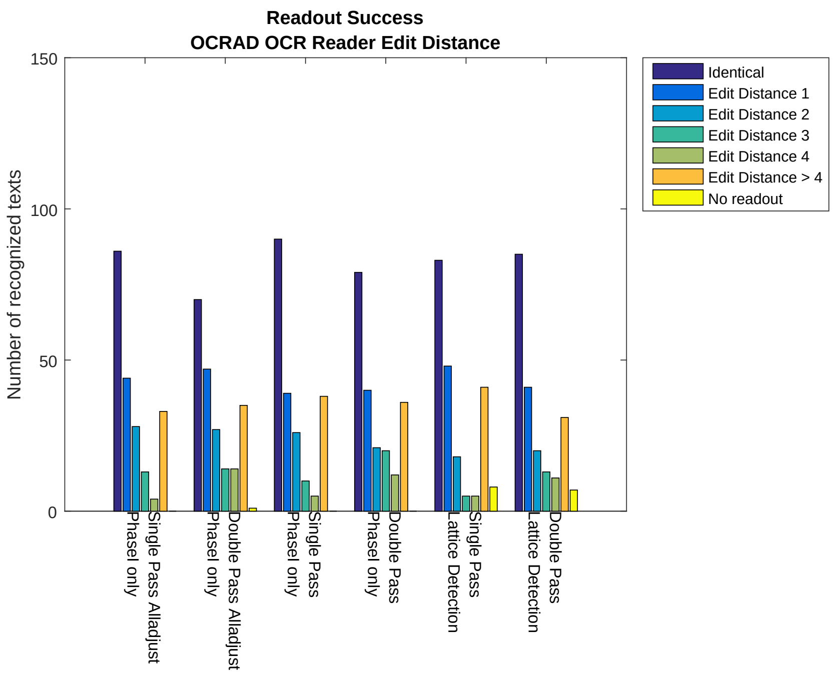


Figure 38

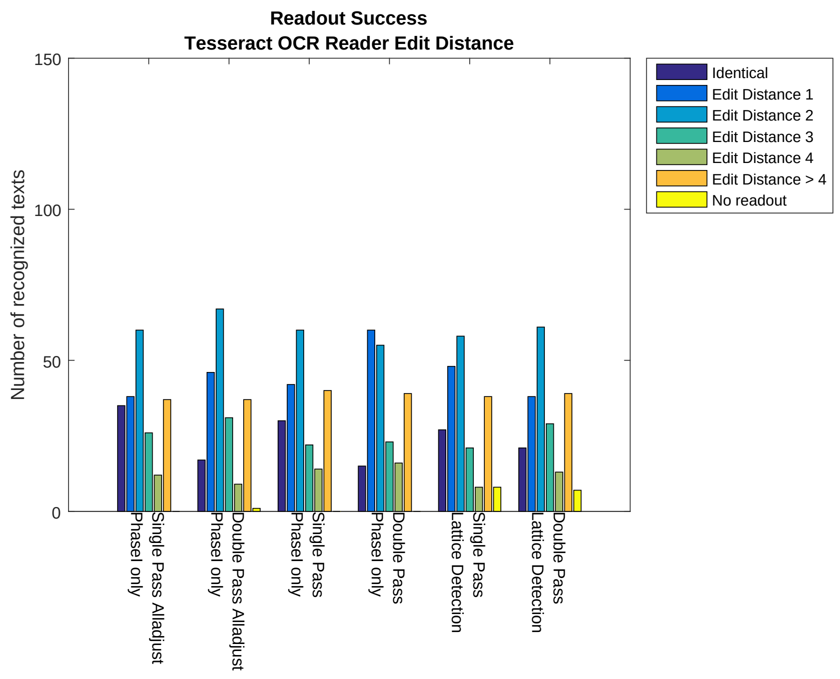
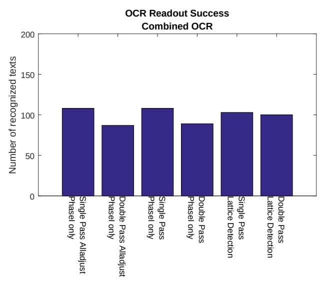
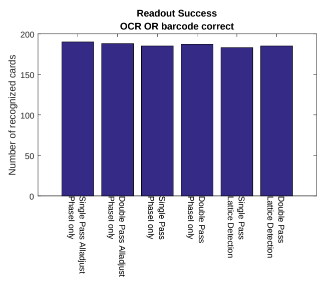


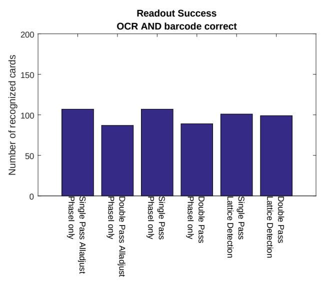
Figure 39



(a) All OCR Readers



(b) OCR OR Bar Code



(c) OCR AND Bar Code

Figure 40

4.6 Summary

We observe that the bar code readout is more robust than the OCR readout. Furthermore combining both results does not yield to a significant overall improvement of the total readout success, i.e. if the OCR reader is successful then the bar code readout has been successful as well. Furthermore, again combining different OCR readers does not lead to a significant improvement of OCR readout success. Given the OCR reader we observe that the OCRAD reader yields the best result without any post-processing. Looking at the edit distance of the OCRAD Readout that are not identical to the original input we observe that the OCRAD reader is more robust than Tesseract or Matlab: The Matlab OCR Reader edit distance of one and two are in the same order of magnitude than the identical ones. For Tesseract, the probability of having an edit distance of two or three is very high. We observe that the results for edgemap *Variant B* (the more blurry one) are generally more successful than the readout results originating from *Variant A*.

Moreover, if we focus on readout success all algorithm variants for edgemap *Variant B* outperform the baseline approach. Furthermore, for both edgemap variants the *Full Lattice Detection* algorithm does not lead to any improvements against the *Phase I only* variants. On the contrary: For edgemap *Variant A*, the algorihm's readout is far behind the *Phase I only* variants. This leads to the conclusion that the additional runtime spend to rectify and further developing the lattice is not paying off. That means if we only perform Phase I of the lattice dection algorithm, we obtain a runtime below 20s. Furthermore, this runtime is more deterministic and robust against outliers. Additionally, if we focus on runtime the the *Phase I only* variants for both edgemap variants outperform the baseline variants by far, while the *Full Lattice Detection* variant is borderline.

While from a runtime perspective the *Alladjust* variant is slightly slower than the *Only Correct Point Labeling* variant the extra effort does not guarantee a better readout success. As a matter of fact the results for edgemap variants *A* and *B* differ, so we cannot conclude if there is a significant difference or not. The same reasoning can be applied to the *Single Pass* vs *Double Pass* variants: While the bar code readout success is inconsistent, the OCR readers readout success show a tendency to *Double Pass* with edgemap *Variant A*, while the *Single Pass* variant is prefered by OCR readers in edgemap *Variant B*.

5 Future Work

In this section we propose further optimization and research directions that our previous work has paved the way for.

5.1 Preprocessing of OCR Readouts

The accuracy in all of the OCR methods used is very fluctuant and in general less accurate than the bar code extraction. However, in our evaluation we did not preprocess any extracted text-patch. Future versions might optimize the text readout e.g. by sharpening, thresholding before passing it to the

OCR readers to get a better OCR result. For example, the authors of the Tesseract OCR engine claim that image preprocessing can rapidly increases the detection rate of their OCR reader [21].

5.2 Heuristic for Pseudonym Readout

The current experimental results show that the automated readout of the bar code is very robust against and works more reliable than the OCR readout. However, in the event that the bar code is not or only partially readable, our current architecture has laid the foundation of future algorithms to compute the most probable result.

5.3 Consideration of Corner Points

Currently, we only consider the innermost points of the color reference card. However, future versions might also consider the corner points of the outer color patch in the color reference card. This might yield a better result in the calculation of the perspective transform. However, currently our point labeling of the corner points does not consider intersection points without direct color neighbors i.e. for the edge corner points an individual feature vector has to be computed.s

5.4 Performance Improvements for Point Labeling

For large amounts of points the complexity of the point labeling is quadratic and hence does not scale with larger numbers of points. While this problem is negligible when the full lattice detection algorithm is performed (the lattice intersection points are typically in the same order of magnitude than the real lattice points), this becomes a bottleneck when using uniform equidistant sampling or a larger number of candidate points.

6 Conclusion

In this paper we proposed an extension of the framework proposed by Jose el al. [3] to include the study subject's pseudonym onto a color checker reference card that is placed within the image to be included in the trial. We build a modular algorithm, which steps can be rearranged in different ways. We evaluated some of those arrangements. Furthermore, we came to the conclusion that our proposed fine adjustment as well as the point labeling of Jose et al. are so good that we do not need the run the complete lattice detection algorithm, since the results of Phase I of this algorithm does not only suffice but yields even better results than performing the whole algorithm. In addition, by only using Phase I we gain a significant performance gain of up to approximately 365% (mean) or 279% (median) for Edgemap *Variant A* and 315% (mean) or 520% (median) for Edgemap *Variant B*. Apart from runtime performance our *Phase-I only* variants all outperform or equal the result of the baseline variants. Our thoughts for future improvements may further optimize those results.

References

[1] Deserno TM, Sárándi I, Jose A, Haak D, Jonas S, Specht P, et al. Towards quantitative assessment of calciphylaxis. In: SPIE Medical Imaging. International Society for Optics and Photonics; 2014. p. 90353C–90353C.

[2] Jackowski M, Goshtasby A, Bines S, Roseman D, Yu C. Correcting the geometry and color of digital images. Pattern Analysis and Machine Intelligence, IEEE Transactions on. 1997;19(10):1152–1158.

[3] Jose A, Haak D, Jonas S, Brandenburg V, Deserno TM. Human wound photogrammetry with low-cost hardware based on automatic calibration of geometry and color. In: SPIE Medical Imaging. International Society for Optics and Photonics; 2015. p. 94143J–94143J.

[4] Deserno TM, Haak D, Brandenburg V, Deserno V, Classen C, Specht P. Integrated image data and medical record management for rare disease registries. A general framework and its instantiation to the German calciphylaxis registry. Journal of digital imaging. 2014;27(6):702–713.

[5] Haak D, Gehlen J, Jonas S, Deserno TM. OC ToGo: bed site image integration into OpenClinica with mobile devices; 2014. Available from: http://dx.doi.org/10.1117/12.2042847.

[6] Macduff: the Macbeth ColorChecker Finder;. Available from: https://github.com/ryanfb/macduff.

[7] Bianco S, Cusano C. Color target localization under varying illumination conditions. In: Computational Color Imaging. Springer; 2011. p. 245–255.

[8] Brunner RT, Hayward D. Automatic detection of calibration charts in images; 2011. US Patent 8,073,248.

[9] Ernst A, Papst A, Ruf T, Garbas JU. Check my chart: A robust color chart tracker for colorimetric camera calibration. In: Proceedings of the 6th International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications. ACM; 2013. p. 5.

[10] Kordecki A, Palus H. Automatic detection of colour charts in images. Przegląd Elektrotechniczny. 2014;90(9):197–202.

[11] Wang S, Minagawa A, Fan W, Sun J, Xu L. A Fast and Robust Multi-color Object Detection Method with Application to Color Chart Detection. In: PRICAI 2014: Trends in Artificial Intelligence. Springer; 2014. p. 345–356.

[12] Park M, Brocklehurst K, Collins RT, Liu Y. Deformed lattice detection in real-world images using mean-shift belief propagation. Pattern Analysis and Machine Intelligence, IEEE Transactions on. 2009;31(10):1804–1816.

[13] McCamy CS, Marcus H, Davidson J. A color-rendition chart. J App Photog Eng. 1976;2(3):95–99.

[14] Carayon P, Wetterneck TB, Hundt AS, Ozkaynak M, DeSilvey J, Ludwig B, et al. Evaluation of nurse interaction with bar code medication administration technology in the work environment. Journal of Patient Safety. 2007;3(1):34–42.

[15] Paoletti RD, Suess TM, Lesko MG, Feroli AA, Kennel JA, Mahler JM, et al. Using bar-code technology and medication observation methodology for safer medication administration. American journal of health-system pharmacy. 2007;64(5).

[16] Information technology – Automatic identification and data capture techniques – EAN/UPC bar code symbology specification. Geneva, Switzerland: International Organization for Standardization; 2007. 15420:2009.

[17] ZXing;. Available from: `https://github.com/zxing/zxing`.

[18] Teseract OCR Software;. Available from: `https://github.com/tesseract-ocr/tesseract/`.

[19] Ocrad – The GNU OCR;. Available from: https://www.gnu.org/software/ocrad/.

[20] Recognize Text Using Optical Character Recognition (OCR);. Available from: http://de.mathworks.com/help/vision/examples/recognize-text-using-optical-character-recognition-ocr.html.

[21] Improving the quality of the output;. Available from: https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality.