ASP: Arbitrary Segment Patterns

Transferring complex data from seven-segment displays using smartphone camera technology.

Master's thesis in Computer Science – algorithms, languages and logic

Filip Levenstam & Jean-Philippe Green

Department of Computer Science
Chalmers University of Technology
Gothenburg, Sweden 2015

Master's thesis 2015

ASP: Arbitrary Segment Patterns

Transferring complex data from seven-segment displays using smartphone camera technology.

FILIP LEVENSTAM
JEAN-PHILIPPE GREEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

ASP: Arbitrary Segment Patterns
Transferring complex data from seven-segment displays using smartphone camera technology.
FILIP LEVENSTAM
JEAN-PHILIPPE GREEN

Supervisor: Henrik Fagrell, Diadrom Holding AB
Supervisor: Fredrik Kahl, Signals and Systems
Examiner: Graham Kemp, Computer Science and Engineering

Master's Thesis 2015
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Generated picture of a seven-segment display showing a random code

Typeset in LaTeX
Gothenburg, Sweden 2015

Abstract

7-segment displays are often used in industry for displaying limited information about the current state. In this thesis we investigate the possibility to transfer more content rich data using our proposed *Arbitrary Segment Patterns* scanned by a smartphone. It is a concept of allowing the displays to use the segments and dots of the display as bits representing the message. Further more this includes displaying the messages in sequences allowing for even more information. Five different algorithms for interpreting the patterns are evaluated where three are based on line detections and two on shape detection. They are analyzed from the aspects of both performance and correctness using 7-segment displays of the types LED with red diodes and reflective LCD. The result is that the most promising approach is to use shape detection allowing for sending patterns with a speed of more than two patterns each second. This frame rate allows for sending 70 bits of information during a period of 5 seconds on a machine with displays of two digits. This is a significant increase of information.

Keywords: Computer vision, Image analysis, Seven-segment, Display, Bar-code, Diagnostics, Data, Communication

S

Acknowledgements

We want to thank the company Diadrom which supported us by offering us access to the hardware required for this thesis along with a workplace. A special thanks to the supervisor, Henrik Fagrell, for aid given throughout the project.
We also want to thank Fredrik Kahl for agreeing to become the supervisor from Chalmers university of technology and for help given in the area of image analysis.
A last thank to *ESAB* for offering us a field study for studying machines of interest for this thesis.

Jean-Philippe Green
Filip Levenstam
Gothenburg, December 2015

Contents

List of Figures

List of Tables

List of Definitions

ASP
: Arbitrary Segment Patterns, the technology presented in this thesis for transferring information from a seven segment display to a smartphone.

ASP-bit
: One bar or dot in a seven-segment unit.

ASP-pattern
: A possible state for a seven-segment display.

ASP-display
: A display which follows the proposed ASP protocol

Calibration bit
: An ASP-bit reserved for calibration rather than representing any information. These are always lit.

Sequence bit
: An ASP-bit reserved for sequence number rather than the actual data.

Seven-segment unit
: 7 bars, and one dot. Each of which can be either on or off. These segments are ordered as an 8 with a dot.

Protocol
: A set of rules defining how two entities may communicate.

Run time
: The total time a machine has been running.

Seven-segment display
: A display device capable of showing seven-segment units.

1 Introduction

This chapter gives a brief overview of the area of the project, further explaining general concepts along with the purpose and delimitations.

1.1 Background

Seven-segment displays are widely used within a variety of products, such as alarm clocks, microwave ovens, and automated drills. The technology is limited compared to other technologies based on pixel graphics, for example dot matrices, or video displays. On the other hand, the technology is cheap and flexible enough for representing some characters, especially digits. However, it is sometimes used for representing the hexadecimal characters but it then requires a use of both upper and lower case characters since for example it otherwise would be impossible to distinguish an uppercase *B* from an *8*, see figure 1.1.

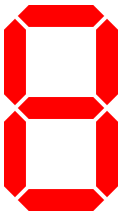


Figure 1.1: How both uppercase B and 8 is represented using a seven-segment character.

However, there are several reasons for seven segment displays to still be used in many products. It is a cheap technology and the interesting information during normal working condition can often be represented using digits only. However, in some cases, a user would benefit from the possibility of being able to extract more sophisticated information. For example, as an error appears a user may be interested in data such as the status of the machine. In this case the seven-segment display becomes a bottleneck, while the machine does have the interesting information it cannot transfer this information in an easy manner to the user.

1.2 Segment displays

Each digit in a 7-segment pattern consists of 7 bars ordered as an 8, see figure 1.1. There is also, in general, a dot belonging to each unit. However, through personal

observations we conclude that the dot sometimes is missing from the last digit of each row, this is particularly true for displays based on LCD-technology. The number of different patterns possible to represent, if all units have a dot, is thereby $2^{8\times d}$ where d is the number of digits. If however the last units in each row is missing the dot the number of combinations are given by $2^{r\times(8\times c-1)}$ where r is the number of rows and c the number of columns. The number of patterns possible to represent using one digit excluding the dot is therefore $2^7 = 128$, see figure 1.2. As seen in figure 1.3, there are also other variations of segment displays such as the 9-segment, 14-segment, and 16-segments. While these allow for even more possible states, they are not as common as the tradition 7-segment display.

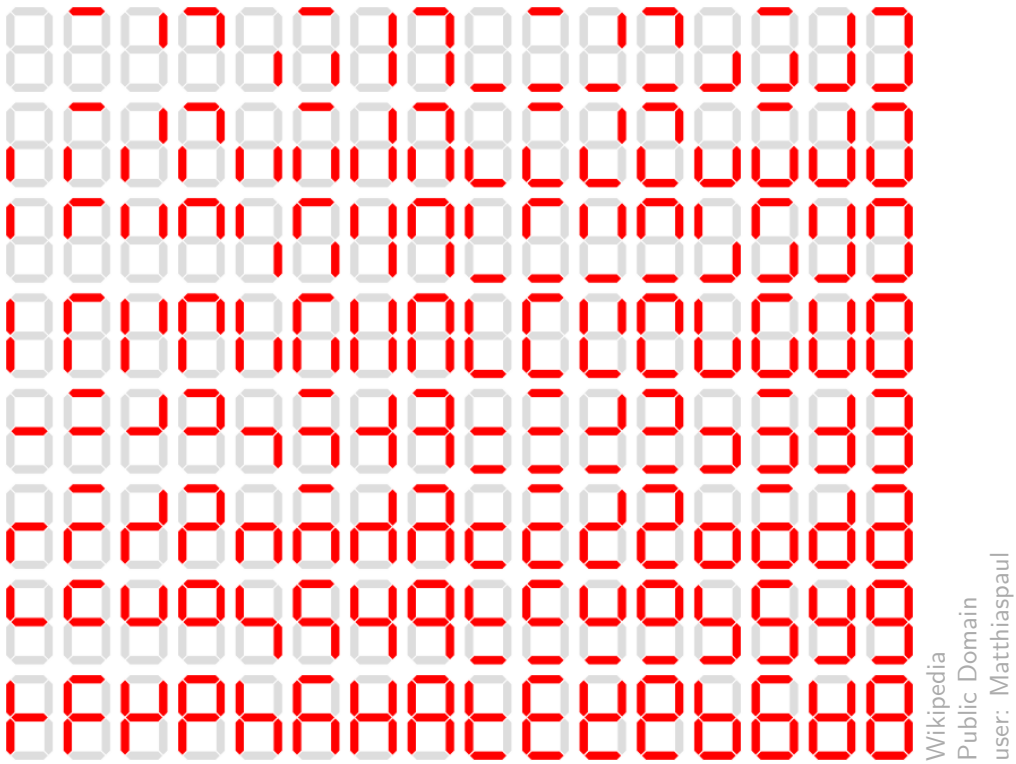


Figure 1.2: The different states the seven bars of a unit may represent. [1]

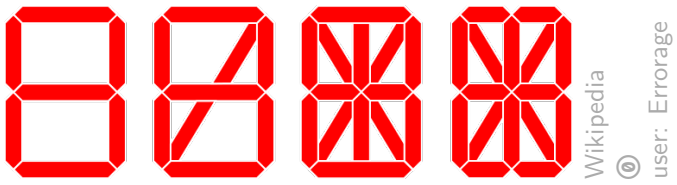


Figure 1.3: Different kinds of segments displays. From the right the 7-segment, 9-segment, 14-segment, and 16-segment display. [2]

[1] Url: `https://commons.wikimedia.org/wiki/File:7-segment.svg`, [Online; accessed 30-November-2015]

[2] Url: `https://commons.wikimedia.org/wiki/File:Common_segment_displays.svg`, [Online; accessed 30-November-2015]

1.3 Purpose

Our aim with this master thesis is to investigate how one could transfer more information from a machine to a user without adding additional components to the machine. More specifically, we want investigate the possibilities of introducing a smartphone to interpret more complex seven-segment patterns using the smartphone's camera as only input. Hence, within this project we assume 7-segment displays where it is possible to control each segment individually.

1.4 Contribution

Previous work has been done in the area of barcode scanning and optical character recognition of seven segment displays, both explained in chapter 2. Our contribution is to merge these two concepts, in order to allow usage of seven segment displays to transmit more than just hexadecimal characters. We also introduce the concept of dynamic sequencing, which allows an increase of the data throughput in one-way communications and can thus be used in a broader context.

1.5 Problem formulation

The problem this master thesis aims to answer is the following:
How can one maximize the data throughput, read optically, from a 7-segment display to a modern smartphone as conveniently while not constraining the environmental settings?
That is, a user should be able to scan seven segment patterns using only a smartphone camera during normal working conditions in a as convenient way as possible. By normal working conditions and convenience, we mean the following:

- **Display type independence:** There should be few constraints on which kind of displays are supported, including both LCD- and LED-displays of different colors.
- **Noise insensitivity:** The solution should allow a large amount of background noise, meaning that the display should be readable regardless of the environment. This include support for different lighting conditions.
- **Rotation, scale and perspective invariance:** The user should be able to hold the smartphone with any rotation. It should be possible to scan from different point of views, given a reasonable distance and angle. The camera properties, such as its maximum resolution and angle of view, should have minimal impact.

1.6 Delimitations

While the project aims to transfer information from a machine to a smartphone, we will not set up any in-use machine for transferring this information. Instead a lab environment will be created for this purpose.

2

Research area and related work

The problem in this thesis relates mainly to computer vision[1], which itself touches many other areas, such as image analysis[2], image processing[3], and pattern recognition[4]. Some methods of computer vision focus on extracting information from codes, while others on recognizing more complex objects such as human faces. While this project aims to extract information from codes, there might be a need to use technichs to extract more complex objects due to the different colors and shapes different seven-segment displays might have.

2.1 Barcodes

A barcode is a representation of data created for interpretation by a machine with some kind of optical scanner, such as a camera. Common methods include 1-dimensional barcodes, such as EAN-8[5], which are widely used for identifying products in stores.
However, since these are 1-dimensional, the amount of data that can be represented is highly limited. With the introduction of smartphones that include high definition cameras, 2-dimensional barcodes with more data capacity have become more common in order to allow users to quickly access information. QR-codes[6] is one such representation that has become popular, probably because of its quickly-readable and error-proof design.
In addition to 2-dimensional barcodes, there also exist 3-dimensional bar codes, where the third dimension is represented by different colors. Langlotz et al. goes further than that, and propose a 4-dimensional barcode, where the fourth dimension is represented by time.[7] Instead of simply using one static barcode, an animated GIF showing different sequences of 3-dimensional barcodes loops on a display, giving the possibility to transmit more data without requiring higher camera resolution. Similar techniques are considered within this thesis.

2.2 OCR of seven-segment displays

More specific work for our problem has been done in the area of optical character recognition (OCR)[8] on seven-segment displays to read hexadeximal digits[9, 10, 11, 12]. One project of special interest is the open source software *Seven Segment Optical Character recognition* (SSOCR) [13], as it also has working code. However, this software differs quite a lot from what we want to achieve, as it does not handle noise in the background very well. Even if it is possible to manually crop and rotate

the image in SSOCR, it cannot find the display automatically. This is a feature which is important when using a smartphone.
Another major drawback of SSOCR is that the algorithm is specifically designed for reading digits from a displays with only one row. It will first segment the digits by traversing from left to right, and then analyze which segments are lit in each digit by looking at very few pixels. Using this algorithm, a seven segment unit with no active segments would be ignored. However, in our case, detection of inactive units is needed, because only zeros in a unit may represent important data too.

3

Optics

This section explains important fundamentals in the area of optics needed for this project. Since this project's aim is to process optically read displays, it is important to understand how optics work and how smartphone cameras interpret different lights.

3.1 Pixel

In electronics, images are usually represented by a two dimensional array of pixels. Each pixel may have different representations depending on the color model. In this section, different, common pixel representations and color models will be presented.

3.1.1 RGB

In the RGB model, each pixel is represented by three channels: red, green and blue. This model is widely used in screens and electronics since the human eye has three different types of cone cells, each one of them being specifically sensitive to one of these colors.
RGB is an additive model, meaning that combining these colors results in a white light, and using none results in the color black. Nearly all colors that can be perceived by the human eye can be represented in RGB. When storing RGB data digitally, the number of bits used to represent each pixel is called the color depth.

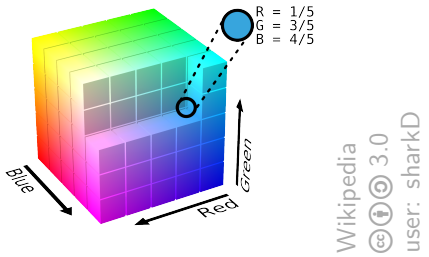


Figure 3.1: A box representation of an RGB color space. [1]

[1] Url: `https://commons.wikimedia.org/wiki/File:RGB_Cube_Show_lowgamma_cutout_b.png`, [Online; accessed 30-November-2015]

3.1.2 Raw

An image which contains unprocessed data from the image sensor is said to use the RAW-format. The exact representation of this may differ from one device to another, depending on the filter in front of the image sensor. A common filter is the Bayer filter which is an RGB-filter, where out of four pixels, one is used for red, one for blue, and two for green as shown in figure 3.2.

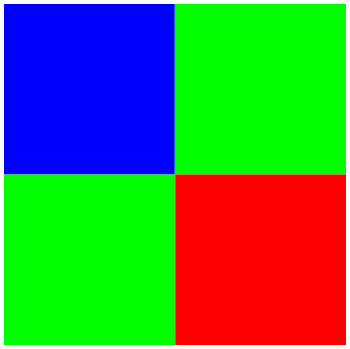


Figure 3.2: The check pattern used by the Bayer filter

3.2 Properties of the camera

There are different types of cameras, for example there are both analog and digital cameras. This project focuses on digital cameras which in turn can be divided into cameras based on the technology of complementary metal oxide semiconductor (CMOS) respectively charge-coupled device (CCD). There are advantages of both technologies, while the CMOS-sensor are cheaper the CCD can offer higher quality photos. However, recent years development of the CMOS-technology has significantly increased the possibilities of taking high quality photos using CMOS-sensor which is why their market share has greatly increased. Beside the cost, CMOS has other advantages such as the speed. A higher frame per second (FPS) allows for video capturing but also for features such as auto-focus. These advantages has led to an decay in the market share for the CCD-sensor which in recent years mainly is used in the Premium models where very high quality but also manual control over focus is desired. Another drawback with the CMOS technology is the phenomenon of rolling shutter. That is, if a camera takes a photo of something that is rapidly changing, the CMOS based cameras, because it has no mechanical shutter, can take photos where not all pixels are captured in an instant, see figure 3.3.

3.2.1 Exposure

There are many variables that affect the exposure of the camera, for instance the lens aperture and the shutter speed. In digital cameras, a signal gain on the sensor may directly affect the exposure value. Digital cameras come with a decided exposure index (EI) rating (also called ISO-setting) that relates the produced sRGB image files to what would be produced with an analog camera film.

Figure 3.3: Photo of a helicopter where the phenomenon of rolling shutter is seen. [1]

3.2.2 Automatic exposure

Many systems offer the usage of automatic exposure (AE) that calculates and adjusts exposure settings in a way that makes the overall exposure acceptable. It usually looks at the image mid-tone and changes the exposure settings thereafter.

3.2.3 Exposure compensation

Sometimes, the user may want to bias the AE system in one way or another, such that it is either more or less sensitive to light. The reason for this is usually that AE-systems may misbehave in certain conditions. For example, when there is much light from one particular area in the image, AE systems tend to dim the whole image. This may not be the intended behavior, as it may dim more important parts of the image.

3.2.4 Overexposure and underexposure

A photo can be described as overexposed if important bright details of the photo are washed out, or even completely white. This happens when too much light is captured. The opposite to this is when a photo is underexposed, that is too little light is captured. Instead this causes shadows to darken and sometimes being completely black.

[1] Url: `https://commons.wikimedia.org/wiki/File:Jamtlands_Flyg_EC120B_Colibri.JPG`, [Online; accessed 30-November-2015]

3.3 Properties of displays

It is important that the product should be able to read displays with different properties. There are different kinds of displays used within the industry for representing seven segment characters. This section explains the technology behind the LCD and LED displays.

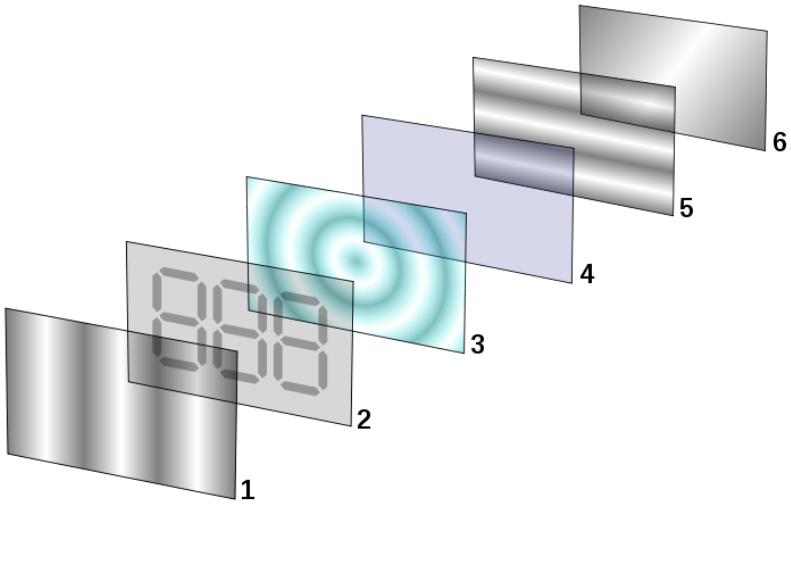


Figure 3.4: Illustration of the different layers within a LCD-display. [1]

1. Polarizing filter to polarize entering light.
2. Glass with positive electrodes formed as seven-segment digits.
3. Twisted nematic liquid crystal.
4. Glass with negative electrode film, used for establishing the electrical field to layer 2.
5. Polarizing filter to either block or pass light.
6. Reflective surface to reflect light or the light source itself.

3.3.1 LCD-display

Liquid crystal displays (LCD) are based on the technology of using a matter within the state of being a liquid crystal, a phase between the solid and the liquid state. More specifically the substance also is nematic, meaning that the molecules have no positional order regarding each other but still tend to point in the same direction. The particular sort of matter used for LCD-displays are the twisted nematics. It is called so because of the property that the molecules are naturally twisted and therefore have the property that the polarization of any light passing through the

[1] Url: `https://commons.wikimedia.org/wiki/File:LCD_layers.svg`, [Online; accessed 30-November-2015]

matter will be shifted. However, as the liquid crystals are under the influence of an electrical field, the molecules are untwisted and the shifting of the polarization is reduced. Therefore, by sending polarized light through the liquid crystals, one can regulate the outgoing polarization of the light by using an electrical field. This allows for filtering the outgoing light so that only light of a certain polarization will pass through. However, the liquid crystals cannot be used for modifying or controlling the wave length of the light. Thus, it is impossible to change any colors using the properties of the liquid crystals. Instead, it is common to divide each pixel into three components, each with a filter of the colors; green, blue, or red to allow for colored screens. An LCD-display may have a backlight as light source but can also depend on the light within the room for displaying the content. The display is not a Lambertian surface meaning that the angle from where the display is seen from influences on what is observed.[14]. Figure 3.4 explains the different layers within the LCD model further.

3.3.2 LED-display

Light-emitting diodes are displays that use diodes to emit light. Diodes are the simplest forms of semiconductors, and allows current to traverse in only one direction [15].
To make this work, a bad conductor material - usually aluminum-gallium-arsenide - is used which is then *doped*. Doping is done to make a material more conductive, and can be achieved in two ways: *a*) adding free electrons (resulting in an N-type material); and *b*) creating holes where electrons can go (resulting in a P-type material). Diodes use both methods of doping; one on each side of the conductor. This results in the interesting property of only being able to pass current in one direction. When current flows through a diode, electrons drop from a higher orbital to a lower one, resulting in released energy in form of photons, thus emitting light. However, all light is not visible to the human eye, so if the light should be seen by humans, the drop must happen from a distance that releases the amount of energy needed to be in the human visible spectrum. This is the case of LED-displays.

4

Image analysis and computer vision

This section explains attributes of images and relevant algorithms for image analysis and computer vision.

4.1 OpenCV

OpenCV is an open source software library originally developed by Intel. It is written in the programming languages *C++* and *C* but has wrappers for working in other languages such as *Java* or *Python*. It provides tools for image and video processing, machine learning, and computer vision. [16]
Images are represented using 2 dimensional matrices where each cell may contain an arbitrary number of channels and the values can be represented using signed integers, unsigned integers or floating numbers of different sizes. Besides providing a representation of matrices it also provides several common operations and transformations on these.

4.2 Pinhole camera model

In computer vision, the camera is usually modeled after the so called pinhole camera model [17]. This model is a mathematical description of the relationship between objects in 3D space and the projected image on a plane. However, it ignores the common effect of lens distortion since its effects are usually negligible and requires much more mathematical effort. Instead, the focus lies on how light works when passing an ideal pinhole camera.
A pinhole camera is ideally a light-proof box with an infinitely small aperture called a pinhole, which, unlike modern cameras, does not have a lens. When light goes through the hole, it is projected on a plane, and the image is rotated 180°.
We can model the pinhole as lying at the origin in a three-dimensional space with the axes (X, Y, Z). Z points outwards from the camera towards the outer world. The image is projected on an image plane which is placed at $(x, y, -f) \forall x, y$, where f is called the focal length.
Using this, it is now possible to calculate how a point $\mathbf{P}$ with coordinates (x_p, y_p, z_p) in the 3D world is projected on the image plane on a point $\mathbf{Q}$ with coordinates $(x_q, y_q, -f)$, where f is already given.

By ignoring the X axis and using the knowledge of triangle similarity as illustrated in figure 4.1, we see that the following holds:

$$\frac{-y_q}{f} = \frac{y_p}{z_p} \Rightarrow y_q = \frac{-f}{z_p} y_p.$$

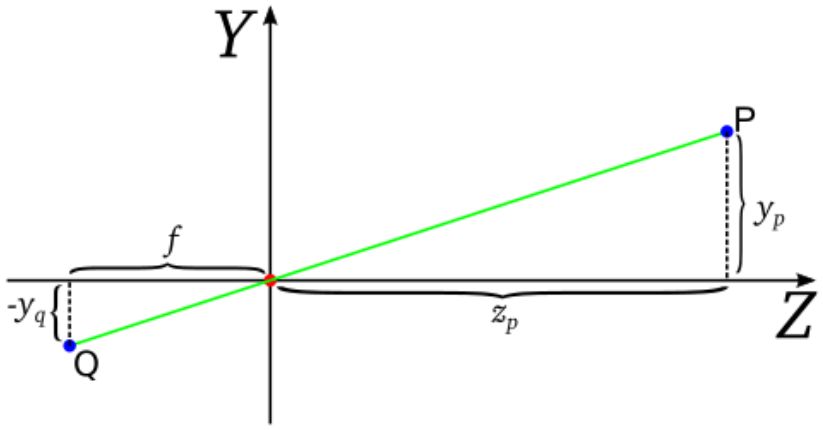


Figure 4.1: Example of a pinhole projection when looking at the direction of X

In the same manner, by instead ignoring the Y axis, we can also get the following:

$$\frac{-x_q}{f} = \frac{x_p}{z_p} \Rightarrow x_q = \frac{-f}{z_p} x_p.$$

Thus, we have the following:

$$\begin{pmatrix} x_q \\ y_q \end{pmatrix} = \frac{-f}{z_p} \begin{pmatrix} x_p \\ y_p \end{pmatrix}.$$

Now, the image is rotated 180°. In order to adjust the image, we simply negate the expression, resulting in the following expression:

$$\begin{pmatrix} x_q \\ y_q \end{pmatrix} = \frac{f}{z_p} \begin{pmatrix} x_p \\ y_p \end{pmatrix}.$$

For reasons that will later be obvious, we will instead use homogeneous coordinates in our expression. This is done by adding a third dimension which will be static and introducing the notion of equality up to scaling, denoted as $\sim$, in the following manner:

$$\begin{pmatrix} x_q \\ y_q \\ 1 \end{pmatrix} = \frac{f}{z_p} \begin{pmatrix} x_p \\ y_p \\ \frac{z_p}{f} \end{pmatrix} \sim \begin{pmatrix} x_p \\ y_p \\ \frac{z_p}{f} \end{pmatrix}.$$

The symbol $\sim$ is thus used to signal that any scalar is ignored. Similarly, by expressing the 3D coordinates in homogeneous coordinates, we get

$$\begin{pmatrix} x_q \\ y_q \\ 1 \end{pmatrix} \sim \begin{pmatrix} x_p \\ y_p \\ \frac{z_p}{f} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} \sim \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}.$$

Thus, we have

$$\mathbf{q} \sim \mathbf{C}\mathbf{p}$$

where $\mathbf{q} = \begin{pmatrix} x_q \\ y_q \\ 1 \end{pmatrix}$, $\mathbf{C} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$, and $\mathbf{p} = \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}$.

$\mathbf{C}$ is called the camera matrix, or the camera parameters. In our case, we assumed that the pinhole was positioned at the origin facing in the direction of the Z-axis, which leads us to this rather simple model of $\mathbf{C}$. If we however take more variables into account, such as the position and rotation of the camera axes in the real world coordinate space, as well as the position and skew of the projected image, we get a more complicated matrix. This matrix may be decomposed into two matrices, such that $\mathbf{C} = \mathbf{A}[\mathbf{R}|\mathbf{t}]$, in the following manner:

$\mathbf{A} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$, $[\mathbf{R}|\mathbf{t}] = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{11} & r_{32} & r_{33} & t_3 \end{pmatrix}$.

$\mathbf{R}$ is the rotation matrix, which describes the rotation of the camera, and $\mathbf{t}$ is the transformation vector which describes the position of the camera. (f_x, f_y) are the focal lengths in proportion to the pixel density and (c_x, c_y) are the image center coordinates.

4.3 Homography

Using the pinhole camera model, a homography is the relationship of a planar surface projected using two different camera parameters. This implies that the knowledge of a homography gives the possibility to redraw the image as if it was taken from another angle.[18]

A homography is usually represented as a 3x3 projection matrix, $\mathbf{H}$, describing how to translate points of an image taken from one perspective, to another. Thus, if we have a homogeneous image point of a plane from one viewpoint, $\mathbf{a} = (a_x\ a_y\ 1)^\mathbf{T}$, it can be translated to a homogeneous point $\mathbf{b} = (b_x\ b_y\ 1)^\mathbf{T}$ in another viewpoint if its homography $\mathbf{H_{ab}}$ is known, using the following formula:

$$\mathbf{b} \sim \mathbf{H_{ab}}\mathbf{a}.$$

Which may be written out as

$$\begin{pmatrix} b_x \\ b_y \\ 1 \end{pmatrix} \sim \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} a_x \\ a_y \\ 1 \end{pmatrix}.$$

Thus we have the following equations:

$$b_x = \frac{b_1}{b_3},$$

$$b_y = \frac{b_2}{b_3}$$

where

$$b_1 = h_{11}a_x + h_{12}a_y + h_{13},$$
$$b_2 = h_{21}a_x + h_{22}a_y + h_{23},$$
$$b_3 = h_{31}a_x + h_{32}a_y + h_{33}.$$

In order estimate the eight degrees of freedom of H_{ab} eight such equations are required. Thus, we need four corresponding points to calculate H_{ab}. [17]

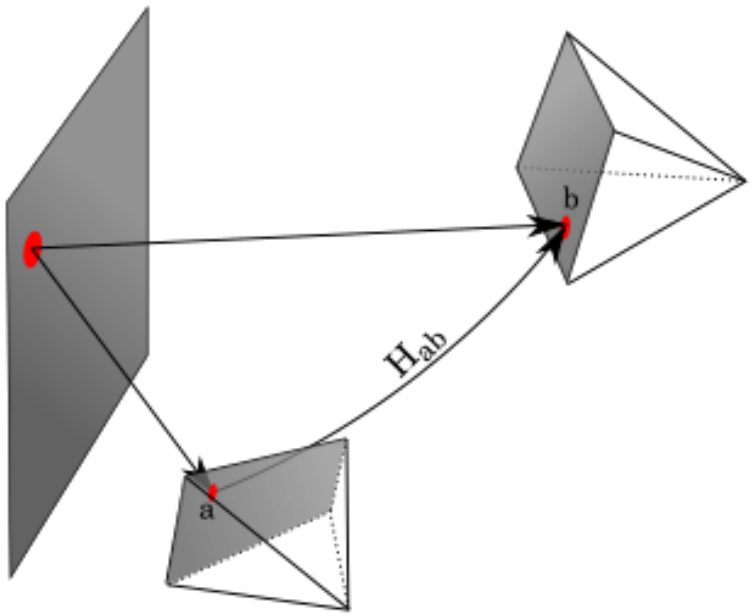


Figure 4.2: A point on a plane being projected on two different cameras can be translated using homography

4.4 Canny and Deriche edge detector

According to John F. Canny these are the requirements on an optimal edge detection algorithm. [19]

- Detection Quality - All existing edges should be marked and all marked edges should be correct.
- Accuracy - A marked edge should be as close as possible to the real edge.

- Unambiguity - An actual edge should only be marked once.

Both Canny and Deriche Edge detectors are designed to fulfil these requirements. The algorithms are similar to each other and can be divided into the same five steps.[19, 20]

4.4.1 Smoothing of image

The first step is to make the image smoother and also remove some noise. The Canny Edge detector uses a Gaussian filter for this purpose. The equation for a Gaussian kernel of size $(2k+1)*(2k+1)$ is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} exp(-\frac{(i-k-1)^2+(j-k-1)^2}{2\sigma^2}).$$

It is important to notice that the size of the kernel will be of significance for performance. Choosing a large kernel slows down the algorithm and decreases the sensitivity to noise.
However, instead the Deriche edge detector uses an infinite impulse response(IIR)-filter of the form:

$$f(x) = \frac{S}{\omega} exp(-\alpha|x|)sin(\omega x).$$

The filter is most effective as ω approaches 0 resulting in the form:

$$f(x) = S\ exp(-\alpha|x|).$$

A higher α results in better localization but worse detection rate for the edge.

4.4.2 Finding intensity gradient

An edge detection operator is used for finding the first derivative in the horizontal G_x and vertical G_y direction. These values can be used for calculating the gradient, the edge strength, and direction of the pixels.

$$G = \sqrt{G_x^2 + G_y^2},$$

$$\Theta = \arctan 2(G_y, G_x).$$

The direction of the gradient is then rounded to either 0°, 45°, 90°, or 135°. There are several edge detection operators that can be used for finding these derivatives. Here are the Sobel, Prewitt and Roberts operators explained, see [21].

4.4.2.1 Roberts

The algorithm was defined to satisfy the following properties; the resulting lines should be well-defined, there should be as little noise in the background as possible, and the intensity of the edges should correspond to what a human perceives. The following equations was proposed by Roberts:

$$y_{i,j} = \sqrt{x_{i,j}},$$

$$z_{i,j} = \sqrt{(y_{i,j} - y_{i+1,j+1})^2 + ((y_{i+1,j} - y_{i,j+1})^2}.$$

where x is the initial intensity value z is the resulting derivative. Thus we have G_x and G_y using

$$G_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} * I,$$

$$G_y = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} * I$$

where I is the source image.

4.4.2.2 Prewitt and Sobel

Both the Prewitt and Sobel operators are used to find the gradient of the image intensity. That is, the direction of the largest possible increase from light to dark how fast it changes. These values are used to determine how suddenly the image changes and thereby how likely it is that the current position is an edge and in that case the direction of the edge. The difference between the Prewitt and Sobel operators is that different matrices are used when calculating the resulting gradient. The following matrices are used for Prewit:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

And these are the matrices for Sobel:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

Hence, we have $G_x = K_x * I$ and $G_y = K_y * I$ where I is the source image.

4.4.3 Non-maximum suppression

The extracted edges are still blurry so to make the edges thinner the technique of non-maximum suppression is used. It compares the strength of an edge pixel with the pixels on both sides in the gradient direction. If the strength of the current pixel is lower than a compared pixel it is suppressed, otherwise it is kept.

4.4.4 Double threshold

While the edges of the image should be quite accurate there may still be some falsely identified edges caused by noise. These are removed by first first defining two threshold values, the high threshold and the low threshold. Each edge pixel is then compared to these values and if it is stronger than the high threshold it is considered a strong edge. If the pixel strength is in between the two threshold values it is considered a weak edge. If the strength is below the low threshold it is removed as an edge pixel.

4.4.5 Suppress weak edges

The final step in the algorithm is to suppress all weak edges that still exist because of noise. This is done by comparing all weak edge pixels with the neighboring pixels. If any neighbor is considered a strong pixel it will be considered part of an edge, otherwise it will be removed.

4.5 Threshold

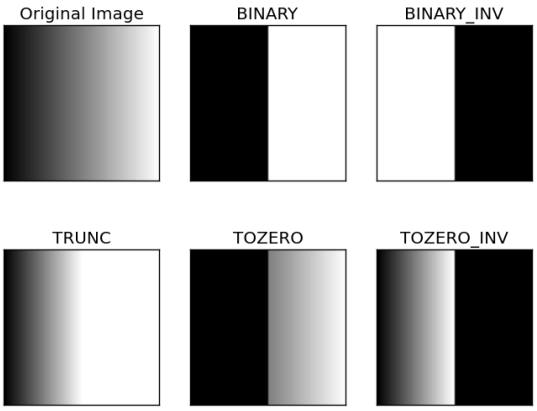


Figure 4.3: The effect of applying threshold using the different operations in OpenCV.

Applying threshold to an single channel image returns an image where any pixel is compared to a given threshold value. Depending on if the pixel value is higher or lower than the threshold, different functions are applied to the pixel. The threshold algorithm existing in OpenCV, see section 4.1, takes a source and destination image, a threshold value, a max value, and a threshold operation. The result of the different operations can be seen in figure 4.3, and the equations for them are as follows.

Binary threshold

$$\text{dst}(x,y) = \begin{cases} \text{maxVal} & \text{if src}(x,y) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Inverted binary threshold

$$\text{dst}(x,y) = \begin{cases} 0 & \text{if src}(x,y) > \text{threshold} \\ \text{maxVal} & \text{otherwise} \end{cases}$$

Truncate

$$\text{dst}(x,y) = \begin{cases} \text{maxVal} & \text{if src}(x,y) > \text{threshold} \\ \frac{\text{src(x,y)} \times \text{maxVal}}{\text{threshold}} & \text{otherwise} \end{cases}$$

Threshold to zero

$$\text{dst}(x,y) = \begin{cases} \text{src}(x,y) & \text{if } \text{src}(x,y) > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Inverted threshold to zero

$$\text{dst}(x,y) = \begin{cases} 0 & \text{if } \text{src}(x,y) > \text{threshold} \\ \text{src}(x,y) & \text{otherwise} \end{cases}$$

4.6 Adaptive threshold

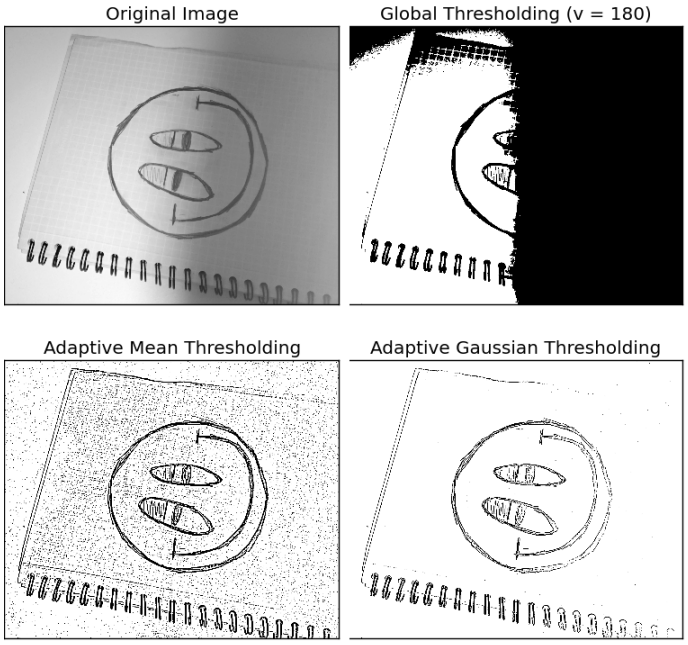


Figure 4.4: A comparison of the different threshold algorithms.

Adaptive threshold is a threshold algorithm which depends on the neighboring pixels for calculating the result for each pixel. This makes it especially useful on images where some regions of an image are in a shadow while other areas are not. In addition, it can be used as an edge detector since it is sensitive for derivatives of pixel values. The OpenCV, see section 4.1, implementation of adaptive threshold requires, apart from the source and destination image; a block size, the size of the area to consider around each pixel, an adaptive method and a constant which is the value to be subtracted from, depending on the adaptive method, the mean or weighted sum calculated. The adaptive method decides whether the algorithm should use a mean or a weighted sum of the neighboring pixels where the weights are a Gaussian window. The results of the different adaptive threshold, along with a simple threshold, can be seen in figure 4.4.

4.7 Finding contours

One can get information about the topological structure using border following algorithms as proposed by Satoshi Suzuki et al [22]. This gives as a result a tree of contours with knowledge about the parents and the children of each contour. As seen in figure 4.5, a contour is a parent if it surrounds other contours, the children. A second algorithm is also proposed which only yields the contours of the outermost borders.

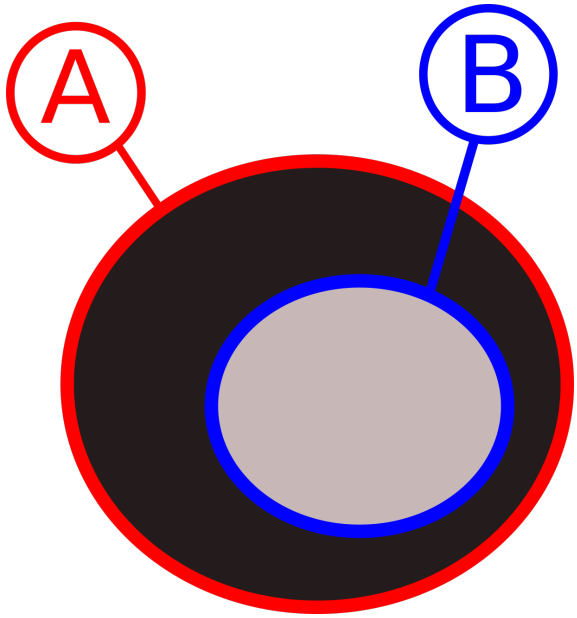


Figure 4.5: The contour B is contour A's child.

4.8 Image moments

Image moments are values that calculated by some function of the weighted averages of pixel intensities, meaning that they can express some general properties of the image. Raw image moments M_{ij} of a grayscale image are defined by the following formula

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y).$$

Where I(x,y) is the intensity of the pixel located at (x,y). Using this, we can calculate the sum of gray level using M_{00} and the image's centroid as $(\bar{x}, \bar{y}) = (M_{10}/M_{00}, M_{01}/M_{00})$.

4.8.1 Scale and translation invariant moments

Image moments that are invariant to both translation and scale can be defined using the following formula

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{(1+(i+j)/2)}}, i+j \geq 2$$

where the central moments, μ_{ij} are defined by

$$\mu_{ij} = \sum_x \sum_y (x-\bar{x})^i (y-\bar{y})^j I(x,y).$$

4.8.2 Rotation invariant moments

A commonly used set of rotation invariant moments were proposed by Ming-Kuei Hu in 1962 [23].

$$\begin{aligned}
I_1 =& \eta_{20} + \eta_{02}, \\
I_2 =& (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2, \\
I_3 =& (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2, \\
I_4 =& (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2, \\
I_5 =& (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
& (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2], \\
I_6 =& (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}), \\
I_7 =& (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - \\
& (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].
\end{aligned}$$

These are also scale and translation invariant, as well as the last one being skew invariant.

4.8.3 Comparing rotation invariant moments

The rotation invariant moments can be used to calculate the resemblance of two images. In particular, OpenCV offers three different methods for calculating the resemblance Δ of two contours A and B yielded by the algorithm described in section 4.7. The methods are the following [24]:

$$\Delta_1(A,B) = \sum_{i=1..7} \left| \frac{1}{m_i^A} - \frac{1}{m_i^B} \right|,$$

$$\Delta_2(A,B) = \sum_{i=1..7} \left| m_i^A - m_i^B \right|,$$

$$\Delta_3(A,B) = \max_{i=1..7} \left| \frac{1}{m_i^A} - \frac{1}{m_i^B} \right|.$$

4.8.4 Conic fitting

Fitting of conic sections can be done by finding the value on
$\mathbf{a} = [A_{xx}\ A_{xy}\ A_{yy}\ A_x\ A_y\ A_0]$ for which the function
$\epsilon^2(\mathbf{a}) = \sum_{i=1}^n \delta(C(\mathbf{a}), \mathbf{x_i})$ attains its global minimum, where

- $C(\mathbf{a})$ is a family of curves, defined by $\{\mathbf{x}|F(\mathbf{a};\mathbf{x}) = 0\}$ with $F(\mathbf{a};\mathbf{x}) = [x^2\ xy\ y^2\ x\ y\ 1] \cdot \mathbf{a}$
- $\mathbf{x_i}$ refers to a point (x_i, y_i) with $1 \leq i \leq n$
- $\delta(C(\mathbf{a}), \mathbf{x}_i)$ measures the distance of a point $\mathbf{x}_i$ from the curve $C(\mathbf{a})$.

There exist different proposals for the function δ. OpenCV uses the algorithm LIN[25], where the δ-function is $F(\mathbf{a}, \mathbf{x}_i)^2$ under the constraint that $||\mathbf{a}||^2 = 1$.

4.9 Hough transform

The Hough transform is an algorithm for detecting features. The classical Hough transform was initially limited for detecting lines but the Hough transform has extended to also handle other shapes such as circles and ellipses. The classical Hough transform for detecting lines will first be explained in order to explain some variations of it.

4.9.1 Classical Hough transform

Lines in the Hough transform may be represented in different ways. The slope-intercept representation, defined as follows, is simple and intuitive:

$$y = kx + m$$

where (x,y) are coordinates, k is the slope, and m is the interception of the line with the y-axis.
However, this representation cannot define a straight vertical line. Therefore, most modern implementation of the Hough transform use a normal representation in polar coordinates. In this representation, a line is defined by its normal that goes through the origin. The normal is defined by (ρ, θ), where ρ is the distance from the origin, and θ is the angle from the x-axis, as pictured in figure 4.6. Thus, given (ρ, θ), the line can be defined by

$$\rho = x \cos\theta + y \sin\theta.$$

The classical Hough transform will, for each pixel (x, y) in a set, give one vote for each line that can possibly go through it and add it in a so-called *accumulator*. It will then use a threshold that determines how many votes a line needs to be considered as a line in this image.
Usually, Hough transform takes a binary image as input, where one color represents the pixels that should be tested. Often, it is used on a picture where edge detection has been made. In some cases, such as in the case of the library OpenCV, further explained in section 4.1, a grayscale image is given as input. In that case, non-zero pixels will be used.

4.9.2 Probabilistic Hough transform

Of course, the classical Hough transform may give performance issues on large pictures. Kiryati et. al [26] proposes that only a fraction of all edge points are taken into consideration, and shows that this doesn't usually lead to considerable problems

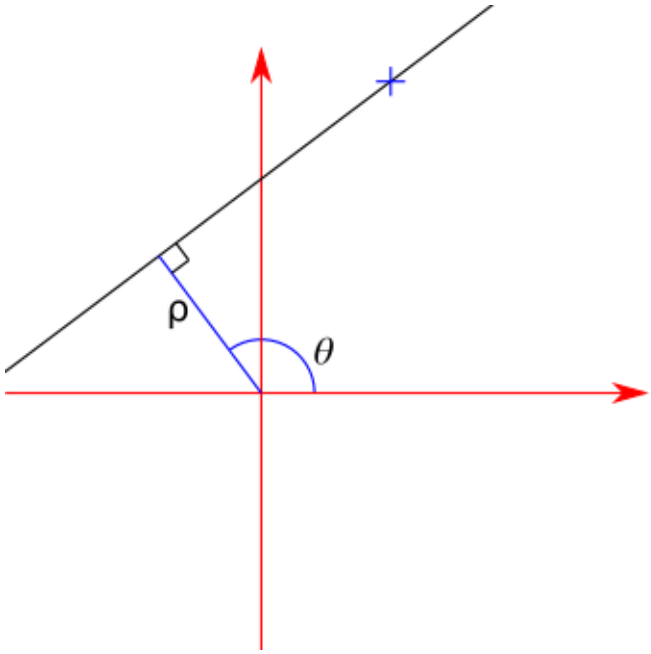


Figure 4.6: A line, described with its normal with polar coordinates, that goes through a point (marked as a '+')

in the correctness of the solution. In fact, experiments with fractions as low as 2% has been shown successful.

4.9.3 Progressive probabilistic Hough transform

As noted by Matas et al.[27], the derived formulas of the probabilistic Hough transform require previous knowledge of the number of points belonging to the line. However, this is rare in practise. Instead, a progressive method to determine the poll size is proposed.
The first step in the algorithm is to look at one random edge point and update the votes for the lines that can possibly go through it. Out of these lines, the line with the highest number of votes is checked against a certain threshold. Unless the number of votes for this line is higher than the threshold, the algorithm goes back to the first step, choosing another random point. Otherwise, it checks for the longest continuous segment (or a segment with no more gaps than in a given threshold). All the points in this segment are removed from the input image and their votes are removed. If the line is longer than a given minimum length, it is added to the output list. This algorithm loops until all input edges have been removed.

4.9.4 Other variations of Hough transform

There exist variations of the Hough transform that can be used for analytically defined shapes, such as circles and ellipses. What differs them from the other Hough transforms are the parameters. For instance, a circle can be defined by the three parameters (x, y, r), where x and y represent the position of the circle's center and r its radius.

For non-analytically defined shapes, such as hand-drawn shapes, parameters describing the reference origin, the orientation, the scale factors are used. These are looked up in a table which has been created in advance.

5
Android

Android is an operating system currently developed by Google for mobile devices such as smartphones or tablets. It is based on a Linux kernel and is primarily designed for devices using touch, such as swiping or taping, as input method [28]. It has currently the largest installed base of all operating system [29].

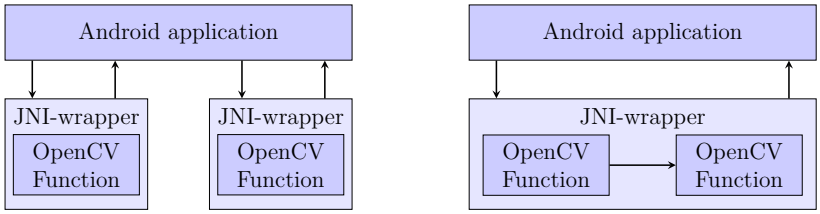
5.1 Development for Android devices

Android offers an application programming interface (API) and a software development kit (SDK) for application development [30]. The API is designed for Java, but other languages can be used. Code written using the Android API is compiled and then executed through a virtual machine which is officially Android Runtime (ART) as of Android 5.0. It is also possible to write code without using the Android API in other languages such as C and C++ by using the Native Development Toolkit (NDK) [31]. However, some parts of the application need to be written using the Android API and each call to the NDK through the Java Native Interface (JNI) requires overhead,so in most cases there will be no speed improvements using the NDK [32].
The Android API includes a camera API and thread handling.

5.2 OpenCV in Android

When developing software that uses OpenCV in Android, the code can be written using either the Java API or the NDK.
When using the Java API, the OpenCV library has to either be loaded at runtime, or a connection has to be established with the OpenCV Manager app which may already have loaded the library. OpenCV Manager is available on Google Play and keeps the library updated while minimizing the size of applications that make use of OpenCV by sharing the data through inter-process communication (*ipc*).
If using the NDK, one can develop the algorithms in C++ and then export the functions as a library which then can be loaded at runtime by the Android app. This minimizes the number of JNI-calls needed, especially if multiple sequential calls are made for accessing data in OpenCV matrices, since every access call to matrices requires overhead. This is illustrated in figure 5.1.
It is possible to load matrices into primitive java arrays and store them back into OpenCV matrices, thus minimizing the overhead while still using the Java API.

However, this is under documented and type unsafe, and empirically we have found the type conversion to be inconsistent.



(a) Two JNI calls are needed when using OpenCV through Android's Java API

(b) One JNI call is needed when using OpenCV through Android's NDK

Figure 5.1: Comparison between using native OpenCV and Java OpenCV. It illustrates how calling two OpenCV functions sequentially would be handled for each method.

5.3 The Android camera

This section explains how an Android device can be configured for dealing with the challenges that can arise within the area of this project.

5.3.1 Camera API 2

In order to access the camera of an Android device using Camera API 2 one first needs to create a session. It is important to consider that opening a session takes time and thus it is recommended to only close a session if the camera should not be used in a relatively long time. In order to actually take a photo different *capture requests* may be sent to the session. These *capture requests* can be configured using *capture requests builders*. The different possible settings depend on the camera hardware level of the device. There are three different hardware levels offering different amounts of properties to modify, *LEGACY*, *LIMITED*, and *FULL* where *LEGACY* is the most limited.

5.3.2 Capturing a display

The properties of the displays have significant effect on how to take a photo of it. While LCD often is reflective, see section 3.3.1, and thus only reflect the lights within the room, the LED displays are always emitting light. Taking a photo of a LED may therefore cause distortions to the photo turning the segments into white color while the color of the LED rather colors the area surrounding the segments as seen in figure 5.2. Avoiding this can be done by decreasing the exposure index or exposure time and thus, capture less light. Another benefit that arises from doing so is that the surroundings also darkens, resulting in less noise from other object within the photo. However, Android devices do not always offer the possibility to manually adjust these settings. Often these settings are automatically configured.

As of devices of API level 21 or above, the camera 2 API indeed supports modifying these parameters but in order to do so the device itself must also have support for it. Unfortunately the device used in this project was of hardware level *LEGACY*, as described in 5.3.1, and is thereby only backward compatible, not offering these features. However, these devices still support a feature of locking the automatic exposure settings. While the camera automatically adjusts the exposure according to the amount of light in the room it is possible to, at a given time, specify that no matter how the lights in the room changes, keep the exposure steady. By making use of these features it is possible to let the user of the android device manually set exposure by holding the camera close to a lit segment, an always lit calibration bit for example, locks the exposure and then afterwards, as the exposure time is steady, start capturing patterns. The result is that the segments now indeed is colored and the surrounding area is darkened.

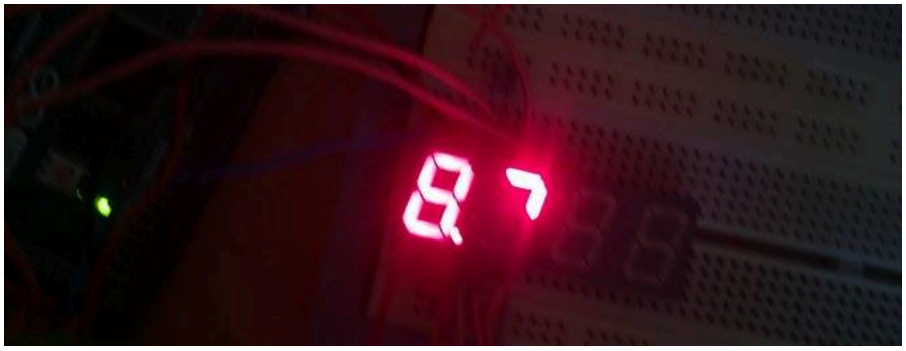


Figure 5.2: How LED displays are captured by a camera. The actual segments are white rather than red while the red color instead distorts the surrounding area.

6
Development and verification

This chapter explains the materials used during the project. Further, it describes the lab environment and development tools.

6.1 Materials

The product has been developed in *Java* using the software library *OpenCV* and is targeted for an Android of *API* 22. The lab environment was created using an *Arduino* and seven segment displays of the types; 1) *LCD* and 2) *LED* with red diodes. The Android device used within the project is a Sony Xperia Z3, API 22.

6.2 Generating synthetic test cases

In order to be able to test our algorithms, we first decided to create a synthetic lab environment that generates pictures of seven segment displays. This software was mainly developed in Haskell [33] using JuicyPixels [34], with some additions of 3Drotate [35], a script that uses ImageMagick [36], and some direct calls to ImageMagick scripts. The seven segment display generator supports creating displays with different colors, shapes, rotations, angles and lights. After generating a display with random configurations, it is put on an image provided by the user in a specified directory. It does this for all pictures in the directory.
The first part, which is developed in Haskell using JuicyPixels, is used to create flat and clear displays. It has configurations for the following parts:

- Segment width and length
- Margin between segments
- Dot radius and position
- Background color
- Active foreground color (when segment/dot is lit)
- Inactive foreground color (when segment/dot is unlit)
- Filter. These can be defined by Haskell functions. A brightness filter is provided which puts gradient light on a display.

An example output of this part is shown in figure 6.1
The 3Drotate part rotates the image in an arbitrary way, applies Gaussian blur to it, and puts it on an image. An example output of this is shown in figure 6.2
The intention of this software was not to prove that our solution works in real life, but rather to easily enable us to investigate the properties of the patterns. In real

Figure 6.1: Example of a generated display using our image generating software

life, many other factors may cause the algorithm to fail, such as more complex lightning, lens distortion and bad camera lens focus.

6.3 Lab environment

The lab environment was built using an Arduino Mega 2560 and seven-segment displays of the following types.

- A reflective LCD display capable of displaying 4 units in a row where the last unit in the row lacks decimal point.
- Two red LED display of two units in a row each. Together, these displays allowed for the dimension 1x4 or 2x2. The units are slightly tilted.

The Arduino was programmed to represent a given number. It automatically divides the message according to the dimension and adds the calibration and sequence bits as descried in chapter 7. Afterwards, it continuously repeats sending the sequences, 2 each second, using either the *LCD*-, or the *LED*-displays. It never terminates unless power is turned off. The environment allows for testing the performance of the product as well as how it handles with different settings:

- Red LED displays and reflective LCD displays.
- Patterns of the maximum dimension of either 2x2 or 1x4.
- Messages divided into sequences.

Figure 6.2: Result of applying Gaussian blur and rotating the generated display, and putting it on an image

7

Communication and representation of data

A primary task within the project is defining a protocol allowing machines to display information using seven-segment patterns. These patterns may be decoded using a smartphone that runs our suggested app. This protocol should allow for transmitting information of different contents. We want the protocol to be as portable as possible. This means, more specifically, that it should be possible to scan patterns with an arbitrary number of seven segment units aligned in rows and colums, and that there should be as little restriction as possible on the amount of transmittable data.

7.1 Bit significance

In order to use ASP patterns as a bit representation it is important that the significance of different segments within that pattern is agreed upon. This protocol defines a standard to ensure both parties know how to represent any data.
A seven segment unit is one entity of a seven segment display that contains 7 bars and one decimal point. Given that all bits within a unit are used, the significance of each bit (b) is pictured in figure 7.1.

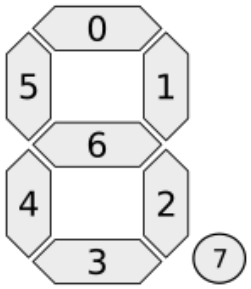


Figure 7.1: Bit significances in a unit. [1]

if a display consists of many units without any unused bits that are aligned in r rows and c columns, the significance of a unit column (i) and row (j) is given by

[1]Image derived from: `https://commons.wikimedia.org/wiki/File:7_segment_display_labeled.svg`, [Online; accessed 30-November-2015]

the following formula:

$$(i \cdot r + j) \cdot 8.$$

Where the i and j are zero-indexed, thus giving them the following range:

$$0 \leq i < r,$$

$$0 \leq j < c.$$

The actual significance of a bit is given by adding the significance of the unit with significance of that bit within the unit, resulting in the following formula:

$$(i \cdot r + j) \cdot 8 + b.$$

Figure 7.2 shows the bit significances of a display with $r = 2$ and $c = 3$

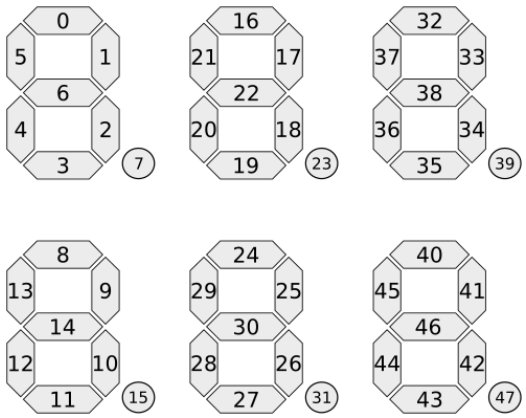


Figure 7.2: Bit significances of a 3x2 display.

If a bit is unused, the significance of the bits with higher significance are decreases by 1 as seen in figure 7.3.

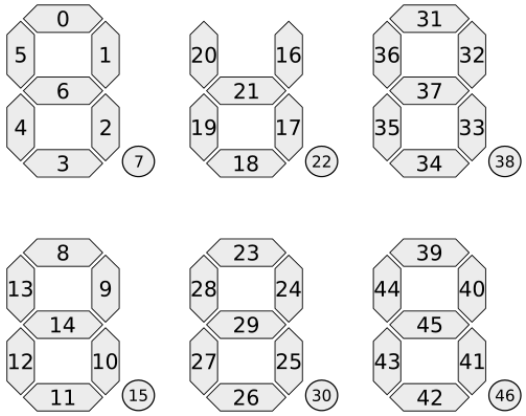


Figure 7.3: Bit significances of a 3x2 display with bit 16 unused.

7.2 Calibration bits

In order to identify a seven segment display as well as extract some information about the image itself, some bits are always lit and considered unused in the data transmission. These bits are called calibration bit. There are two requirements when choosing which bits that should be calibration bits.

- They should provide information about the region of interest. In other words, it should be possible from the calibration bits to identify the outer layer of segments.
- They should provide a mechanism for identifying the correct rotation of the display.

Satisfying the first requirement was done by simply choosing corner segments. Extending these segment will mark the region of interest. However, the last digits of each row may still have dots that will be outside this frame. Either one can decide not to use these dots in the encoding of information, or one can choose to add a border to the identified frame to make sure the dots are included even if outside the frame. The result of ignoring the dots is that each frame can contain less information and thus we chose to add a border when cropping the image to include any outlying dots.

In order to satisfy the second requirement, that it should be possible to determine the correct rotation, we simply used a dot as an extra calibration bit. A dot can only be positioned correctly with respect to any other segments if the image itself is rotated correctly.

In conclusion, the segments that are set as calibration bits and are thus unused in the transmission of the message are 0, 5, 7, $(8 \cdot r - 5)$, and $(8 \cdot r \cdot (c-1) + 1)$ where r is the number of rows and c the number of columns. For example, with r=2 and c=3, we get that the calibration bits are 0, 5, 7, 11, and 33. This is shown in figure 7.4, together with an illustration of why these calibration bits have been used so that it includes a frame of the whole display.

7.3 Dynamic sequencing

In order to allow for more information, ASP uses message sequencing. This means that long messages can be split up into multiple frames and sent one after another, in sequences, using a decided frame rate. The proposed frame rate is set to 500ms, which is discussed in chapter 10.

7.3.1 Bit significance using dynamic sequencing

In order to allow the user to start scanning whenever it wants, ASP repeats the process after it has reached the last frame. For this to be feasible, some indication about which frame it is showing is necessary. The proposed solution in ASP is to use frame index bits, which are bits that are unused in terms of transmitting data, but include an index number. The frame index bits are put on the bits with lowest significance. The bits are unused in that they are not used in the decoded message,

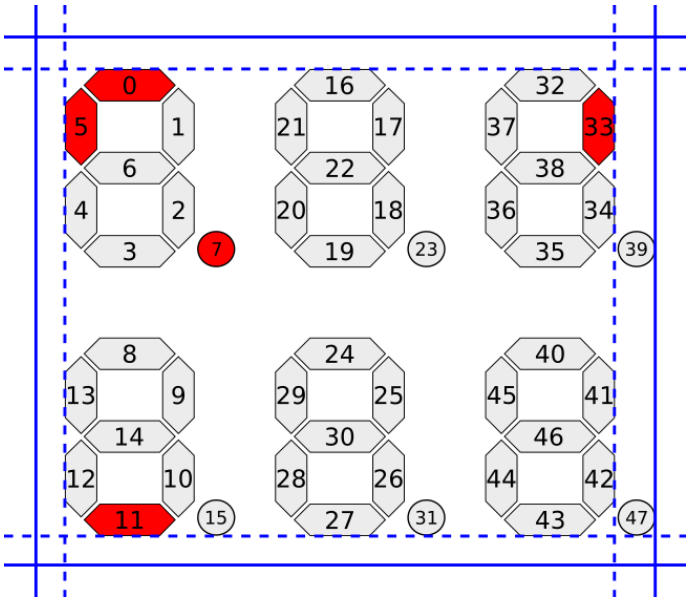
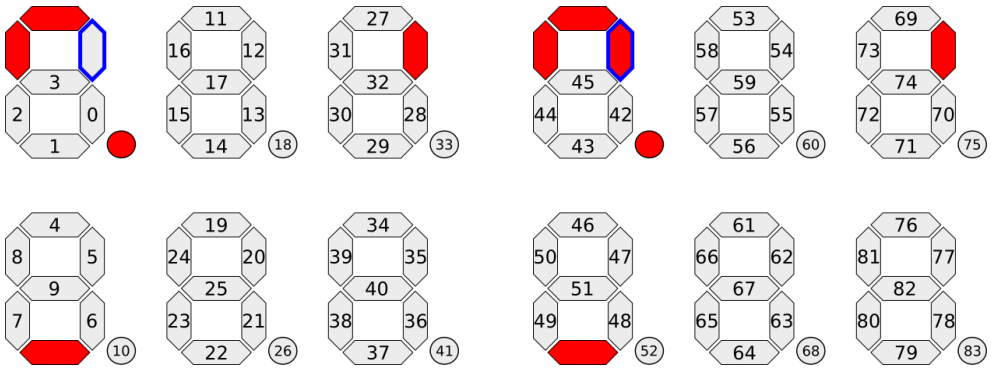


Figure 7.4: Calibration bits shown in red on a 3x2 display. By extracting the outer lines from the calibration bits, a region of interest can be found (dashed lines). In order for the right-most dots to be included, a margin needs to be applied (full lines). Bit 7 was chosen in order to know the direction of the picture.

but the index shown by the frame index bits i_f (used as unsigned integers) also changes the significance of the resting bits by the following formula:

$$s_f \cdot i_f + b.$$

Where s_f is the number of used bits in a frame and b is the significance if the index bits were simply unused. An illustration where the calibration bits are unused and the number of frame index bits is set to 1 is shown in figure 7.5



(a) $i_f = 0$ **(b)** $i_f = 1$

Figure 7.5: Bit significance of two consecutive frames in a 3x2 display when the number of bits used for sequencing is 1 and all other bits but the calibration bits are used, giving s_f the value 42. The used frame index bit is outlined in blue, filled in red when it is lit

How many bits that are used as frame bits depends on the length of the message,

for which the necessary calculations are described the next section. This means that the sender (the ASP-display) and the receiver (the smartphone app) must know the length of the message, or at least on the number of frames to be used.

7.3.2 Calculating the number of frame bits

Dynamic sequencing is used to divide a message m represented as a bit array of size s_m into multiple, smaller frames. It does this by reserving some bits that are going to be used for indexing. The relationship between the number of sequences n_s that are needed and the number of bits used for indexing n_b is given by the following formula:

$$n_b = \lceil \log_2(n_s) \rceil.$$

When sending a message on a display with size s_d given by the number of segments and decimal points excluding the calibration bits, the number of sequences n_s that will be needed can be calculated as follows:

$$\min_{n_s \in \mathbb{Z}^+} n_s \text{ st. } (s_d - n_b)n_s \geq m_s.$$

Note that the number of sequences n_s cannot be solved if the number of bits reserved for sequencing n_b is greater or equal to the display size s_d. Thus, that message is too large for sequencing according to that frame size.
As mentioned before, these bits will be reserved in the beginning of each frame. Hence, the frame size s_f, used for each sequence of m can be calculated by $s_f = s_d - n_b$. The message will be partitioned so that each part will be of size s_f. Any unused bits within the last frame will be zero. Thus it is necessary that both the encoder and the decoder using this protocol knows the message length.

8

Algorithms for reading ASP-displays

There are different algorithms that may be used for reading ASP-displays. These algorithms can be divided into different steps that can be combined in different ways. These algorithms are evaluated in the next chapter but an overview together with explanations over the steps are given in this chapter.

8.1 Evaluated algorithms

As seen in figure 8.1 there are different approaches suitable for reading and processing scans of seven-segment displays. However, any combination is not possible since some approaches during one phase may depend on a certain method in the preceding step. Therefore, in order to fully understand the different approaches a brief explanation of the different evaluated algorithms is hereby given.

Method 1

This method is based on using the algorithm *Match Pattern*, see section 8.3.1, for finding the display in an image. The algorithm depends on using Hough line transform for finding lines in the image.

Method 1A In order to create a one channel image a simple conversion from color to gray scale is used. Adaptive threshold is then used for marking the edges.

Method 1B As with method 1A a simple color conversion are used for gray scaling. However, in order to mark the edges Canny edge detector is used.

Method 1C This approach differs from the previous methods by using the algorithm *Find color*, see section 8.5.1, for creating a gray scaled image. Canny edge detector is the applied for marking edges.

Method 2

This algorithm is instead based on the algorithm *Match Shapes*, see section 8.3.2, for finding the display. The algorithm does not require edges being marked but requires adaptive threshold for finding the characteristics of the image.

Method 2A Simple color conversion is used for gray scaling the image.

Method 2B Find color is used for the gray scaling.

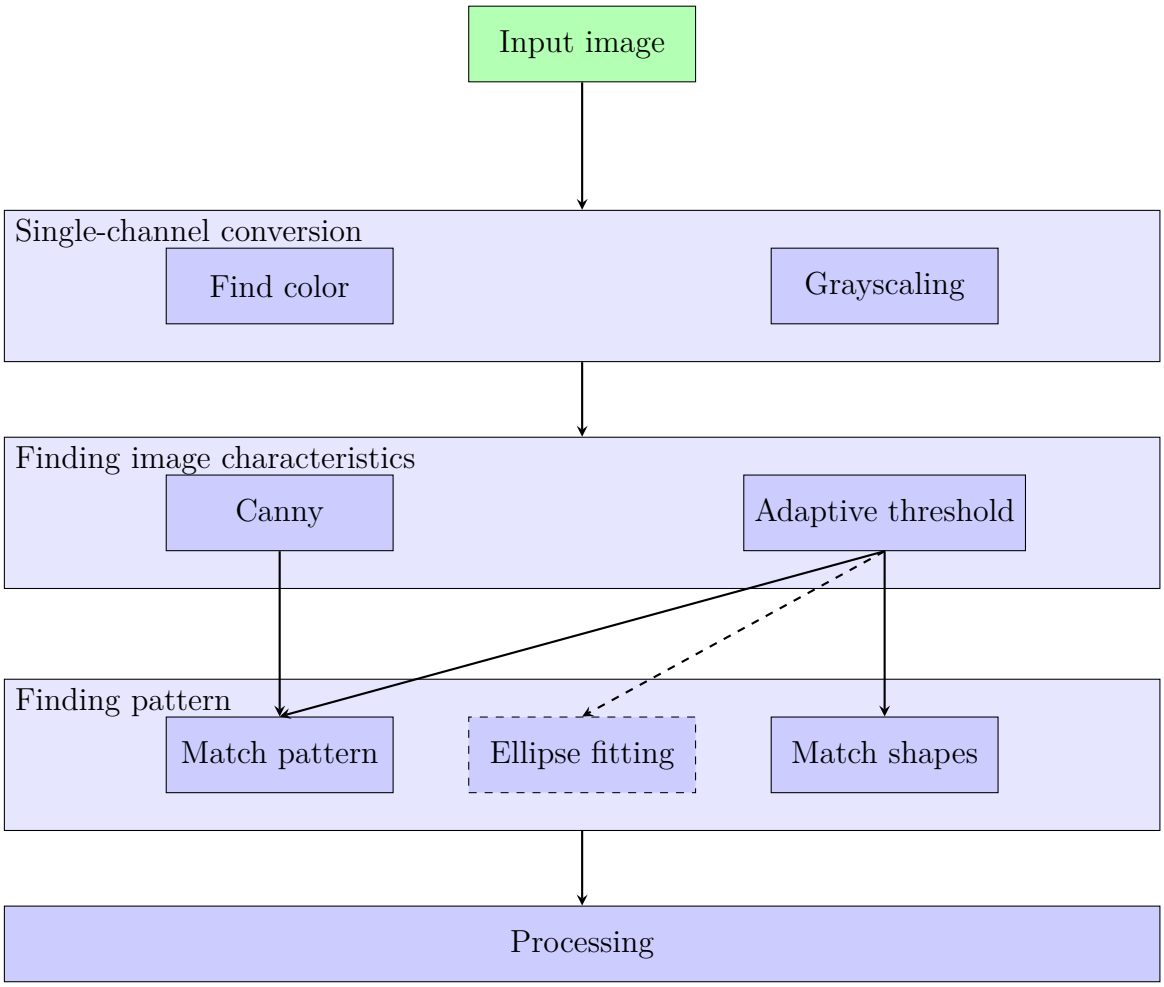


Figure 8.1: Flowchart of different possible paths for a seven-segment display detection algorithm. Lighter blue boxes means that one of the darker blue boxes in them may be used. Dashed lines and boxes represent untested steps and paths in our results. Each step is further explained in this chapter.

8.2 Processing

The final processing takes a single-channel image of a display with a straight, front-up perspective, and analyzes which segments are lit. Lit segments should be highlighted with a non-zero value, while the rest should be set to zero. This thesis proposes no more than one algorithm for this step because of its simplicity. The algorithm looks at specified pixels and determines if the segments are lit or not. If the calibration bits are unlit, the algorithm rotates the image 90° or 180° depending on previous knowledge. This is repeated until all combinations are tried or a pattern is found.

8.3 Finding a pattern

The algorithms described in this section take lines found in an image to determine where the pattern is. As the four corners are found the next step is to perform a quadrilateral crop to extract the identified frame to a perfect rectangle suitable for interpretation. As mentioned in 7.2, it is important to add a border to the identified frame in order to avoid ignoring any dots outside the frame. As the lines surrounding the pattern are found it is possible find the four corners, (x, y) of the pattern by calculate the following simultaneous equations for each intersecting line:

$$\begin{cases} y = k_0 \cdot x + m_0 \\ y = k_1 \cdot x + m_1 \end{cases} \Rightarrow x = \frac{m_1 - m_0}{k_0 - k_1}.$$

In the event of a completely vertical line, the corresponding k is ∞. Thus, the equation cannot be solved. However, since the x-coordinate then is fixed it is possible to just calculate y using $y = k \times x + m$ using the k and m of the other line.

8.3.1 Match pattern

Match Pattern is a classifying algorithm that finds a seven segment pattern from lines represented as (x_0, y_0, x_1, y_1) where (x_0, y_0) represents the beginning of the line and (x_1, y_1) represents the end. These lines were detected by using the progressive probabilistic Hough line transform as described in section 4.9.3 since it is considered both faster and more accurate than the other implementations. As lines are inserted the algorithm translates them into polar coordinates where θ is the angle of the line and ρ is the smallest distance from the line to the center of the image, see figure 8.2. Theta is orthogonal to the angle of the line, thus we have the following:

$$\Delta x = x_1 - x_0,$$
$$\Delta y = y_1 - y_0,$$
$$\theta = 90^\circ + \arctan \frac{\Delta y}{\Delta x}.$$

Calculating ρ is now done by first calculating the center, (x_c, y_c), the angle pointing to that center, α, and the distance, d from the origin to that point, d;

$$x_c = x_0 + \frac{\Delta x}{2},$$
$$y_c = y_0 + \frac{\Delta y}{2},$$
$$\alpha = \arctan(y_c, x_c),$$
$$d = \sqrt{y_c^2 + x_c^2}.$$

Note that arctan with two arguments is used for having it defined 360°. Using these values it is now possible to calculate ρ,

$$\rho = d \times \cos(\theta - \alpha).$$

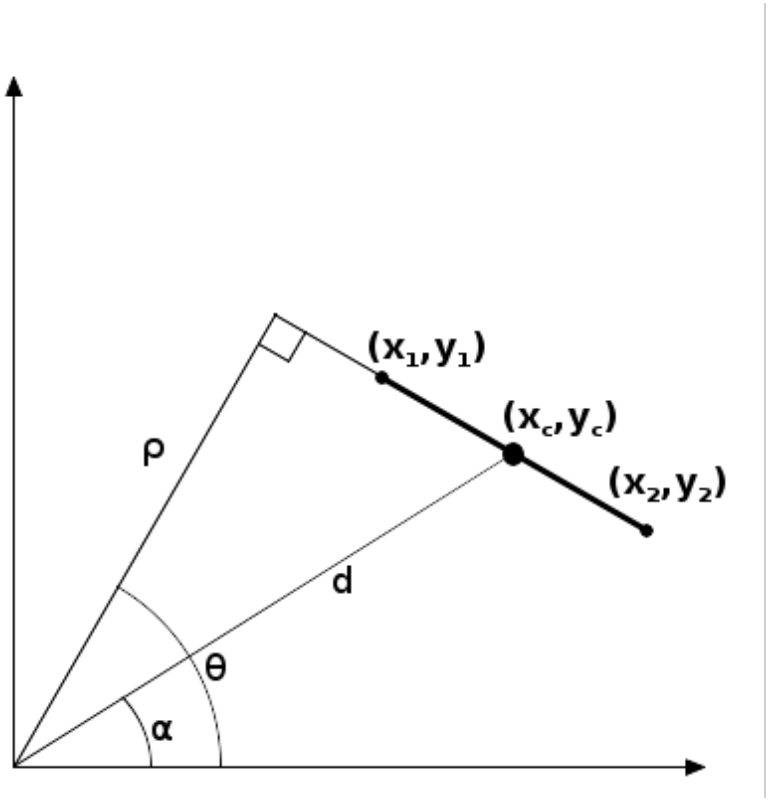


Figure 8.2: Illustration of how to represent a line

These lines are then categorized into separate classes after their corresponding theta. As all lines found are inserted the algorithm enters a voting procedure for each class. During this procedure it takes two lines inserted and calculates the point where they would intersect and the distance from each other. Given θ_1, ρ_1, θ_2 and ρ_2 the intersection, (θ, ρ), can be calculated as follows:

$$t = \sqrt{\rho_1^2 + \rho_2^2 - 2\rho_1\rho_2\cos(\theta_1 - \theta_2)},$$

$$\theta = \frac{\arccos(\rho_2 \sin\theta_1 - \rho_1 \sin\theta_2)}{t},$$

$$\rho = \frac{t}{\sin(\theta_1 - \theta_2)}.$$

Using this information along with the dimension of the display one can assume which of the lines in the pattern that the two selected may correspond to. With this information the algorithm then creates a representation of where all segments in the given direction may be, see figure 8.3. All detected lines in that direction are then used to up- or down-vote that assumption. If, Hough Lines Transform has been used to find the lines there will often be a detected line on both sides of each segment. If, in that case, there are two lines up-voting an assumption for a line, it is weighted as 2 votes while if only 1 line up-voted the assumption it is weighted as 1. If a line exists in the pattern that is positioned further away from an assumed line than the threshold given it down-votes the assumption.

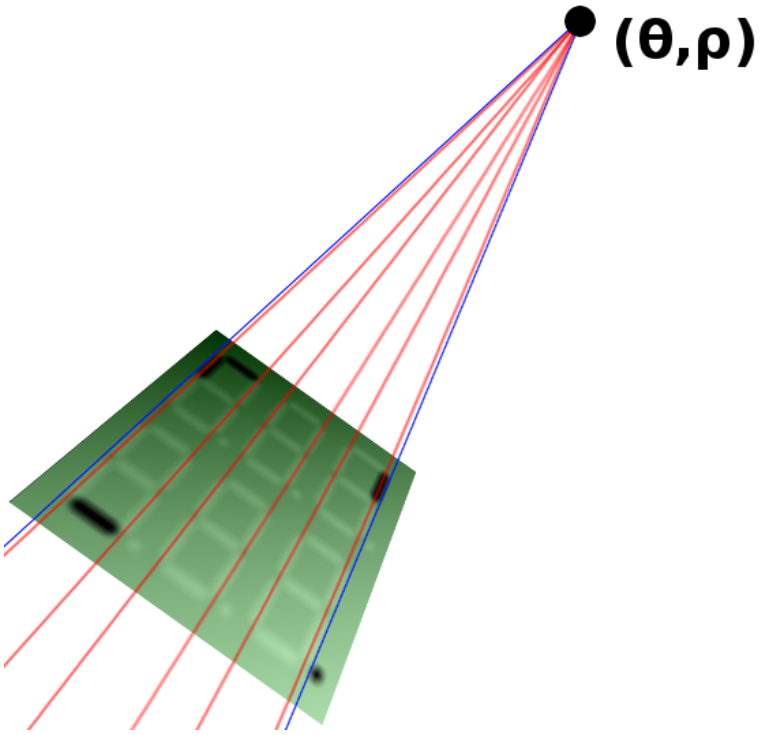


Figure 8.3: Illustration of how the segments in one direction are estimated, red lines, given the identified lines, blue lines.

The assumptions are then sorted after the number of votes. As this voting procedure is done for all classes it combines the result for any two different classes in order to create a grid. Any combination where it is clear that the two assumptions cannot be valid together, for example if the two assumptions are of different scales or that the angle difference is to small, are filtered out. Finally the grid with the highest number of votes are returned, see figure 8.4.

8.3.2 Match shapes

This algorithm needs as input a binary image where segments are of one color and the display background of another. The adaptive threshold algorithm described in section 4.6 is used with a relatively big kernel size for this purpose. The algorithm then uses this image to find contours using the algorithm explained in section 4.7 removing the ones that are not within a given size range, and then compares the contours to other contours (called *shapes*, see figure 8.5) in a dataset using one of the three comparison methods explained in section 4.8.3. The dataset consists of shapes that describes different combinations of segments since segments can sometimes merge after doing an adaptive threshold binarization. The closest match is used as long as it is close enough (using some threshold) and if either both have children or none does.
Some noise is to be expected, so we want to find a suitable middle point and only include the contours within a certain radius of this point. The middle point is chosen using the following voting procedure: For each match, its middle point gets

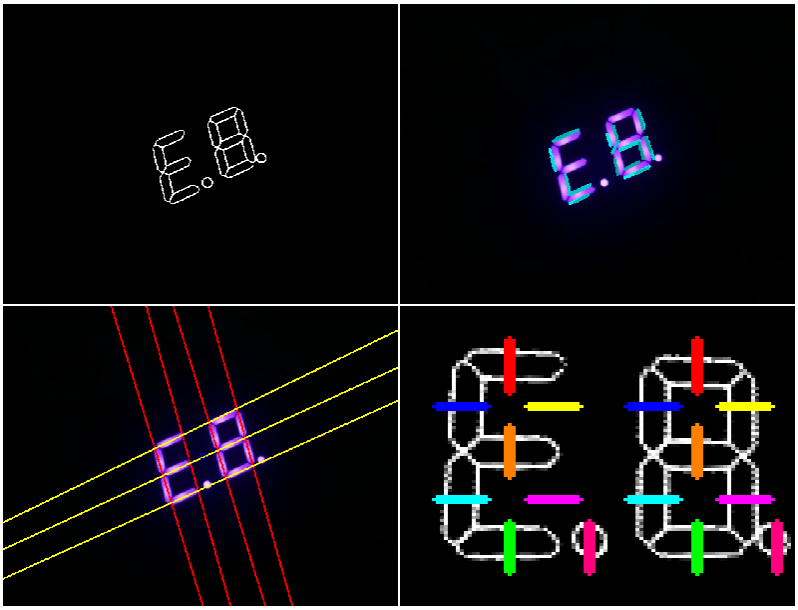


Figure 8.4: The top left image is the result after applying adaptive threshold. The top rights shows the lines detected by Hough Line transform. The bottom left image shows the up-voted grid using the Match Pattern algorithm. The last image illustrates the scanning procedure returning the result.

the number of segments it represents as number of votes. Then, for each point, the voting-weighted average is calculated and the image's middle point is chosen. For instance, imagine a situation with three merged segments located at (x_0, y_0) and one lone segment at (x_1, y_1). The three merged segments will most likely be matched with a shape which is known to represent three segments and the lone segment with a shape which is known to represent one segment. Therefore, the image's middle point will become $((3x_0 + x_1)/4, (3y_0 + y_1)/4)$.
Finally, a rotated rectangle with a minimum area enclosing the contours is found and rotated to a straight state.

8.3.3 Ellipse fitting

Segments of seven segment displays have shapes that somewhat resembles ellipses. Using optimized algorithms for finding ellipses instead of general shapes could therefore be beneficial for the performance of the final algorithm.
Initially the algorithm finds contours in an image using the contour finding algorithm explained in section 4.7 and then fits ellipses to them using the algorithm explained in section 4.8.4. It then filters these ellipses by only taking those that has the form of a segment, which is defined as having a height-width ratio within a certain range and a height of a certain range. The range of the height is set with knowledge of the camera properties (such as its resolution and angle of view) and the requirements (maximum distance between the camera and display) in mind. Outliers are then removed by selecting a averaged middle point of all the ellipses and selecting only those within a certain radius of that point.
The found ellipses could then be used to either extract lines by taking the widest

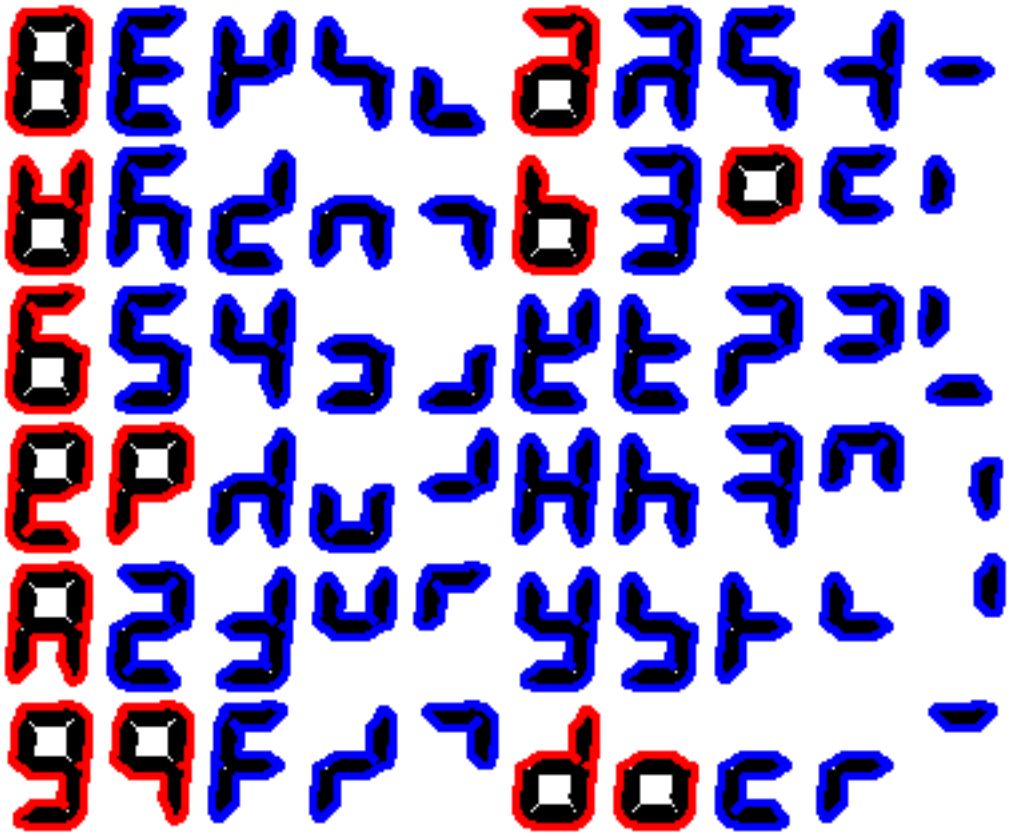


Figure 8.5: Unique segment shapes for the LCD Display. Shapes with children are marked in red and those with no children are marked in blue.

parts of the ellipses and sending them to a pattern finding algorithm as described in section 8.3, or to enclose a rotated rectangle around the ellipses in the same manner as described in section 8.3.2 in order to send it directly to the processing algorithm.

8.4 Finding image characteristics

It important to create a binary image representing characteristics of an image rather than the image itself. These characteristics may be edges, or more specifically the difference in color rather than the actual color. This section will explain the different algorithms tested and compared within the project.

Canny edge detector

The Canny Edge algorithm, as described in section 4.4, is an algorithm specified on finding edges and may therefore be used to find the sides of the segments.

Adaptive threshold

Adaptive thresholding, see section 4.6, is another candidate algorithm for finding the characteristics that has been compared. It can either be used with a relatively small kernel to find edges, or with larger kernels to make whole segments in displays having

the same color. Smaller kernels are more appropriate if the result is sent to a line detection algorithm, while larger kernels are more appropriate for shape-matching algorithms.

8.5 Creating an image of one channel

The previously mentioned algorithms requires an 8 bit single-channel image to process. However, this image can be derived using different methods.

Grayscaling

A simple and fast way to convert a color picture into gray scale is to sum each channel with some ratio. In OpenCV and in this project, the following formula is used: $S = 0.299R + 0.5870G + 0.1140B$ [16], where R,G and B represent the red, green and blue channels respectively, and S represents the final intensity.

8.5.1 Find color

Instead of a plain conversion from a color image into gray scale, another conversion technique which filters colors and thereby also removes noise is investigated in this thesis. This is done by specifying a **trinarization function** $f : \mathbb{Z}_{256}^3 \to \mathbb{Z}_3$ on colors for each type of display. The function is applied for each pixel in a 3-channel image and results in three different colors:

- **0**, *foreground*: The color of the segments
- **1**, *background*: The color of the area surrounding the segments
- **2**, *undefined*: Color that is neither the foreground or the background

Doing this conversion may be beneficial for both the edge detection algorithms and the shape finding algorithm. However, a disadvantage of doing so is that it adds requirements on knowledge about the colors of the display being scanned. Beside being impractical for the user to specify what kind of display is being scanned, it also requires specifying definitions of the colors for each different kind of display.

A *Matlab* script was defined that takes samples from the background and foreground, analyzes them and illustrates the colors found in *3D* or more specifically *RGB-space*, see figure 8.7. Using these plots, geometrical surfaces were defined for deciding whether a certain color belongs to foreground, background or undefined.

The following equation is proposed for describing the red LED display used in this study. However, in the case of the reflective LCD we found that it is impossible to define a clear distinction between the foreground and the background. This is because the reflective display reflects the light in the room making it very sensitive for noise such as shadows.

$$f_{RED\_LED}(r,g,b) = \begin{cases} \text{foreground} & \text{if } r \neq 0 \\ & \wedge\ b < \frac{r^2}{400} + 400 + \frac{g}{2} + 40 \\ & \wedge\ b > \frac{r^2}{600} + \frac{g}{2} - 60 \\ & \wedge\ g < \frac{r^2}{255} \\ & \wedge\ g > \frac{4000}{r} - 30 \\ \text{background} & \text{if } r = 0 \\ & \vee\ g < \frac{4000}{r} - 70 \\ & \wedge\ b < \frac{r^2}{400} + \frac{g}{2} + 40 \\ & \wedge\ b > \frac{r^2}{600} + \frac{g}{2} - 20 \\ & \wedge\ g < r + 10 \\ \text{undefined} & \text{otherwise} \end{cases}$$

An example of applying f_{RED_LED} on an image of a red LED display is shown in figure 8.6

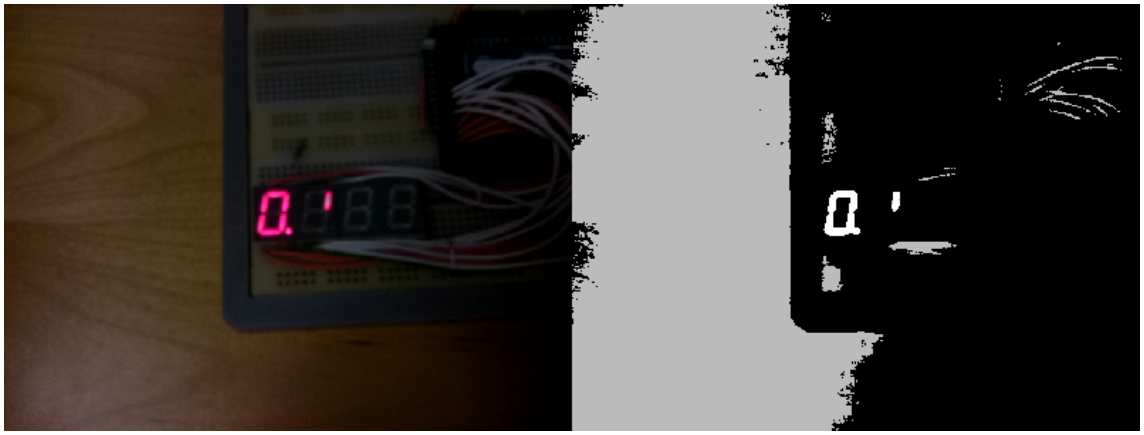
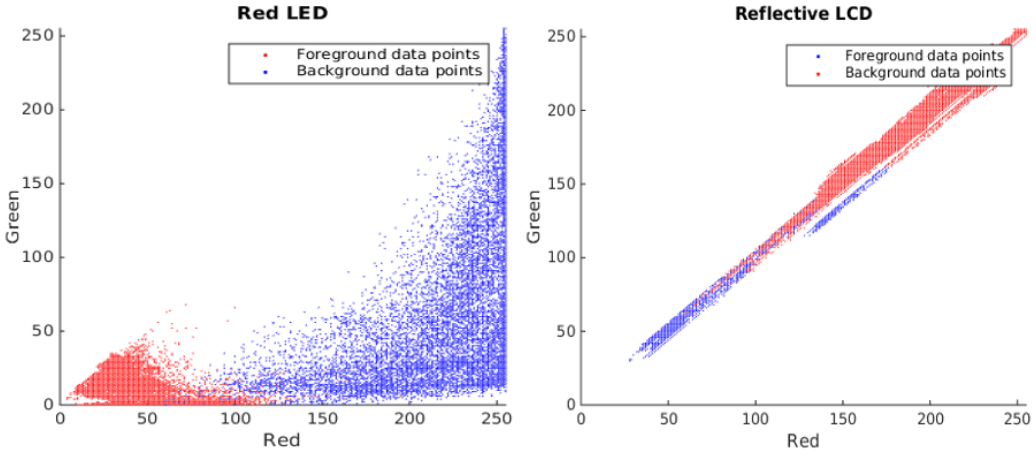
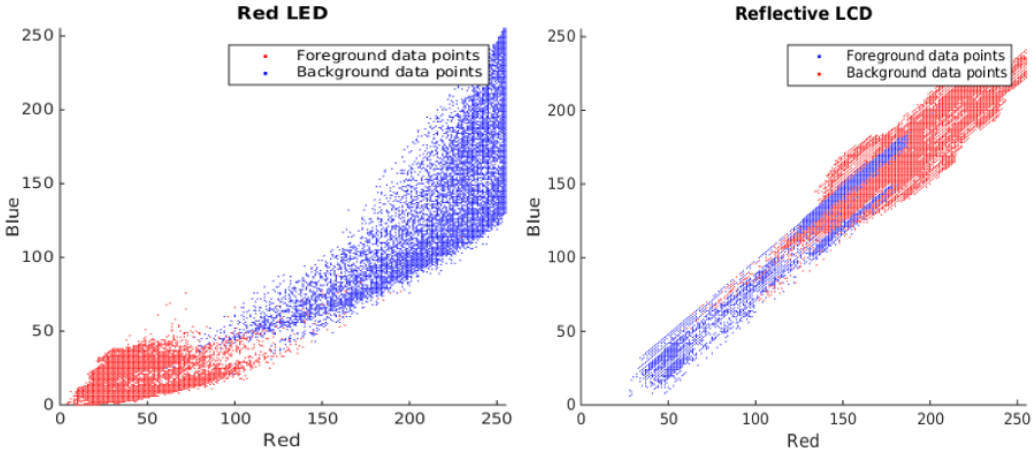


Figure 8.6: Left: An image of a red LED display. **Right:** Applying f_{RED_LED} on each pixel of the image. Foreground is colored as white, background as black and undefined as gray



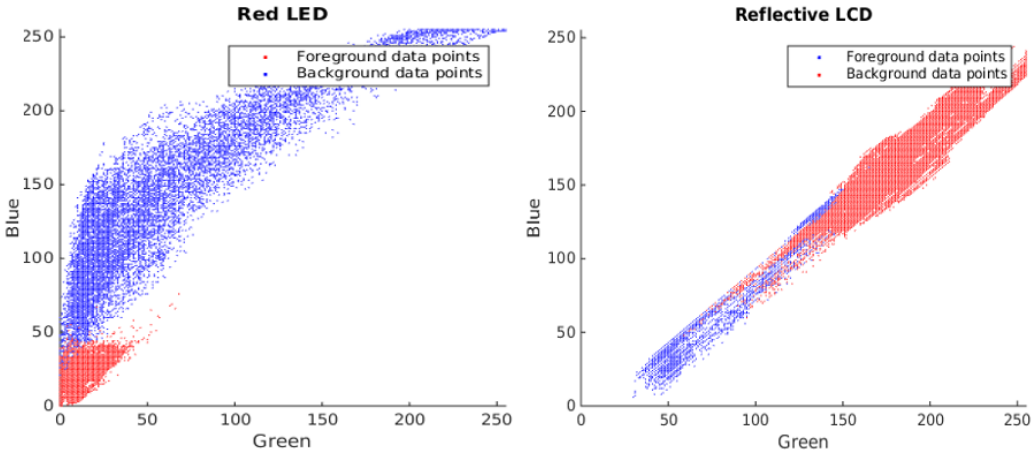
(a) Red LED
X-axis: Red, Y-axis: Green

(b) Reflective LCD
X-axis: Red, Y-axis: Green



(c) Red LED
X-axis: Red, Y-axis: Blue

(d) Reflective LCD
X-axis: Red, Y-axis: Blue



(e) Red LED
X-axis: Green, Y-axis: Blue

(f) Reflective LCD
X-axis: Green, Y-axis: Blue

Figure 8.7: The result when analysing the colors of two displays seen from three perspectives of the *RGB* cube. Foreground color is plotted in blue and background color in red.

9
Results and discussion

In this chapter, we evaluate how much data the proposed system is capable of transmitting. We first give a general overview of how the ASP protocol and its parameters affect the throughput, then we compare the proposed ASP-display reading algorithms in term of speed and correctness, and finally we motivate our choice of algorithm and parameters accordingly. As seen in figure 9.1, the choice of the parameters highly depends on the environmental properties. However, we chose values for the parameters in the algorithms that gave the best result. That is, in order of decreasing priority, the following:

- No or very few incorrectly identified patterns.
- As many correct identified patterns as possible.
- A high frame rate.

9.1 Data amount and throughput

The amount of data that can be sent each sequence depends on the number of units in the pattern. However, no matter the dimensions of the pattern there is always 5 bits reserved for calibration. In addition, with an increasing number of sequences more bits must be reserved for denoting what sequence is currently sent. This number increases logarithmically to the number of sequences. The data throughput also depends on how many patterns that can be interpreted each second. The maximum data that can be sent if not considering the time it will take can be calculated by reserving all bits but one as sequence bits. Thus, we have

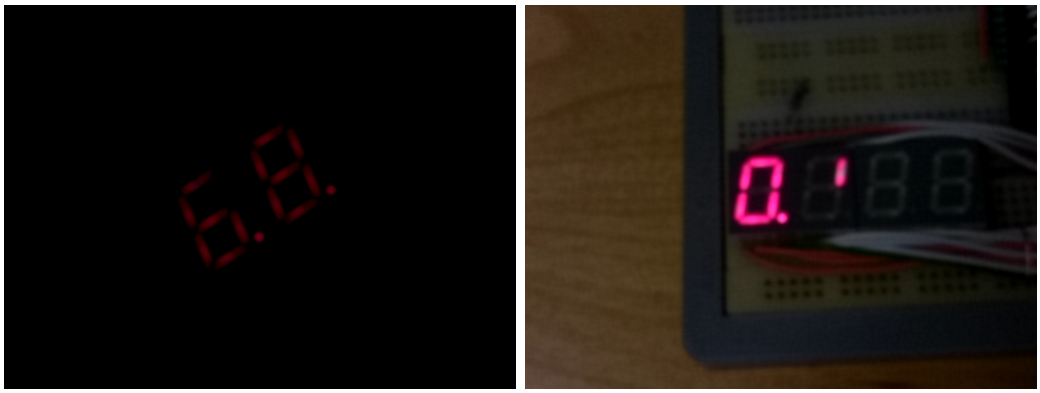
$$2^{n_b-5-1} \times 1$$

where n_b is the number of segments and dots. However, having this many bits as sequence bits also results in sending only one bit at a time. Therefore we have assumed that we cannot expect the user to wait for longer than 5 seconds to get the complete message. Hence, the maximum number of sequence bits that can be reserved can be calculated by

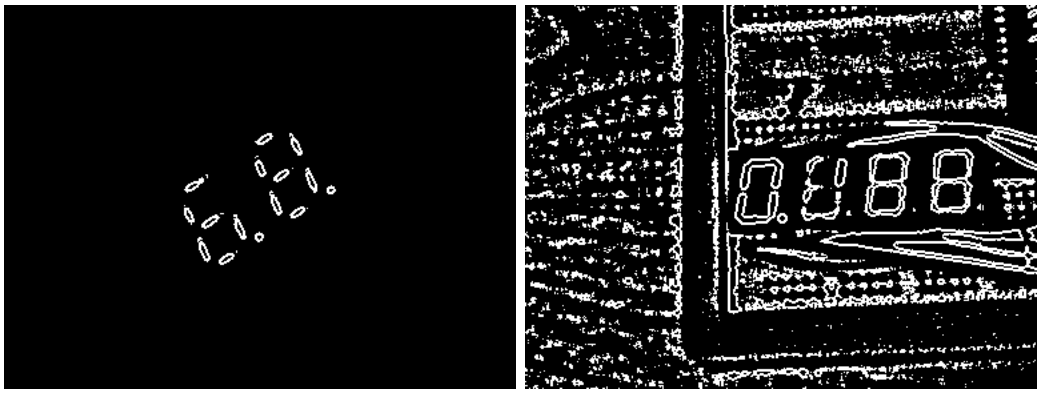
$$s_{max} = \log_2(\frac{5}{f_r})$$

where f_r is the frame rate and s_{max} is the maximum amount of sequence bits that may be used. The maximum amount of data can therefore be calculated using

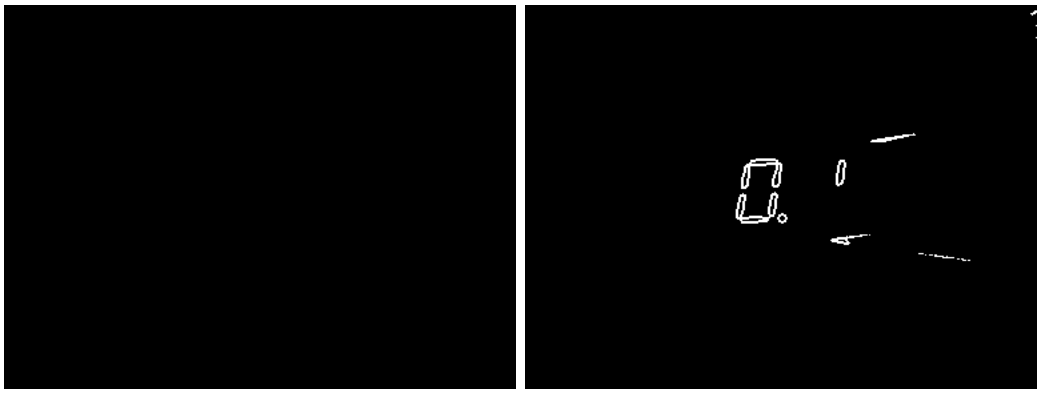
$$2^{s_{max}}(n_b - 5 - s_{max}).$$



(a) LED display in different lights.



(b) C = -2



(c) C = -20

Figure 9.1: The parameters of the adaptive threshold highly depends on the environmental properties.

However, the maximum data throughput happens when only sending one message and therefore not using any bits at all as sequence bits. Of course, this requires that the message cannot be of more bits than the number of bits in the pattern excluding the calibration bits.

9.2 Statistics

In this section, the different algorithms are compared with each other. Both correctness as well as speed are tested. The testing set consists of a total of 208 images where 57 of them includes a LCD-display and 50 a red LED-display and 101 non-sense images. Images has been chosen based on interesting properties worth investigating such as different angles and distances.
The different algorithms were compared with each other to estimate the potential of each approach. The test result can be found in the following tables and the results are also discussed in the following sections. During the testing the parameters were optimized for having as few incorrect interpreted images as possible. Thus, the algorithm is considered better if it rejects an image completely than falsely interpret it. Since falsely identified and interpreted patterns are considered a major drawback, additional testing was done using nonsense images containing no seven-segment displays at all.

Method	Corr.	Incorr.	Undef.	Avg (ms)	Worst case (ms)
1A	10	4	36	103	507
1B	21	4	15	49	371
1C	8	2	40	41	277
2A	39	0	11	67	380
2B	12	6	32	60	312

Table 9.1: Comparing performance and correctness of algorithm when using different edge detection algorithms on LED

Method	Corr.	Incorr.	Undef.	Avg (ms)	Worst case (ms)
2A	12	3	42	296	959

Table 9.2: Comparing performance and correctness of algorithm when using different edge detection algorithms on LCD

9.2.1 Discussion - Method 1 - Match pattern

The algorithm was not tested on LCD displays since the edges of the segments are much weaker than many other edges found in the image. Thus, any parameter during the edge detection that identifies the segment will also identify very much noise. Also the result for LED displays was unsatisfactory as seen in table 9.1. However the study showed several reasons for why these algorithms did not perform well. Firstly, the distance between each segment is about equal to the distance between the units in the display. This means that the grid created by the algorithm consists of perfect squares and that this, in turn, results in the algorithm displacing

Method	Corr.	Incorr.	Avg (ms)	Worst case (ms)
1A	97	4	305	1035
1B	80	21	509	2129
1C	101	0	14	128
2A	94	7	241	1136
2B	101	0	25	87

Table 9.3: Comparing performance and correctness of algorithm when using different edge detection algorithms on nonsense images

the grid if any outlying edges were not detected. A possible improvement for this could be to let more than one up-voted grid to the processing phase. This would however also result in a higher average and worst time. Another major drawback of the algorithm is that lines vary a lot in length. In order to detect the lines when scanning the segment from a distance it is necessary to allow for short lines. This however increases the possibility to also to find lines in the noise of the image.

Method 1A - Using Adaptive threshold As seen in 9.1, using adaptive for detecting edges was not beneficial. A main cause is that adaptive threshold may mark edges with a varying width. This causes Hough Line Transform to be able to draw lines in more directions than supposed, especially when allowing short lines. However it did reject almost all nonsense images.

Method 1B - Using Canny Edge Detector Canny Edge detector, in contrary, always marks edges using a width of one pixel. This caused the lines detected by the Hough Line Transform to indeed get the correct direction, thus the results are much better. However, it was a worse algorithm when it comes to rejecting false images, it does identify more edges in these images since it does not as effectively reduce larger areas of somewhat the same brightness of color.

Method 1C - Using Canny Edge Detector after Find Color Using the color filter did not increase the number of interpreted pattern, neither did it increase the percentage that are correctly identified. However, by filtering the color the noise was heavily reduced, thus none of the nonsense images was accepted. It had also a positive effect on performance.

9.2.2 Discussion - Method 2 - Match shapes

During the scanning of LED it was noticed that there were three different causes for not successfully scan a pattern. Most of the rejects happened because the light from a dot and a segment was merged together, see figure 9.2. This issue can however be solved by introducing the shapes where this phenomenon has happened to the training set. Beside this issue, it also failed once when the photo was shot in a bright environment. The pattern of the wooden desktop was rather enhanced when

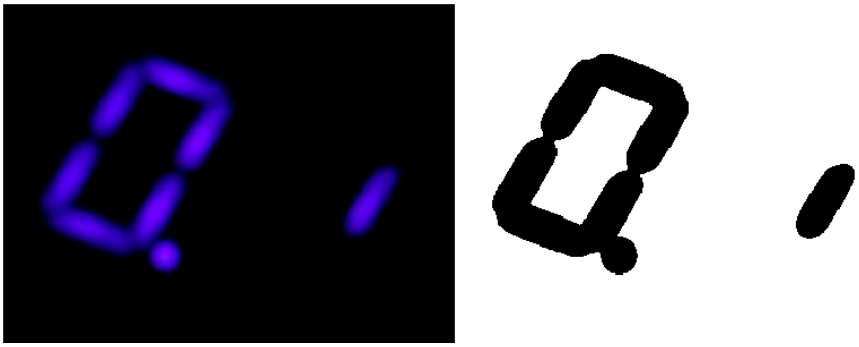


Figure 9.2: The dot and a segment are merged together creating a different shape.

adaptive threshold was applied, resulting in a false identification of a segment, see 9.3. The last issue causing the algorithm to fail identifying the pattern happened because some of the segments were currently changing state. Some of the segments was neither on nor off, see 9.4 causing adaptive threshold to create a cracked pattern.

Method 2A - Without Find Color It is rather stable when scanning LED displays, see table 9.1. It correctly identifies around 80% of given images. Those not interpreted are also rejected which is better than a false identification. However, it does not perform as well when it comes to LCD displays, see 9.2, and 7 of the nonsense images are also falsely interpreted, see 9.3.

Method 2B - With Find Color Using find color was not beneficial for the interpretation of patterns. Preprocessing the images using Find Color also resulted in the segments fading away when adaptive threshold was applied. However, it increased the overall performance and it also improved the ability to reject nonsense images. However, even if the result was not correctly interpreted it was noticed that the region of interest in noisy environment was better positioned.

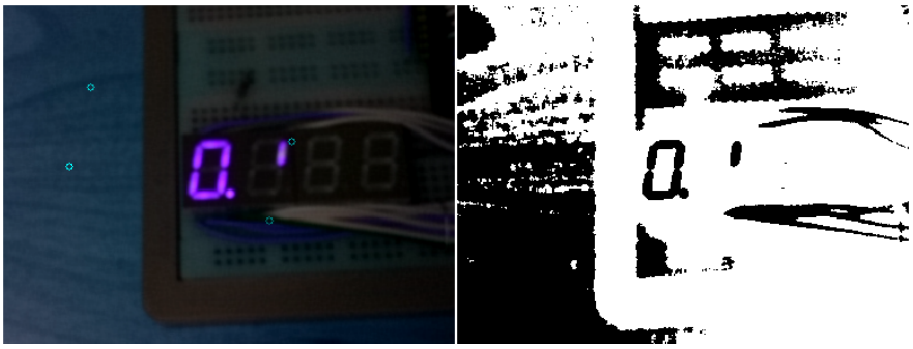


Figure 9.3: The photo is taken in a light room causing some structures in the wooden desk to be enhanced rather than filtered out.

In the event of interpreting patterns displayed on an LCD display, it was found that the algorithm does not perform as well. Worth to notice is that the environment where the photos have been shot contains a lot of noise as seen in figure 9.5. In order to improve this algorithm it is worth to consider implementing more filters

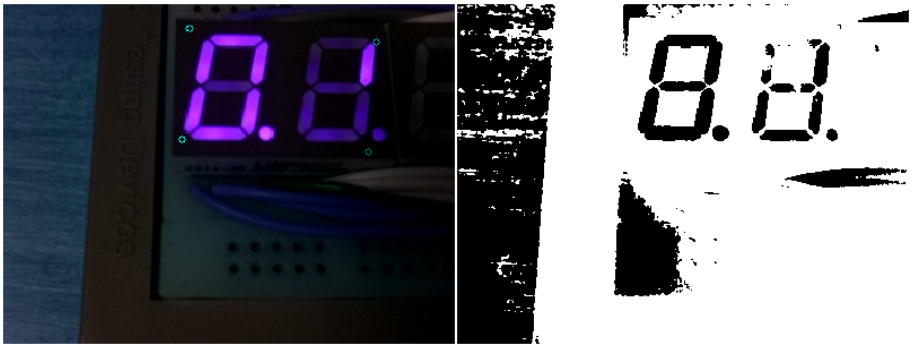


Figure 9.4: The segments are neither on or off making it hard to determine an actual state.

for filtering out noise that looks like segments. As the figure shows the detected rectangle has a very large width compared to the height why it should be possible to only allow rectangles of possible proportion given the dimension of the display.

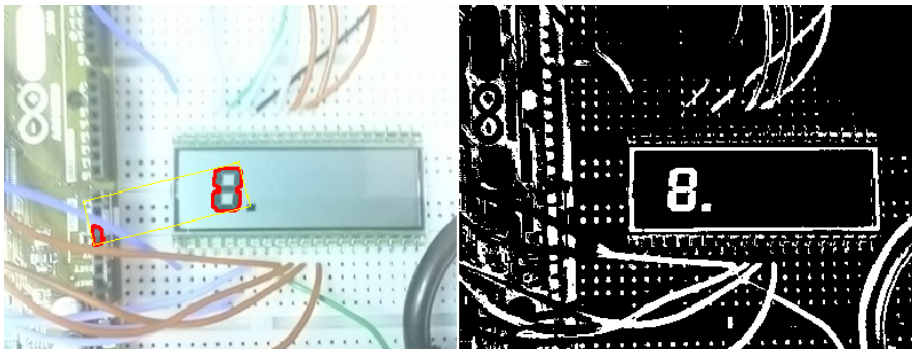


Figure 9.5: One of the segments found is actually noise, thus the pattern is not correctly identified.

9.3 Choice of algorithm and parameters

In this section the results are discussed and an algorithm is proposed. The result showed that the algorithm that performed best is Match Shapes. To add a margin to the time it takes to interpret the pattern we decided to choose a frame rate of 2 frames per second. Thus, this results in a maximum of 10 frames. The maximum number of bits possible to send can therefore be calculated by using the following formula:

$$10(n_b - 5 - 4))$$

and the maximum data throughput, sending only one frame, is given by:

$$2(n_b - 5)$$

where n_b is the number of bits, dots and segments, in the pattern.

10

Conclusion

We have created a set of methods for reading seven segment displays optically and a protocol for maximized throughput using several proposed techniques, including a minimum set of calibration bits, bit mapping and dynamic sequencing. The methods were compared for both time and correctness. This chapter summarizes the result from the previous chapters, concluding what algorithms and settings that are most suitable for use in a final system. This chapter also discusses what improvements that should be considered for any future work.

10.1 Conclusion of result

From our point of view, the most important aspect of an ASP reader is to have as high a ratio of correctly interpreted pattern as possible. However, the time is also important and especially if the worst case scenario varies a lot from the average time. This is because it would be an issue since the frame rate of the machine is static. However this issue can always be solved by rejecting images if a certain time has been exceeded. However, if the amount of rejected images is too high the application would be very infeasible to use.

Another interesting conclusion is that having Find Color for gray scaling, and thereby introducing more processing in an early stage, increases the overall performance. However it did not increase the amount of successfully interpreted patterns. However we believe this could be improved. Anyway, adding Find color to the algorithm also demands that color settings are predefined when scanning.

Method 2 has the most promising correctness ratio, and is the only method which is usable on LCD displays. It has a worst case time of 380ms on a PC, which is why we propose a frame rate of 500ms in the dynamic sequencing part of the protocol. This allows for some margin when using the algorithm on a smartphone. It could be possible to change this in the future if it is deemed necessary, but it is not our estimation that this is the case in the industry. The messages that might be needed would probably fit in under four sequences on a display that contains two units, thus the time it would take to scan, assuming the scanning works flawlessly, would be two seconds. This, we believe, is an acceptable waiting time in an industrial context.

10.2 Future work

In this section it is discussed how it is possible to improve the application and algorithms further. We will also discuss what requirements need to be fulfilled in order to actually implement the technology in the industry.

Software updates on machines This project has been focused on developing an algorithm required for implementing the functionality of interpreting seven-segment displays as bit-patterns rather than digits using an android smartphone. In order to use this technology it is required that any machine that should be capable generating these patterns have appropriate software for doing so.

Handling of information The application itself may be extended in order to redirect any user to, for example, a website presenting information about, for instance, an error code found within a message. Also, if a message contains information such as an identifier to a serial or product number, the databases needs to be implemented.

Different settings for different devices The device used within the project lacked a lot of capabilities to manually configure the camera even though the *Camera API 2* of device has support for more features. When using other devices, one has greater potential to configure the camera without the requirement for the manual calibration of the camera described in 5.3.2. Depending on the device the application is installed on, it is possible to offer different manual and automatic settings. It is also possible to implement functionality so that devices using the *Camera API 1* can use the application.

Adding support for redundancy or parity control There has been not any focus to implement any functionality for an error-correction mechanism or parity checking. While error-correction may be difficult because the limited number of bits, parity checking of both each pattern and the whole message can be implemented to increase the reliability.

Porting algorithms to native code In order to increase the performance of the application it is possible to port the algorithm to native code, *C++*. Using *Java* for the *GUI* only and perform only one *JNI* when calling the image processing algorithm would significantly increase the performance.

Bibliography

[1] "Computer vision." http://en.wikipedia.org/w/index.php?title=Computer_vision. [Online; accessed 29-April-2015].

[2] "Image analysis." http://en.wikipedia.org/w/index.php?title=Image_analysis. [Online; accessed 29-April-2015].

[3] "Image processing." http://en.wikipedia.org/w/index.php?title=Image_processing. [Online; accessed 29-April-2015].

[4] "Pattern recognition." http://en.wikipedia.org/w/index.php?title=Pattern_recognition. [Online; accessed 4-May-2015].

[5] "EAN-13 and EAN-8." http://www.gs1.se/en/Standards/Capture/ean-13-and-ean-8/. [Online; accessed 07-October-2015].

[6] qrcode.com, "Seven-segment display." http://www.qrcode.com/en/. [Online; accessed 08-October-2015].

[7] T. Langlotz and O. Bimber, "Unsynchronized 4D Barcodes," in *Advances in Visual Computing* (Bebis, George and Boyle, Richard and Parvin, Bahram and Koracin, Darko and Paragios, Nikos and Tanveer, Syeda-Mahmood and Ju, Tao and Liu, Zicheng and Coquillart, Sabine and Cruz-Neira, Carolina and Müller, Torsten and Malzbender, Tom, ed.), vol. 4841 of *Lecture Notes in Computer Science*, pp. 363–374, Springer Berlin Heidelberg, 2007.

[8] "Optical character recognition." http://en.wikipedia.org/wiki/Optical_character_recognition. [Online; accessed 08-October-2015].

[9] I. Bonačić, T. Herman, T. Krznar, E. Mangićm, G. Molnar, and M. Čupić, "Optical Character Recognition of Seven–segment Display Digits Using Neural Networks," 2009. http://morgoth.zemris.fer.hr/people/Marko.Cupic/files/2009-SP-MIPRO.pdf. [Online; accessed 30-November-2015].

[10] E. Vázquez-Fernández, A. Dacal-Nieto, H. González-Jorge, F. Martín, A. Formella, and V. Alvarez-Valado, "A machine vision system for the calibration of digital thermometers," *Measurement Science and Technology*, vol. 20, no. 6, p. 065106, 2009.

[11] R. Ghugardare, S. Narote, P. Mukherji, and P. Kulkarni, "Optical character recognition system for seven segment display images of measuring instruments," *TENCON 2009 - 2009 IEEE Region 10 Conference*, pp. 1–6, Jan 2009.

[12] E. M. Ivan Timofeev, Ilya Paramonov, "Measurement Data Recognition from Seven-Segment Indicator by Mobile Device," *Proceedings of 15th Conference of Open Innovations Association FRUCT* .

[13] E. Auerswald, "Seven Segment Optical Character Recognition - sscor," 2004.

[14] J. Tyson, "How LCDs Work." http://electronics.howstuffworks.com/lcd.htm, July 2000. [Online; accessed 22-June-2015].

[15] T. Harris and W. Fenlon, "How Light Emitting Diodes Work." http://electronics.howstuffworks.com/led.htm, January 2002. [Online; accessed 22-June-2015].

[16] G. R. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly, 2004.

[17] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. West Nyack, NY, USA: Cambridge University Press, 2004.

[18] L. Zeinik-Manor and M. Irani, "Multiview constraints on homographies," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, pp. 214–223, Feb 2002.

[19] S. Pande, V. S. Bhadouria, and D. Ghoshal, "A Study on Edge Marking Scheme of Various Standard Edge Detectors," *International Journal of Computer Applications*, vol. 44, no. 9, pp. 33 – 37, 2012.

[20] R. Deriche, "Using Canny's criteria to derive a recursively implemented optimal edge detector," *International Journal of Computer Vision*, vol. 1, no. 2, pp. 167–187, 1987.

[21] A. K. Cherri and M. A. Karim, "Optical symbolic substitution: edge detection using Prewitt, Sobel, and Roberts operators," *Appl. Opt.*, vol. 28, pp. 4644–4648, Nov 1989.

[22] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32 – 46, 1985.

[23] E. M. Saad, M. M. Hadhoud, M. I. Dessouky, M. E. Elhalawany, and A. M. Abbas, "Fusion of Zernike moments and Fourier-Mellin transform for invariant image resolution," *Optical Engineering*, vol. 47, no. 1, p. 17002, 2008.

[24] "OpenCV matchShapes function." http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#double%20matchShapes%28InputArray%20contour1,%20InputArray%20contour2,%20int%20method,%20double%20parameter%29. [Online; accessed 08-10-2015].

[25] A. Fitzgibbon and R. B. Fisher, "A Buyer's Guide to Conic Fitting," in *British Machine Vision Conference*, pp. 513–522, 1995.

[26] N. Kiryati, Y. Eldar, and A. Bruckstein, "A probabilistic Hough transform," *Pattern Recognition*, vol. 24, no. 4, pp. 303 – 316, 1991.

[27] C. Galamhos, J. Matas, and J. Kittler, "Progressive probabilistic Hough transform for line detection," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 1, pp. –560 Vol. 1, 1999.

[28] S. Allen, V. Graupera, and L. Lundrigan, "Android," in *Pro Smartphone Cross-Platform Development*, pp. 35–50, Apress, 2010.

[29] F. Manjoo, "A Murky Road Ahead for Android, Despite Market Dominance." http://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html, May 2015. [Online; accessed 22-June-2015].

[30] O. Cinar, *Android Quick APIs Reference*. Berkeley, CA: Apress, 2015.

[31] F. Liu, *Android native development kit cookbook*. Birmingham: Packt Publishing, 2013.

[32] N. A. Halli, H.-P. Charles, and J.-F. Mehaut, "Performance comparison between Java and JNI for optimal implementation of computational micro-kernels," 2014. http://arxiv.org/pdf/1412.6765v1. [Online; accessed 30-November-2015].
[33] "Haskell." https://www.haskell.org/. [Online; accessed 8-October-2015].
[34] V. Berthoux, "Juicy Pixels." https://github.com/Twinside/Juicy.Pixels, 2015. [Online; accessed 1-July-2015].
[35] F. Weinhaus, "3D Rotate." http://www.fmwconcepts.com/imagemagick/3Drotate/index.php, 2014. [Online; accessed 1-July-2015].
[36] "Image Magick." https://www.imagemagick.org/. [Online; accessed 31-July-2015].