MASTER'S THESIS

# Optical Character and Symbol Recognition using Tesseract

Victor Ohlsson
2016

Master of Science in Engineering Technology
Computer Science and Engineering

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

Thesis
X7007E

# Optical Character and Symbol Recognition using Tesseract

*Internal Supervisor*
Anders Landström

*Author*
Victor Ohlsson
vicohl-1@student.ltu.se

*External Supervisors:*
Johannes Reesalu
Tobias Olsson

July 1, 2016

Luleå University of Technology
Department of Computer Science

Volvo Cars
Infotainment department

# ABSTRACT

The thesis objective was to examine and evaluate Optical Character Recognition technology for symbol recognition. Can the technology be used to recognize and verify symbols? If so, how good? Today symbol recognition relies on other image registration technologies. The other objective was to provide new functionality of Optical Character Recognition to an existing automation tool at Volvo Cars Corporation. The implementation should be stable, efficient, robust and support various testing functions for the ability to set up automated test cases. The thesis work were conducted in the VU-team at Volvo Cars Corporation, Gothenburg. The working method was agile with two week sprints and constant deliveries.

Symbols could indeed be recognized using Optical Character Recognition, some even very accurately. Results show that symbols which was recognized with a confidence above 73% in perfect conditions could very likely be recognized accurately when exposed to various noise. Symbols were recognized with a confidence above 75% even though the image resolution decreased from 96 to 16 PPI. Tesseract OCR engine were able to recognize most symbols confidently even though they were put through heavy noise filtering, where pixel-to-pixel comparison techniques would fail. Generally the symbols sharing the same outlines were difficult to recognize correctly due to the heavy analysis of symbol outlines. This might be changing when a Neural Network classifier is fully implemented. This could make Tesseract OCR engine a possible candidate for symbol recognition in the future. Symbol recognition using Optical Character Recognition cannot directly replace image registration but the two techniques could complete each other. However, the process to allow symbols to be recognized is very protracted. The thesis implementation result provided a stable and efficient program, it will be used for automated testing to localize, recognize and verify text in Volvo Cars display's interfaces. The implementation is in a JAR file format and can be added as a library in any Java project. This format helps with future development or if the functionality is needed in other projects at Volvo Cars.

# PREFACE

I would like to thank Anders Landström, my internal supervisor for sticking with me all the way. Giving supportive feedback and suggestions from the first day to the last. Julia Uttberg for helping me proofreading the thesis-report through out the thesis.

I would also like to acknowledge and thank Johannes Reesalu and the VU-team for bringing me in to Volvo Cars and giving me the support and help every day. The Journey at Volvo gave me an insight into how the real business works and I have learned quite a lot about company culture.

# Contents

# CHAPTER 1

# Introduction

Infotainment systems in automobiles are a technological feat, providing endless possibilities such as listening to radio, navigating through unknown territories, making phone calls or even starting the seat heaters. The system is there for a reason, which is to simplify and improve the driving experience.

Modern cars usually have a central stack display located between the front seats, the touch-display technology makes the communication between the driver and system extremely easy by a simple touch. Software companies have recognized the potential has began to integrate their own applications to be run inside the infotainment system.

The amount of different interfaces on the display is quickly rising, and with this comes an important responsibility of quality assessment. For the user to understand the current state while navigating through the, system visual feedback is crucial. Improvement in test methods and tools have led to a more extensive use of computer aided image analysis to verify the visual feedback. The infotainment system of Volvo Cars test this today by an in-house developed java-tool: CameraVision. However, the tool lack the ability to read and identify content from images, which is desirable for testing.

Thus, this thesis has been requested by Volvo Cars to implement the functionality of text localization to verify the visual feedback. To accomplish this Tesseract Optical Character Recognition will be used, an engine using the technology which can translate word on images into data strings. Along the implementation, research will be conducted to see if the same technique can be used to verify symbols as well which usually relies on different techniques.

## 1.1 Purpose

The primary purpose of the thesis is to develop an implementation to verify symbols with the help of OCR technology, evaluate the results and compare it to already known symbol verification techniques in image registration. The secondary purpose is to use the implementation to provide a new automation tool functionality for Volvo Cars Corporation for system testing, to localize and verify the text with OCR technology.

## 1.2 Requirements

To understand the implementation design it is important to consider the requirements from Volvo Cars Corporation along with thesis requirements to succeed recognize symbols by OCR.
The requirements are:

- Recognize words and symbols
- Java implementation
- Open source
- Real time analysis
- Support for several languages
- Support for different types of interface themes

To be able to recognize symbols it is essential that the OCR-engine is trainable since symbols is not included in any core language library's. Words must be recognized with a satisfying result and symbols must be able to be recognized.

The already existing automation tool: Camera Vision from Volvo is implemented in Java, which in turn requires the thesis implementation to be implemented in the same language. Enabling the ability for future development in-house.

Open source provides the possibility to distribute and develop freely without any additional costs. It is also preferred to not be dependent on any third party. Today CameraVision is considered to run in real time. OCR and image processing requires heavy computing which makes it difficult to run in real time. The implementation should be built with efficiency in mind to make it as fast as possible. The implementation need to support skew adjustment, meaning that if the image contain a slight skew it should automatically correct it.

The interface on the central stack display comes in different fonts, sizes, colors and languages. To support a global company like Volvo Cars with different languages it is suited to support as many of them as possible. The implementation should also support a dark and bright theme. Se Figures in Chapter 5 for visuals.

## 1.3 Test setup

Once the implementation is finalized it will be used for system testing. Testing will be conducted on a test-rig containing all essential electronic hardware of the Infotainment system. From the thesis perspective the only thing that will be tested is the two displays which can be seen in the left image of Figure 1.1. Where the Central Stack Display (CSD) is in the middle and the Driver in Motion (DIM) can be seen at the top.

There are two image sources in the rig, one camera mounted in front and a frame grabber taking images directly from the image source. The reason for using two different sources is because in some cases the image grabber is not allowed to be used, due to its signals can collide and interrupt others. To work around this an independent image source in form of a camera is used.

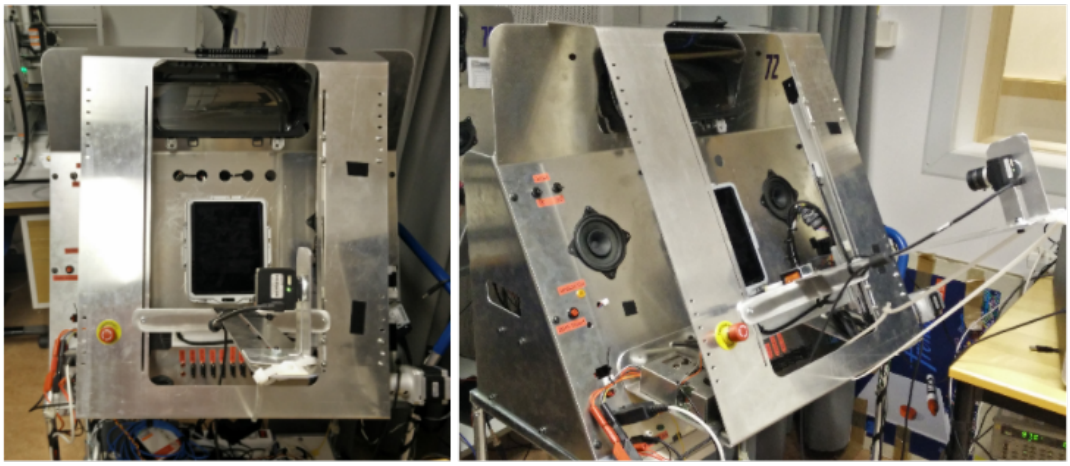


Figure 1.1: *Volvo Cars Infotainment system compressed into a test-rig*

## 1.4 Delimitation

The thesis will not include any implementation of image registration techniques for comparison to the results from Optical Character Recognition, instead they will be compared based on studies of recognizing symbols with image registration.

The thesis will provide an automation tool to support automated testing for Volvo Cars Corporation, no actual testing performed by Volvo Cars Infotainment department will be documented.

## 1.5 Contribution

The thesis will provide an evaluation of the ability to verify symbol recognition using Tesseract OCR , along with an automation tool to be used for system testing and quality assessment.

# CHAPTER 2

# Background

Infotainment systems is a brand new technology considered how long cars have been around, but the idea of an infotainment system have been around for long and slowly developed into today's system. It allows for integrating the phone into the system, listening to personal music or making a call by a speech command. Almost anything can be achieved with a few clicks on the screen, such as: "switching radio source, navigating to an address, checking the weather or even ordering food."

It all began with broadcasting systems which made it possible for manufacturers to install compact shortwave radios in their vehicles. Eventually, the technology to print integrated circuit boards along with small transistors came, making broadcasting tuners to rapidly emerge on the market. Frequency modulated signals(FM) were invented and resulted in an exponential growth in car radios.

Manufacturers began to integrate the radio into a complete car entertainment system, replacing all single devices which were previously used. Digital- information and broadcasting systems replaced previous analogous systems through a digital entertainment system. In the beginning the system were very simple but could provide the option to the driver and passengers to interact simultaneously.

As technology grew, the demand to incorporate it also grew. After the release of the Global Positioning System(GPS) car manufacturers started to integrate navigation and phone abilities turning the previously entertainment system to become a infotainment system.

The number of cars with touchscreen interfaces is growing rapidly, the market in USA increased from 44% to 55% in 2009 [1]. The trend continue rapidly worldwide, with 5.8 million units installed at 2011 and is expected to be 35 million by 2019 [2]. That is an increase of 503% over 8 years.

Touchscreens design guidelines [3] state that visual feedback should be immediate in order to reassure the user that their input have been registered. And the provision of feedback is an essential part of the interaction experience.

*The Gulf of Evaulation* [4] is a degree to which the system provides representation that can be directly perceived and interpreted in terms of the expectations and intentions of the user. Donald Norman state [4] *"The gulf is small when the system provides information about its state in a form that is easy to get, is easy to interpret, and matches the way the person thinks of the system."*

Testing the interfaces in the infotainment system have traditionally been done manually, where testers interact with the infotainment system in a predefined way. Infotainment systems are delivered with different languages, making manually testing an expensive and inefficient procedure. With the help of computers this can be done much more efficient since it speaks several languages. Now, one tester could be responsible to test the interface in different languages both faster and more cost effective than several testers testing each language separately.

## 2.1 Optical Character Recognition

To evaluate the visual text on the images Optical Character Recognition(OCR) [5] is used. OCR was first developed in early 19th century, Optophone [6] was a analog machine and could be used by visually impaired persons to scan text generating different chords and tones to identify as letters. The first digital version of OCR is dated back to the 1970s when Omnifont [7] was released.

OCR is a technology for converting text on images into data strings. The strings can be used for many things but some examples are: to digitize old documents, translate into other languages or to test and verify text positions.

A common use of OCR is in the congestion tax stations in big cities: where vehicles passes the control point a camera automatically will take a picture and send it to an OCR engine. With the help of image processing the engine can identify the registration numbers and invoices can be sent to respective owner. Wherever text can be found OCR can be applied, an example of OCR can be seen in Figure 2.1

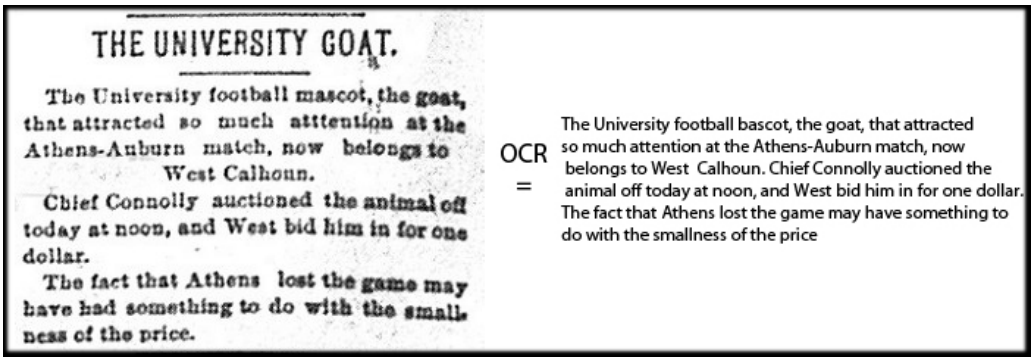



Figure 2.1: *Example of OCR, recognizing words from an old paper and translating them to data stings in a document*

An OCR engine reads the characters in the image and compares them with a database, if a match is found the character and words will successfully be translated into a data string.

There are different OCR engines on the market, some are open source: meaning they are free to use and distribute. While other engines are for commercial use only. They all vary in the capability and performance, where some engines can read and recognize whole words while other only can read characters.

## 2.2 Image Processing

To be able to provide a better and more consistent result from the OCR engine preprocessing is needed. OCR engines usually have very limited image processing [8] internally and need some help. Image processing can help by extracting important features in the image and provide better results. It is helpful for finding and isolating important contents of images, which later can be recognized separately by the OCR engine. It is also useful for detecting and correcting eventual rotation skew.

In general image processing can be used to observe objects which are not visible by the naked eye, sharpening or restoring images into better quality, measuring patterns in various objects, or distinguishing objects in an image.

## 2.3 Image Registration

Symbols are traditionally recognized with Image registration [9]. To evaluate the results from symbol recognition from OCR it needs to be compared to some other technique. The thesis will compare symbol recognition using OCR to already known and used techniques from image registration

Image registration is a general name for different methods including: correlation and sequential methods, fourier methods, point mapping or elastic model-based matching. The

methods assign feature points onto the image to form a coordinate system, this will then be used to match a reference system. If alignment between the two can be found it is considered a match.

The satellite view on Google Maps is a great example of applying image registration. The image appear to be one but have been created with a lot of different images stitched together. To be able to represent the image to appear as one, matching data from different images have been found and stitched together.

## 2.4 Related work

The preprocessing of the image is what matters in order to extract text and symbols with a high confidence. The preprocessing is most often limited inside the engines which is preferable, if too much preprocessing is done to the image it might not come out as the developer expected, or it might be a strategical move to tell the developers to make their own preprocessing before utilize the OCR engine.

A general OCR system, including preprocessing should perform the following tasks [10]:

1. Text Digitization
2. Gray Tone to Two Tone Conversion
3. Noise clearing
4. Text Block Identification
5. Skew Correction
6. Line and Word Detection
7. Character Segmentation
8. Feature Extraction
9. Character Recognition
10. Error Correction

This general order is preferred in order to be sure to analyse an image with good results, but the will vary depending on the circumstances of the images or purpose. The common way to start the preprocessing of an image is to convert it into a gray-scale image before continued preprocessing [11].

However, another technique can be used where color-binarization is utilized to find specific colored areas in an image. It works by setting an upper and lower color shade threshold, for all pixels in the image fitted between the threshold is converted into 1 (white) else 0 (black) [12]. The work by I. Kastelan et al. [13] align perfectly with one of the thesis purposes, to verify extracted text. Tesseract OCR engine is mentioned and present good results.For future work they wanted to improve the algorithm to detect text areas better because problems occurred when the engine interpreted the symbols. Providing an inferior result.

As stated by J.Liang et al. [14] “There is no commercial market to support the additional development of OCR systems that handles special symbols”. The problems arising when implementing such functionality is to make the system into an general solution, if too many

symbols are stored in the database it will cause the efficiency to decrease. The biggest trend in symbol recognition is to be able to analyze mathematical symbols scanning documents. Tesseract have supported mathematical language, where symbols can be recognized. This is a smart an efficient way to utilize the OCR engine to extract mathematical symbols as well as normal characters.

OCR can be used to detect symbols on road surfaces for mapping purposes [15]. The paper state that the procedure to detect road symbols is similar to the process of face recognition (a system designed for recognizing faces). Yet again Tesseract were the engine of choice for the researchers. The road symbol detection experiment provided results with 80% detected symbols where 2.5% of them were incorrectly classified. Considered Tesseracts optimal conditions with high resolution and sharp images, the experiment provided good results even though the camera had poor resolution and the vehicle were moving during the test. The most difficult symbols to be recognized were the more complex symbols with the same outline with different insides. To be able to classify symbols with the same outline with different insides separate analysis were needed. The experiment were only able to distinguish and analyze symbols with different outlines.

## 2.5 Tesseract

Tesseract began as a PhD research project in HP Labs, Bristol. It gained popularity and was developed by HP between 1984 and 1994. In 2005 HP released Tesseract as an open source software [16]. the engine is not trained to recognize damaged data, instead it uses a classifier to recognize damaged characters, to be replaced with complete characters. Meaning the database of training samples could get significantly smaller and more efficient. originally it were trained with 20 samples of 94 different characters from 8 different fonts with 4 attributes ( normal,bold,italic,bold italic), a total of 60160 training samples.

During the OCR engine investigation one engine was mentioned again and again: Tesseract. Except providing accurate results [17], it is also known to have been tested with symbols [15]. The engine looks promising but is not enough to draw any concrete conclusions. Tesseract [18] support UTF 8, and can recognize more than 100 languages out of the box, and the support for more languages is continuing to grow. The engine is trainable, meaning that a new language or font which is not normally supported can be trained and recognized. Tesseract is considered to be the most accurate open source engine available on the market. It utilizes Leptonica image processing library for the basic processing [19]. It is developed and built in C or C++, but can be used by others in other programming languages. It is licenced by the Apache License 2.0 [20].

To be able to work with Tesseract in Java, a third party wrapper is used: JavaCPP along with its different presets [21].

## 2.6 Deselected OCR Candidates

Tesseract were the only open source engine to pass the requirements of Section 1.2. But it is far from the only engine, there are several engines and wrappers to considered in a project. The following engines were studied before being deselected:

- Lookup OCR [22]
  A straight forward java library, activity can be seen on Github. However, it is poorly documented which provides limitations.

- Tess4J [23]
  Similar to JavaCPP which is used, but poorly documented

- Java OCR [24]
  Cross platform support and runs on the BSD-license, making it OK to be used commercially, but also lack documentation

## 2.7 Commercial Engines

Results from 2010 show that Tesseract is not far off compared to commercial products [25], considered it only utilizes the limited internal image processing library. Tesseract engine assume the image is already prepossessed before analysis. while commercial products have included heavy preprocessing of the image. In a more recent test from 2015 show it is now very difficult to distinguish a clear winner [17].

There are a few commercial OCR engines to choose from, but two that are worth mentioning:

- Abbyy
  The biggest commercial OCR software available today. The test from 2010 confirmed it beat all other software available [25]. Abbyy is a whole package deal: it brings a combination of image processing, heuristics and corrections together with the OCR engine [26].

- Asprise OCR
  This stands out because it has support for barcode recognition as well [27], attracting other types industries. It can convert a lot of different images into editable documents.

# CHAPTER 3

# Theory

This chapter will cover some of Tesseracts theoretical details and its capability of character and word classification, its language error rating and training abilities. The chapter will also cover the mathematics of the different algorithms used during the practical implementation. Every implementation were given by OpenCV [28], a computer vision library for image processing.

Terminology:

- Pixel
  - A pixel is a small bit containing the colors: Red, Green and Blue (RGB). An image is built of many pixels with combinations of the RGB intensity to display different color combinations. A resolution of 2480x3508 pixels (A4 paper) means that there is 2480 pixels for every row and 3508 pixels for every column together forming an image of total $2480 \cdot 3508$ pixels.
- Digital image
  - A digital image is computer image represented by pixels. The digital images in the thesis will be used for image processing and word recognition.
- Binary image
  - A Binary image is the same as a digital image but only contains two colors: Black and White.

## 3.1 Tesseract

Papers or articles with updated functionality about Tesseract have not been released since 2007 [16]. However, interesting details has been presented during DAS 2014 conference [29]. Tesseract is now able to recognize several languages simultaneously and can be run in multiple instances in separate threads. Support of OpenCL, an open source platform supporting parallel programming languages that utilize the GPU(graphic card) instead of traditional CPU(processor), has been added. GPUs have 10 times the compute capability compared to CPUs and the most impressing speed-up ratio showcased were from image thresholding, with a speed-up ratio of 20.425 between a GPU and CPU.

An artificial Neural Network (NN) character classifier has been implemented as an alternative recognizer, this alternative provided an 50% reduction in word recognition errors in the Hindi language. However, it was disappointing for other languages with a few percent reduction and running 3 times as long. The Neural network word classifier files can be found in the English training data indicating NN development is in progress with alphabetical languages.

Artificial NN is inspired by the how the biological nervous systems works, such as the brain and how it process information. Like humans, NN learns by example. It consist of a large number of highly unified processing elements (the neurons) working simultaneously to solve specific problems. A NN system is configured for a specific application, in this case OCR. But could be for other pattern recognition or data classification applications as well [30]. Using NN to increase recognition rate of characters have successfully been tested [31] and shows character recognition rates of 97%-99%. *"In a user environment the system is not constrained by factors like font dependence and need for perfect alignment of text material which makes this system a practical feasibility"* [31]. Since NN is not constrained by factors such as font dependence it can be a indication that this technology can be highly successful for symbol recognition.

### 3.1.1 Architecture

Tesseract analysis architecture is built in an iterative pipeline process, except it revisits old steps. The recognition is done twice; during the first recognition run a static classifier is used and for an second the adaptive classifier is used. Tesseract is designed to recognize text even if it have a small skew without having to deskew the image, even though it is preferable to have the text horizontal for better recognition. The first part of the recognition process is the connected component analysis. It includes line finding, baseline fitting, character and word segmentation. Then every word passed in the static classifier is fed into the adaptive classifier for training. During the second recognition the adaptive classifier is used and words not previously recognized by the static classifier could now possible be recognized see Figure 3.1.

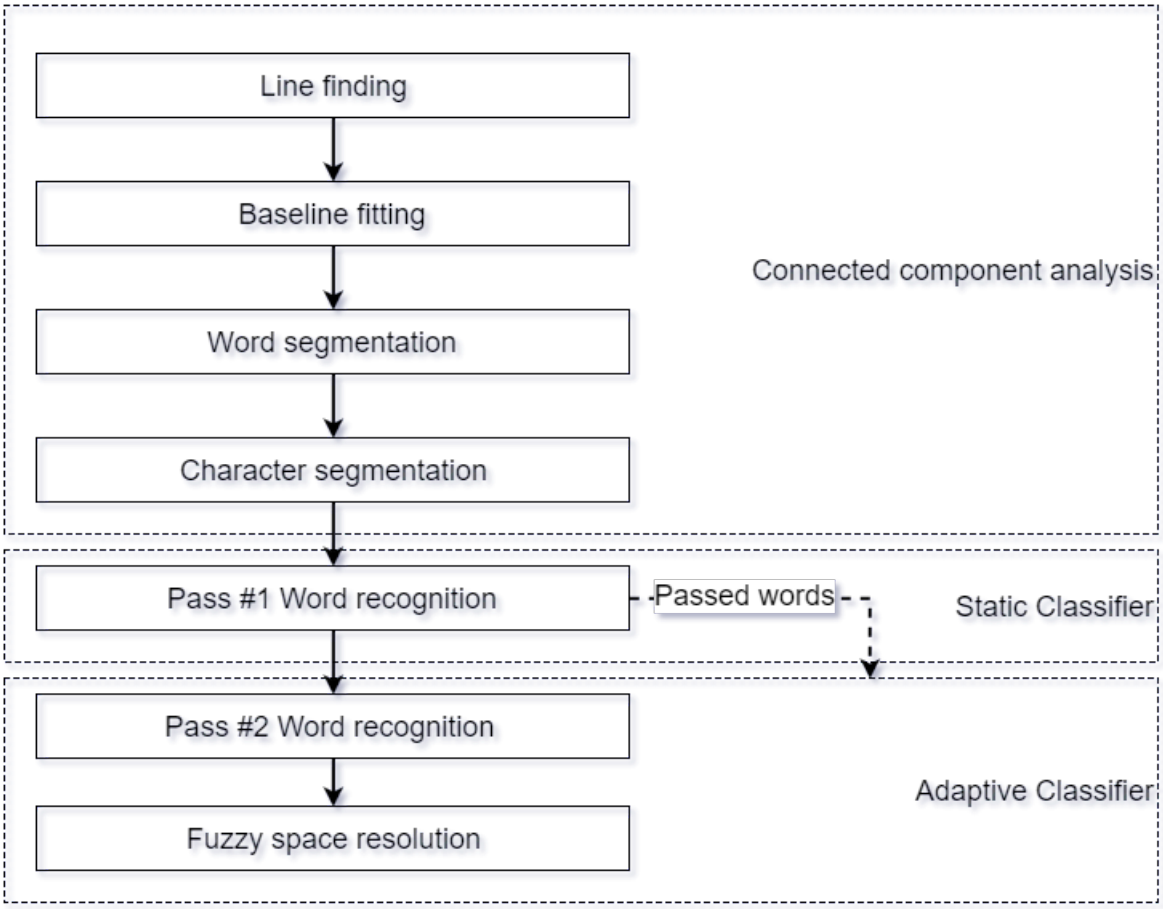



Figure 3.1: *Tesseract component architecture*

### Finding the baselines

The key ingredient to baseline finding is blob filtering and line construction. A blob is a word, or a symbol or any content not connected to the rest of the image (Figure 3.2). The mean height of the blobs are approximated, helping the engine to filter out and remove all small blobs, typically punctuation or eventual noise.


Figure 3.2: *An example of finding two baselines from the 7 blobs lines, one for each word.*

The blobs are likely to fit without overlapping and be parallel with the same slope. The blobs are not necessarily recognized after another in the correct order. To keep track of where the blobs should be positioned they are assigned an x-coordinate. This reduces

eventual ill-effects of assigning an incorrect text when skew is present. Once every blob is assigned a line a least median of squares fit is used to approximate the baselines. The blobs are then fitted back into their assigned lines [16].

### Baseline fitting

When the lines of blobs have been found Tesseract examines them a little more closer. Baselines are fited more precisely with a quadratic spline, i.e four parallel lines that analyses the blob. This feature is very useful to help Tesseract handle curved words; e.g scanned books where the words most often is curved in the center near the book bindings.

### Word segmentation

Words with characters sharing the exact same width (*fixed pitch*) are treated as a special case, the words is sliced equally based on the pitch and marked for recognition. However, most often do characters in words have different pitches and have to be treated separately which can be seen in Figure 3.3.


Figure 3.3: *An example of a word with almost a fixed pitch(n). Only the letter i's pitch differentiate from the other characters pitch.*

Tesseract handle different pitches by measuring the gaps in a limited vertical range between the baseline and mean line [16]. When it find spaces that are too close to the threshold it marks the space as a *fuzzy* space. Tesseract then passes it off for a decision to be made later during the second recognition when the adaptive classifier might have learned more useful information.

### Character segmentation

Tesseract tries to resolve the segmentation of characters by chopping the blob that was assigned the worst confidence by the character classifier. Potential candidate chop points are found from concave vertices of a polygonal approximations of the outlines [16] (see Figure 3.4 below).

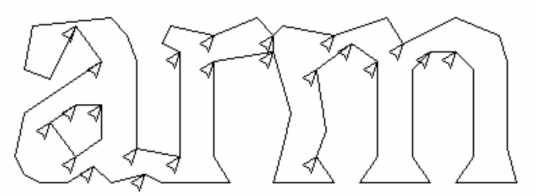


Figure 3.4: *Candidate chop points in the blob [16]*

The chops are classified in a prioritized order. Any chop failing to improve the confidence of the result will be undone, but not completely discarded. It will be analyzed once more by the broken character associator

### Associating broken characters

If all the potential chops have been tried and the word is still not good enough it is given to the *associator*. The associator tries different candidate chops from a prioritized queue and evaluating them by classifying unclassified combinations of fragments. R. Smith states that this approach is at best inefficient, at worst liable to miss important chops. But the advantage is that words missing vital pieces of fragments still could be recognized [16], an example of can be seen in Figure 3.5.

Figure 3.5: *A word easy recognized thanks to the associator approach [16]*

### Character Classification

Character classification is divided into two passes, *pass 1* and *pass 2*. The first pass is through the static classifier, and the second is through the adaptive classifier.

The first step includes a pass through the static classifier, the outlines are extracted from the character to be recognized. Different sized fixed-length features are extracted from the outlines and matched many-to-one to a reference from the training data. Even though character outlines sometimes lack specific details to directly be verified with a reference the engine can utilize polygonal approximation to match associated broken characters to a reference (see Figure 3.6).

Figure 3.6: *The static character classifier can recognize complete and broken characters [29]*

Every character successfully matched by the static classifier is sent into the adaptive classifier for training. This training is essential for the second step of the recognition process where it will use the general information gathered from the static classifier to help it match the previously failed characters. All the *fuzzy* spaces along with the words are now considered by the adaptive classifier, once a character passes the adaptive classifier it is safe to consider it a match.

### Word Classification

Tesseract support the functionality to recognize words as well. It is done by separating the characters in a word, and then stitching them back together one at a time. While stitching the characters together the algorithm constantly compares the current word to a language model. If current character fails to improve the confidence of the word the result is undone. If the highest confidence is high enough it is considered to be a complete word. A visual example from DAS 2014 [29] is displayed in Figure 3.7.

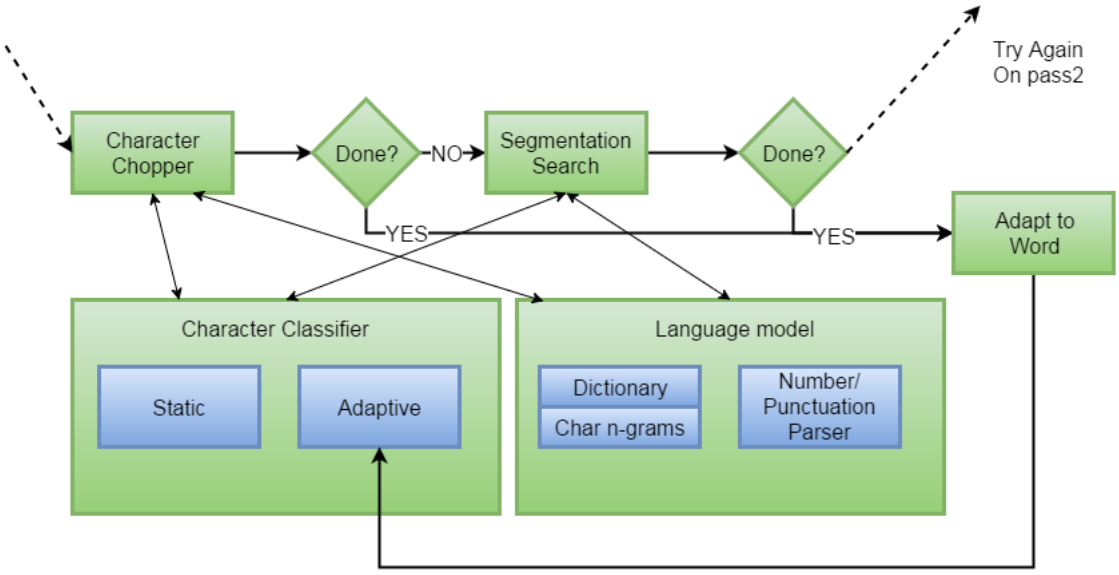



Figure 3.7: *Tesseract Word recognizer [29]*

### 3.1.2 Languages

Tesseract support many languages (see Appendix A.1 for full details) and it is easy to assume the statistics of recognition share the same errors disregarding of language. Table 3.1.2 below shows differently, error rate differ largely depending on the chosen language. The recognition is dependent on its training data, the more data a language have the better accuracy of the recognition. The OCR engine is developed and tested with the English language, making it trivial and rather easy to learn other languages with same alphabetic letters. But languages that does not share the alphabetical letters can suffer as we see in figure below.

| Language | Char Error Rate % | Word Error Rate % |
|---|---|---|
| English | 0.47 | 6.4 |
| Italian | 0.54 | 5.41 |
| Russian | 0.67 | 5.57 |
| Simplified Chinese | 2.52 | 6.29 |
| Hebrew | 3.2 | 10.58 |
| Japanese | 4.26 | 18.72 |
| Vietnamese | 6.06 | 19.39 |
| Hindi | 6.43 | 28.62 |
| Thai | 21.31 | 80.53 |

Table 3.1: *Tesseract error rate result from 2014 [29]*

Even though the error rate is pretty low over all, the trend seems to be that Tesseract have a stable recognition when recognizing alphabetic letters. It is safe to say that the error rate is successively increasing as other non-alphabetical languages are recognized. This could possibly mean that the languages lack training data for more accurate results, but it could also be that the engine is not capable of classifying the "symbol-like" letters from other languages. As previously stated: using the new *Neural Network*(NN) classifier the Hindi language error rate were reduced by 50%. However, this is not a direct indication that Tesseracts' classifiers is poorly designed; it could be a potential flaw in the the chopping algorithm where it might be very difficult to chop Hindi words, which have a very different layout design compared to alphabetic letters.

The biggest difference between languages and symbols is that characters tend to have unique outlines without anything inside. Where symbols can share outlines with complex insides. This indicates that Tesseract will probably have difficulties recognizing symbols sharing the same outlines, but will successfully recognize unique outline symbols.

### 3.1.3 Training

A great functionality that Tesseract posseses is the ability to learn new fonts and new languages. It can be custom made to recognize a particular handwriting or new languages

not yet supported. Figure 3.8 displays a sample of Volvo's symbols trained by the engine but unfortunately were not able to be recognized.

The training fundamentals of Tesseract consists of [29]

- Character samples must be segregated by a font
- Few samples are required; 4-10 of each combination is good but one should be sufficient
- Not many fonts are required
- The number of different characters is limited only by memory

The first bullet explains that the engine require a font to be trained. Thus, to be able to recognize symbols with Tesseract an conversion of vectorized symbols into a font is required. The process is done by the following steps:

1. Convert symbols into vectorized symbols
2. Combine all vectorized symbols into a truetype font
3. Create a tif image with the symbol font exposing the different symbols
4. Create a box file containing the data representation of each symbol from the tif image. The box file is essential for the training process to understand which symbol correspond to what data.
5. Generate a language training data file and add it into Tesseracts language folder


Figure 3.8: *Example of Volvo interface symbols converted into a font to be trained with Tesseract*

## 3.2 Hough Line Transform (HLT)

The theory of *Hough Line Transform*(HLT) is that any point in a binary image could be a part of a set of possible lines, together creating a distinct shape.

Before applying the HLT the most common preprocessing stage is to apply an edge detector where each point is mapped in an image. The edge detector can miss points, or add disjoint points elsewhere from pixel noise making shapes "unconnected". The HLT addresses this problem by grouping edge points into possible object candidates by a voting procedure.

Hough Lines in the Cartesian coordinates can be expressed as:

$$y = kx + m \tag{3.1}$$

where $k$ is the gradient, $m$ is the intersection point in y-axis.

Consider a single isolated edge point $(x, y)$ in the image plane, meaning there is an infinite number of lines that could pass through this point. Each of these lines can be characterized as the solution to some particular equation. Now let $x$ and $y$ be constants(i.e. a fixed point) instead and let $m$ and $k$ be the variables. One can think of it as a transformation from $(x, y) - space$ to $(k, m) - space$.
The equation 3.1 above can now be expressed as:

$$m = -kx + y \tag{3.2}$$

A visual example of how the equation is used is displayed in Figure 3.9.

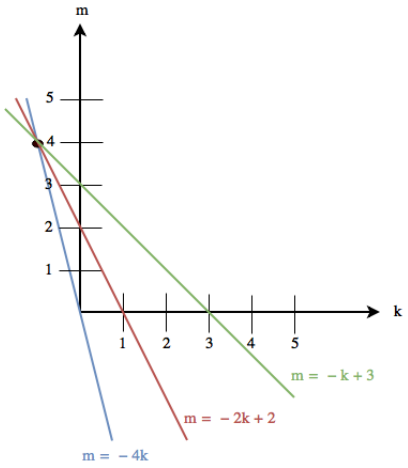



Figure 3.9: *Three lines plotted in the* $(k, m) - space$ *all passing through the fixed point* $(-1, 4)$

A line the $(x, y) - space$ in Cartesian coordinate system can also be represented by a point in the $(r, \theta) - space$ from the Polar Coordinate system which can be seen in Figure 3.10.

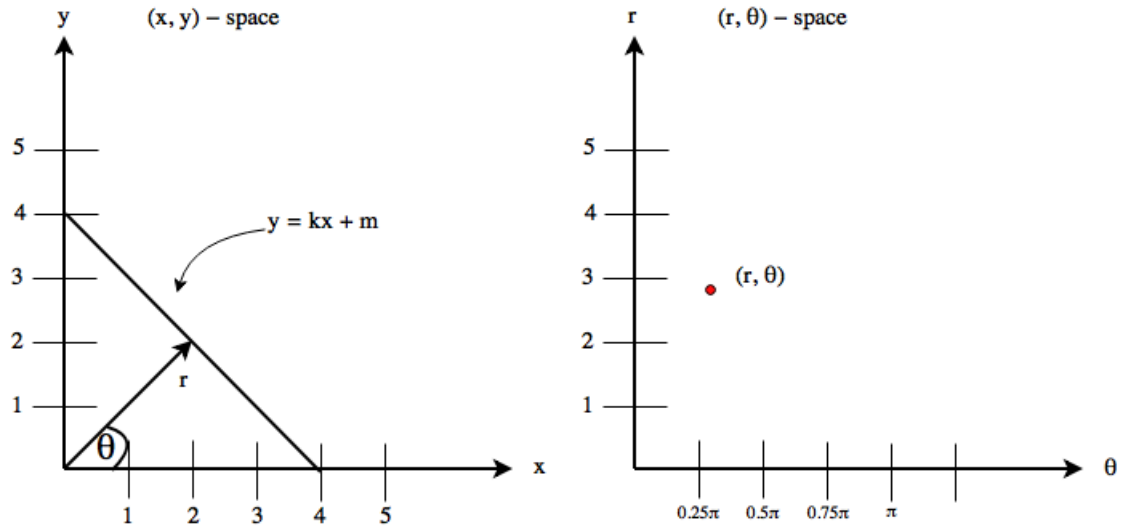



Figure 3.10: *A line in the* $(x, y) - space$ *represented by a point in the* $(r, \theta) - space$

The problem with Hough transform in *Cartesian coordinates* is that vertical lines can not be represented as $k \longrightarrow \infty$. *Polar Coordinates* is used to solve this issue, where each line represent an angle $\theta$ and the perpendicular distance to the origin $r$. Where $\theta$ is bound by $[0, 2\pi]$ and $r$ is bounded by the diagonal of the image.

However, another problem remains: for any single point, infinitely many lines can pass trough it if the $(r, \theta) - space$ is continuous. To restrict the endless possibilities of different sets of angles, the angles is addressed in specific amount to be computed: $\theta = \theta_1, \theta_2, ....\theta_n$ the parameter can now be described as:

$$r_i = xcos\theta_i + ysin\theta_i \tag{3.3}$$

Each $(r, \theta)$ point in the image compute the $r, \theta$ values and increase the values of the corresponding element of the accumulator matrix. Once the process is completed for every point, the elements in the accumulator matrix having the highest values will correspond to the lines in the image. A visual example can be seen in Figure 3.11.

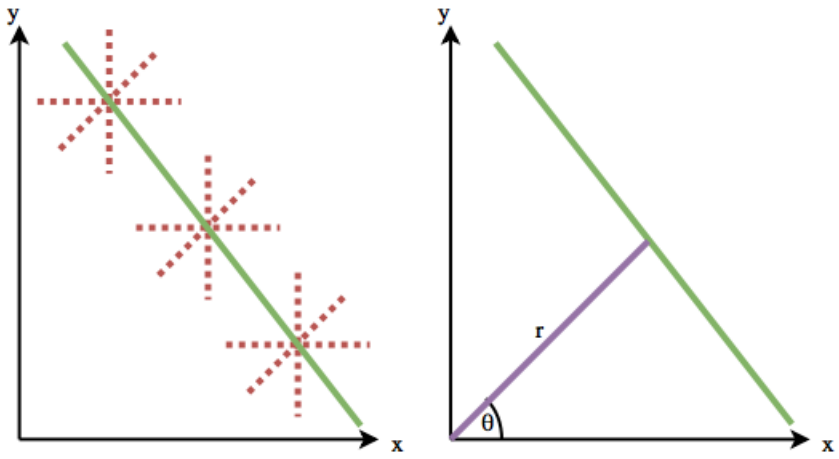



Figure 3.11: *All three points have one $(r, \theta)$ pair in common. This $(r, \theta)$ entry of the accumulator matrix is 3, Remaining lines in the accumulator only have the value one and is therefore ignored.*

## 3.3 Minimum-area bounding rectangle (MBR)

The idea of the *Minimum-area bounding rectangle*(MBR) is to encapsulate every component from the image into a bounded area. If the bounded rectangle contains an angle in respect to the $(x, y) - axis$ then the image most probably contain a skew. See Figure 3.12 for an example.

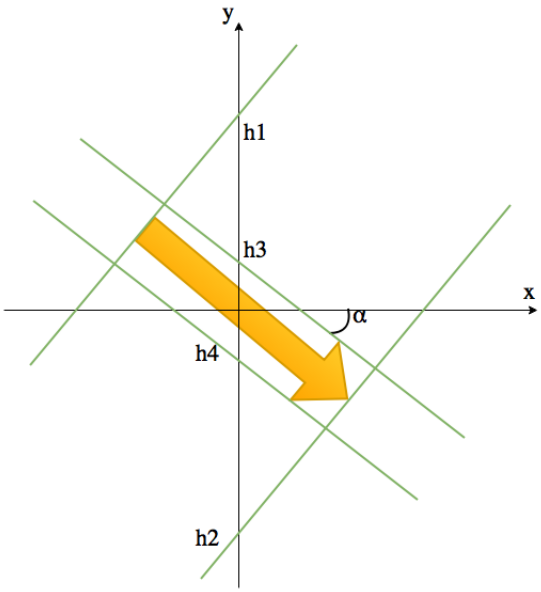



Figure 3.12: *Calculation of the rotation angle by minimum-area bounding rectangle algorithm*

The MBR algorithm is defined by following steps [32]:

1. Initialize the angle $\theta$ to zero, and the minimum area $A_{min}$ to the current area of the bounding rectangle of the connected component at current angle $\theta$.
   Initialize $\alpha_{min}$, $\alpha_{max}$ and $\Delta\alpha$ to their desired values. Consider the origin at the center of the rectangle

2. For different values of $\alpha$, from $\alpha_{min}$ to $\alpha_{max}$ with a resolution of $\Delta\theta$ and $\alpha \neq 0$ repeat step 3 and 4.

3. The area $A$ of the bounding rectangle in the $\alpha$ direction can be calculated by:

$$A = (h_1 - h_2) \cdot (h_3 - h_4) \cdot |cos\theta \cdot sin\theta| \tag{3.4}$$

   Where $h_1$ and $h_2$ are the maximum and minimum intercepts from origin of the lines having a slope of $\tan\alpha$, and passing through any boundary pixel of the connected component.

   $h_3$ and $h_4$ are the maximum and minimum intercepts from origin of the lines having a slope of $-\frac{1}{\tan\alpha}$ and passing through any boundary pixel of connected component. component.

4. if $A < A_{min}$, set $A_{min} = A$ and $\alpha = \theta$

5. The rotation *angle* of the connected component equals to the skew angle $\alpha$

## 3.4 Affine Transformation (AT)

The *Affine Transformation*(AT) is a general basic algorithm which contains geometrical operations of translation. The thesis implementation only encounter the geometrical rotation operation which is defined by as:

Clockwise :

$$R_\theta = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.5}$$

Counter clockwise :

$$R_\theta = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.6}$$

Where $R_\theta$ is the rotation matrix by $\theta$. This is a fitting scenario because the image format itself. It is in a matrix format, meaning that image is the rotation matrix.

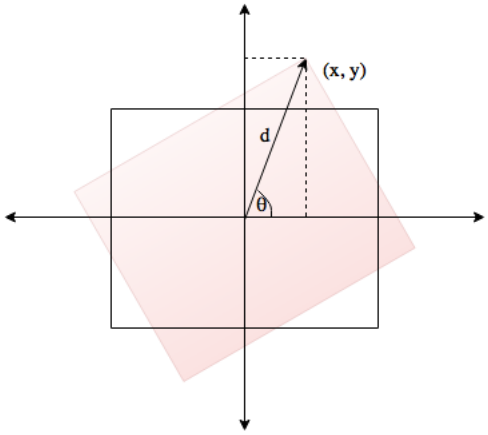



Figure 3.13: *An image rotation by $\theta$*

## 3.5 Canny Edge Detection (CED)

The *Canny edge detector*(CED) was developed by John F. Canny in 1986, the detector may also be referred to as the optimal detector or canny algorithm [33].

Edge detectors is an essential part of many computer vision systems. It serves to process and simplify the image analysis by drastically reducing the amount of data to be processed, but at the same time preserving useful structural information about objects.

Figure 3.14: *A telephone symbol exposed to the Canny Edge Detection algorithm*

The CED algorithm is built on following steps:

1. **Filter out any noise in original image**

   It is done with a Gaussian filter. The filter will blur the images and remove any unwanted details or noise by calculating a suitable 5x5 Gaussian kernel with standard deviation 1. Every point in the image will be convolving with the kernel, and then summing them all to produce a new smoothened image.

The convolution is defined by:

$$f(x,y) * g(x,y) = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} f(i,j) \cdot g(x-i, y-j) \tag{3.7}$$

Using the 2D kernel:

$$g(x,y) = \frac{1}{\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.8}$$

where f(t) the image g(t) is the Gaussian kernel

where $x$ is the distance from the origin in the horizontal axis, $y$ is the distance from the vertical axis, and $\sigma$ is the standard deviation of the Gaussian distribution

2. **Find the intensity gradient of the image**
This is done with a procedure identical to Sobel [34]. The Sobel operator performs spatial gradient measurement on an image. By applying a mask on the image to find prominent vertical edges. It works like a first order derivate and calculates the differences of pixel intensities inside a edge region.

   The formula for Gradient strength($G$) and direction ($\theta$) is described by:

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.9}$$

$$\theta = \arctan(\frac{G_y}{Gx}) \tag{3.10}$$

   $\theta$ is rounded to one of the four possible angles: 0°, 45°, 90° or 135° to enhance the edges.

3. **Apply non-maximum suppression**
Removal of all existing pixels which is not considered to be part of an edge. Thus, only the lines/edges will remain in the binary image.

   Let $G(x,y)$ be the gradient strength at the edge position (x,y). If the strength is largest, then preserve the value of the edge. Otherwise suppress it into a black edge.

4. **Apply Hysteresis**
Eliminating all streaking, breaking of edge contours caused by fluctuating above and below the given threshold $[T_{low}, T_{high}]$

   - If $G(x,y) > T_{high}$, the edge is rejected
   - If $G(x,y) < T_{low}$, the edge is rejected
   - If $T_{low} \leq G(x,y) \leq T_{high}$, the edge is accepted if connected edge $G(x_{conn}, y_{conn}) > T_{high}$

## 3.6 Morphological filtering

Morphology means "study of shape" in greek. In image processing it is used as a tool for extracting useful image components by a describing image shape. The binary images may contain noise or imperfections and the goal is to remove them by accounting a reference form.

The technique utilize an structuring element, a small shape. The element will be positioned on every pixel in the image and compared with neighboring-pixels, if it "fits" inside the neighborhood changes apply on the image or if it "hits", meaning that the SE intersects with the neighborhood changes is rejected and nothing changes. The thesis implementation only encounter binary morphology. However, it can be applied to gray scale images as well.

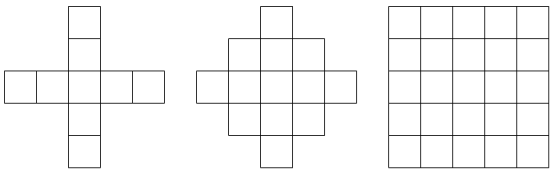


Figure 3.15: *Different structuring elements, Cross, Diamond and Square*

### 3.6.1 The fundamental operators: Erosion and Dilation

Erosion shrinks the contour in an image by stripping away pixels from the inner and outer boundaries. Holes between regions become larger, and small details gets eliminated.

The erosion of an binary image $A$ by the structuring element $B$ is defined [35] by:

$$A \ominus B = \{z \in E | B_z \subseteq A\} \tag{3.11}$$

where $B_z$ is the translation of B by vector z:

$$B_z = \{b + z | b \in B\}, \forall z \in E \tag{3.12}$$

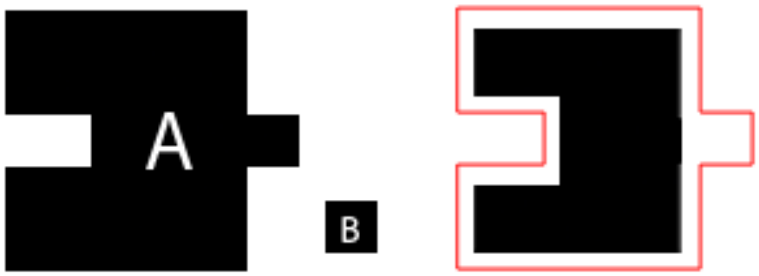



Figure 3.16: *Binary Erosion of an image A by a structuring element B*

**Dilation** is the opposite to erosion, it increases the size of the contours by adding layers of pixels on the inner and outer boundaries. Holes between regions now become smaller, and small details is enlarged.

The Dilation of an binary image $A$ by the structuring element $B$ is defined [35] by:

$$A \oplus B\{z \in E|(B^s)_z \cap A \neq \emptyset\} \tag{3.13}$$

where $B^s$ denotes the symmetric of B:

$$B^s = \{x \in E| - x \in B\} \tag{3.14}$$

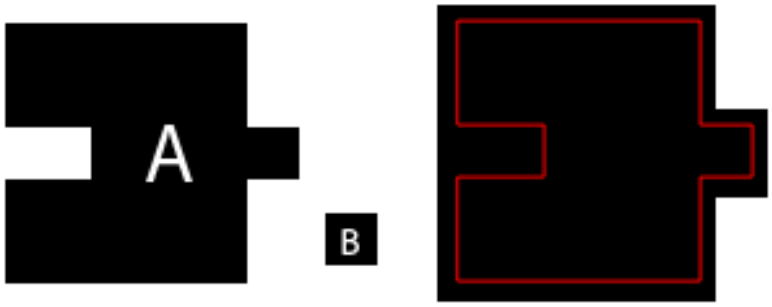



Figure 3.17: *Binary Dilation of an image A by a structuring element B*

### 3.6.2 Opening

Opening generally smooths the contours, breaks narrow areas and eliminates protrusions. The parts that will remain in the image is where the SE hits properly, else when the element fits, content will be removed. The opening of A by B is obtained by erosion of A by B, followed by dilation of the resultning image by B:

Opening of an binary image $A$ by the structuring element $B$ is defined [35] by:

$$A \circ B = (A \ominus B) \oplus B \tag{3.15}$$

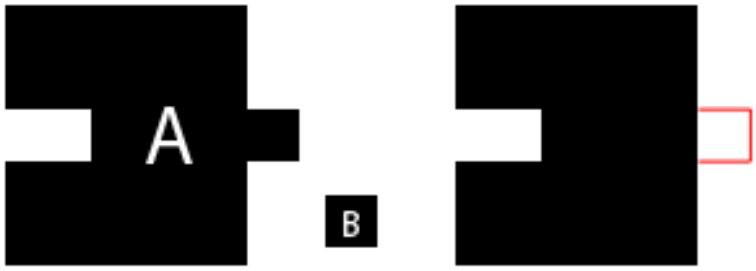



Figure 3.18: *Binary Opening of an image A by a structuring element B*

### 3.6.3 Closing

Closing also tend to smooth the contours, but as opposed to opening, it generally fuses narrow breaks and long thin gulfs, eliminates small holes and fill gaps in the contour.

The closing of the binary image A by the structuring element B is done by first dilate set A by B, then erode the result by B.

Closing of an binary image $A$ by the structuring element $B$ is defined [35] by:

$$A \bullet B = (A \oplus B) \ominus B \tag{3.16}$$

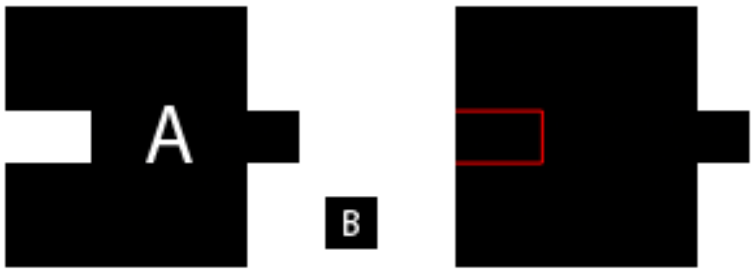



Figure 3.19: *Binary Closing of an image A by a structuring element B*

# C H A P T E R 4

# System Design

This chapter covers an overview of the system with the connected components. It also covers an implementation design in form of a state chart diagram.

## 4.1 System Overview

The thesis implementation will be added as a dependency to CameraVision, meaning it will be a stand alone library in form of a Java Archive (JAR). This design choice gives an advantage to be able to extend the implementation to us it for other projects in the future, and not to bind it into the current automation tool.

Testers use CANoe which is an external testing software to set up test-cases to send instructions to CameraVision. CameraVision receives these instructions and proceed to handle them internally. It has a wide range of different features to verify different test cases, from the thesis perspective the only important functionality is the OCR functionality. Thus, only the thesis implementation is displayed as a connected component to CameraVision. See Figure 4.1 below.

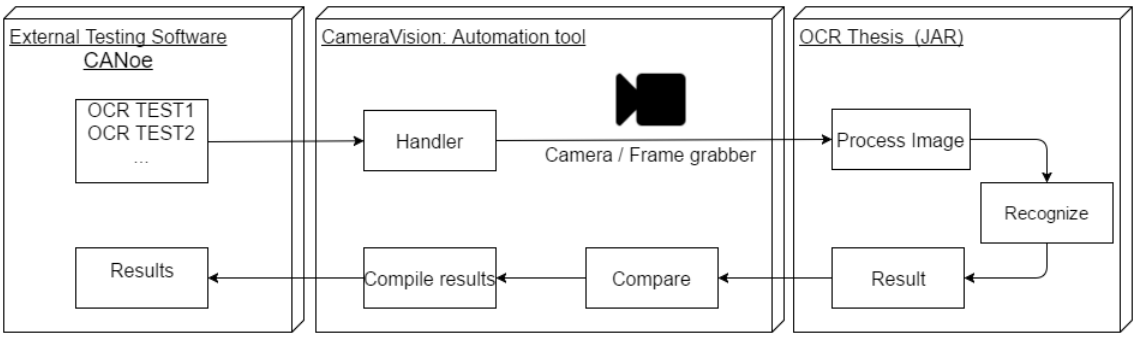



Figure 4.1: *The OCR implementation is connected to CameraVision which in turn is connected to CANoe testing software.*

Different OCR test cases from CANoe send instructions to CameraVision, the handler process the instructions and address them into the thesis implementation. The thesis implementation receives the request along with an image which in turn breaks down the image by recognizing text and symbols. The recognition is compiled into a result class and sent back into CameraVision where the OCR result is compared with reference data. The comparison is then compiled and is sent back into the CANoe testing software.

## 4.2 Implementation Overview

The thesis implementation is built dynamically to fit the purpose of symbol recognition as well as for word verification. In the implementation there is no way to programmatically distinguish between a word or a symbol which gives the advantage to use it as multi-purpose tool. If the only purpose were to provide an evaluation of symbol recognition using OCR, the implementation would be too complex. However; since the second purpose is to support Volvo with new efficient OCR functionality for testing, the implementation requires the complexity (see Figure 4.2).
To start the program the function *initialize* needs be called. The call require three parameters to proceed: what language to be recognized, if a camera is used as an image source and finally a calibration image to calculate eventual skew and interface theme.

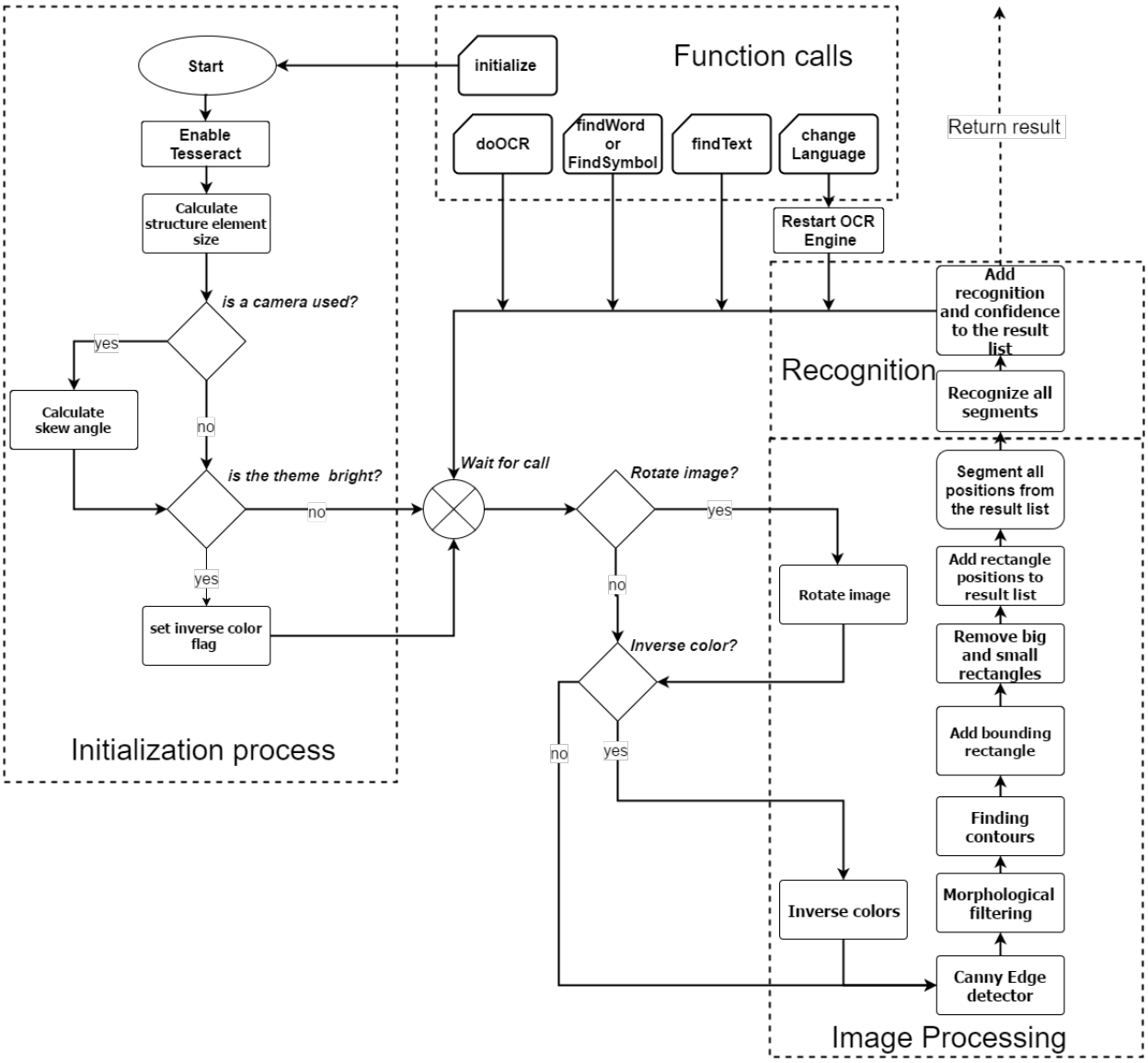



Figure 4.2: Statechart diagram of the OCR implementation

### Initialization process

The initialization process starts by enabling Tesseract, followed by calculations of the Structuring Element (SE). The SE is calculated from the image resolution and will be used later for morphological filtering. If a camera is used a skew angle of the image is calculated. If the theme of the interface is bright a flag will be set to ensure that the colors will be inverted later. The reason for the flag is because the image processing in the implementation requires white content on black background. Once the initialization process is completed the program is put to a halt and will wait for a new function call.

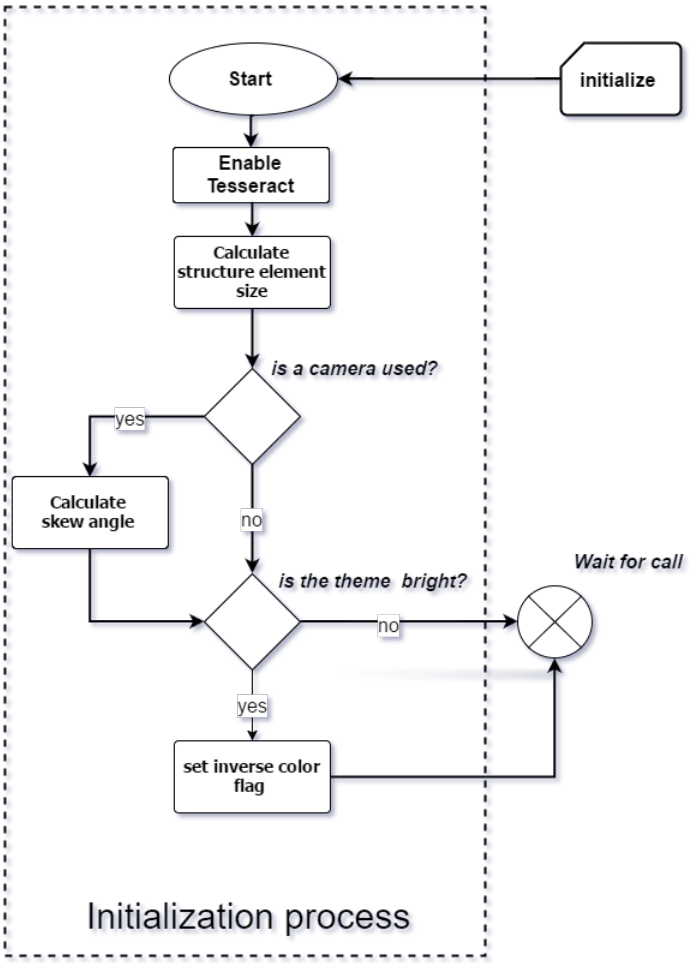



Figure 4.3: *The initialization process*

### Function calls

There are 5 functions calls available in the implementation depending on what you want to achieve. The first function is *initialize* which will start or restart the **initialization process**.

The second is *doOCR* which is used to scan an image for all words and symbol. This function is preferred for efficiency reasons to be used if there are several tests of one specific image. The third is *findWord*(findSymbol) and is used to verify if one specific word or symbol can be found. The fourth is *findText* and is equal to *findWord*, the only difference that it searches for several words instead of one only. The fifth and last function call is *changeLanguage* which can be used to change the current recognition language without having to start the **initialization process**.

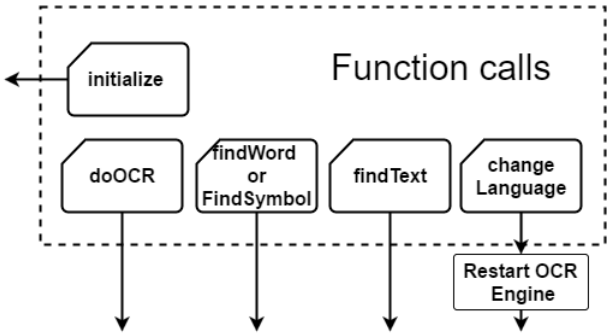



Figure 4.4: *The implementations function calls*

### Image Processing

Once the program receives a function call it proceeds to check if the image needs to be rotated. It is done by checking the previously calculated skew angle. If an angle is detected the image will be rotated, otherwise it will continue. Next stop is to check if the image needs to be inversed. This is done by checking the inverse color flag previously calculated. If it is flagged to be inverted it is put through a invert color function, otherwise it will continue.

Now the image has arrived into the *image processing* block where the objective is to extract blobs of information and transform them into segments. The first function is the *Canny Edge Detector* (CED, see Sect 3.5) where the essential edges will be found and background noise is removed. Next up is the *morphological closing* filter where the SE will be used. The *closing* algorithm will connect characters and symbols into blobs. These blobs are then used to extract contours of every blob. This allows for the next function: *bounding rectangle*. The function will bound all contours in the image by bounding rectangles. Each rectangle hold the position where each segment is located. A *filtering function* removes all rectangles which are significantly large or small. Segments containing a word or a symbol is extracted from the image with the help of the remaining rectangles and sent into Tesseract for recognition.

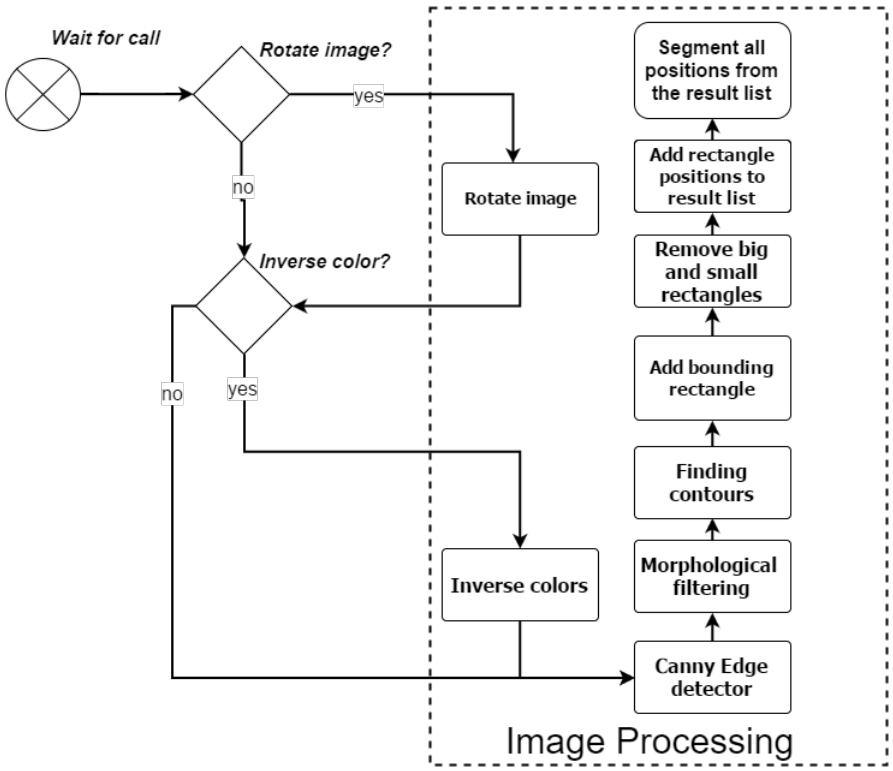



Figure 4.5: *Image processing*

### Recognition

The recognition result will be returned along with the confidence from Tesseract and represent the grade of the recognition. The result will be inserted into a result list containing information about all the recognized words and symbols: what word were recognized, the recognition confidence and position in the image. The result list will then be used to deliver a final answer. After the result list had been built, the implementation return to the waiting state where it will wait for a new function call to restart the process.

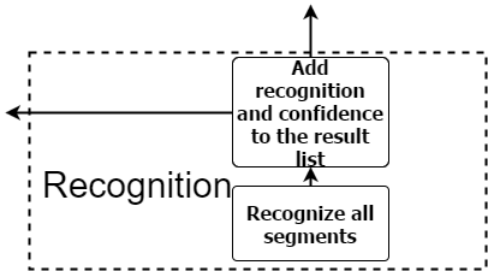



Figure 4.6: *Recognition block*

# C H A P T E R 5

# Implementation

This chapter covers the implementation process of the system. It is divided into four sections: The initialization process, function calls, image processing and recognition.

## 5.1 Initializing Process

To keep the implementation efficient by not recalculating the skew angle or the dynamic parameters for every new image the following assumptions were established through out a test seesion:

- The image resolution will be the same
- The theme will remain the same
- The camera skew will stay the same

If any of these assumptions are not followed the implementation will not work as intended. To resolve this issue and add support for other types of interfaces, the initialization process can easily be restarted by calling the initialize function again.

The first step is to start Tesseract OCR engine, it is done by executing and starting a new thread running Tesseract separate from the implementation thread.

### 5.1.1 The Morphological Structuring Element

The morphological filters utilize a rectangular Structuring Element (SE). The SE is consist of a horizontal and a vertical parameter. To produce a satisfying result the size of the SE needs to be changed dynamically depending the image resolution.

If the resolution width is greater than the height it is considered to be a wide image and the program will use a wide SE ($SE_{wide}$). Otherwise the image is considered to be tall and a tall SE ($SE_{tall}$) will therefore be used. The tall SE have a bigger vertical parameter in order to match the image. While the wide SE have a smaller vertical parameter.

To find the satisfying dynamic parameters a reference image from Volvos interface with the resolution of 768x1024 was used. A rectangular SE of the size 7x3 pixels gave a satisfying result. A satisfying result means that the closing filter connect characters or symbols-segments together forming a blob containing a word or a symbol (see Figure 5.4 )

Testing another image with the resolution of 960x1280 gave a satisfying result with a rectangular SE of size (8,4) Testing continued with different resolutions until the following formula was established:

$$SE_{tall} = Size\ (6 + ratio, 3 + ratio) \tag{5.1}$$

$$SE_{wide} = Size\ (8 + ratio, 1 + ratio) \tag{5.2}$$

where the ratio is described by:

$$ratio = \left\lceil \frac{width_{image}}{width_{reference}} \right\rceil \tag{5.3}$$

### 5.1.2 Skew detection

Good recognition require straight horizontal lines of words and symbols, increasing skew angle means an increasing recognition error rate. Thus, if a skew angle is detected it needs to be handled. There are two types of algorithms implemented to detect the eventual skew. The first algorithm, *Hough Line Transform* (HLT); is applied on the image where natural edge lines can be found. The second algorithm *Minimum-Bounded Rectangle* (MBR); is used as a fail-safe option if HLT fails to find any lines.

#### Hough Line Transform

Due to the characteristics of Volvo's interface HLT is a good choice for calculating the skew angle. HLT works by extracting lines from defined shapes found in the image. Any lines that are not approximately horizontal ($\pm 10°$) will be removed.

The first step is to apply CED on the original image with a low threshold to find and enhance edges in the image. The HLT algorithm is applied to find the lines on the new enhanced shaped image. Once the lines have been found the angle is calculated by taking the average of all the line angles($\theta$). $\theta$ is then saved to be used later during image processing to correct the image rotation. A visual example can be found at Figure 5.1.


Figure 5.1: *An example of utilize the HLT algorithm for finding a skew angle. The original image (1) is put through CED (2) which is used for finding the Hough Lines (3). The image is rotated by the angle calculated from the Hough Lines(4).*

#### Minmal-area Bounded Rectangle

The *Minimal-area Bounded Rectangle* (MBR) algorithm is applied as a fail-safe option to create a general skew detection solution. It is important to not only use the HLT algorithm for skew detection since in some cases it can fail to produce an angle due to missing characteristics. If that would happened the MBR algorithm then will take over to find a skew angle.

The MBR is not relying on line characteristics like the HLT do: instead it works by bounding all the content available in the image and finding the smallest rectangle possible by systematically rotating in every angle. Once the smallest rectangle have been found it calculates the angle of the rectangle. A visual example can be found at Figure 5.2.

### 5.1.3 Theme Determination

The process of theme-determination consist of converting the image into a binary image, counting all white and black pixels and comparing the ratio. If the white-to-black ratio is above 0.8 it can easily be assumed the current theme of the interface is white. If this is the case the image is marked to be inverted, see Figure 5.3. The determination of the theme is essential for the upcoming image processing. where every image is assumed to consist of black background and white content.

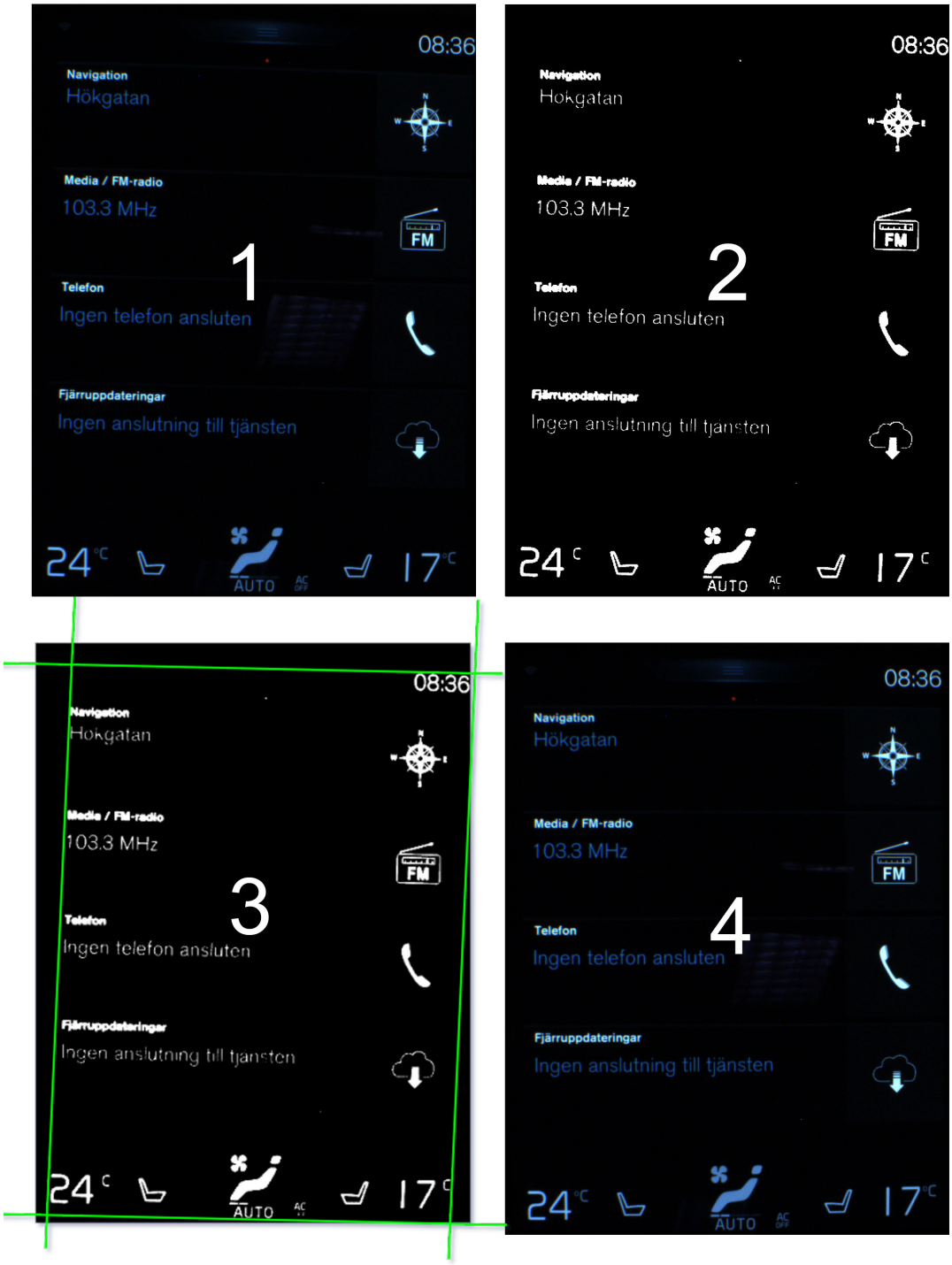



Figure 5.2: *An example of utilize the MBR algorithm for finding a skew angle. The original image (1) is converted into a black and white(binary) image(2). Which is encapsulated by the MBR algorithm for finding the skew angle(3). The image is rotated by the angle calculated from the MBR algorithm (4)*

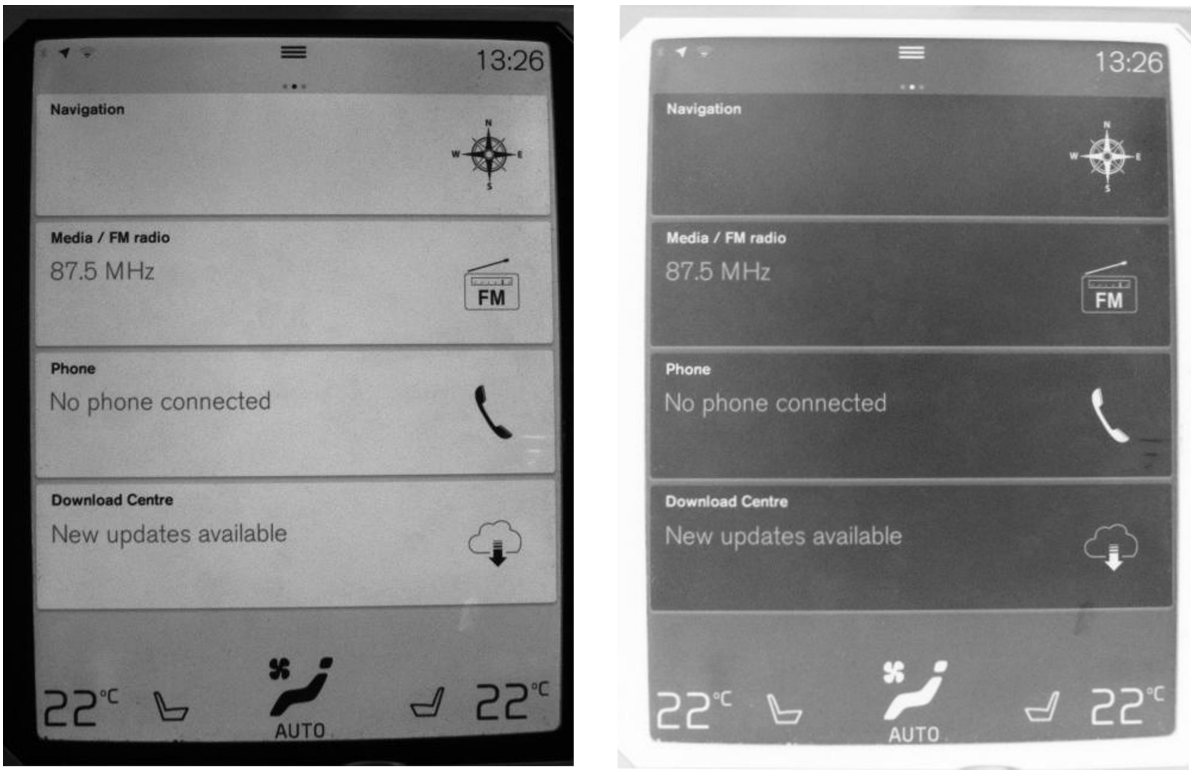



Figure 5.3: *Example of an white theme inverted to black to fit the image processing parameters*

## 5.2 Function Calls

The function calls are important for Volvo's system testing but are also very helpful for experimenting with symbol recognition. `doOCR`,`findWord` and `findText` are three functions which shares the same logic for finding the result but in the last step of each function different results is returned depending on the function calls objective. `initialize` is the first function to be used to start and initializing the system. `changeLanguage` can be used to change recognition language during run time. But it is a slow process and requires Tesseract to restart.

## 5.3 Image Processing

Processing an image is essential for extracting segments of words and symbols. The process starts by rotating an image but only if a skew angle were detected during the initialization. It is rotated by the *Affine Transformation*(AT) algorithm. An image can be represented by a matrix, to rotate it the $AT$ algorithm transform the current matrix based on the given skew angle into a new perpendicular image.

The rotated image is then inverted if the invert parameter flag is enabled from the initialization. The image is then converted into gray and morphological *opening* and *closing* is applied. This is to remove any static noise which can be found in bad quality images. The next step of the image processing is to convert it into binary, but only if the image is considered noisy, Otherwise it is put through the CED Algorithm. It is extremely helpful to enhance the characters and symbols and at the same time remove disjoint pixels. The image is then *closed* again, this is to connect letters and the small symbols into a blobs forming words or a symbols. See Figure 5.4 in the end of the section, showcasing the extraction of the blobs on a noisy image.

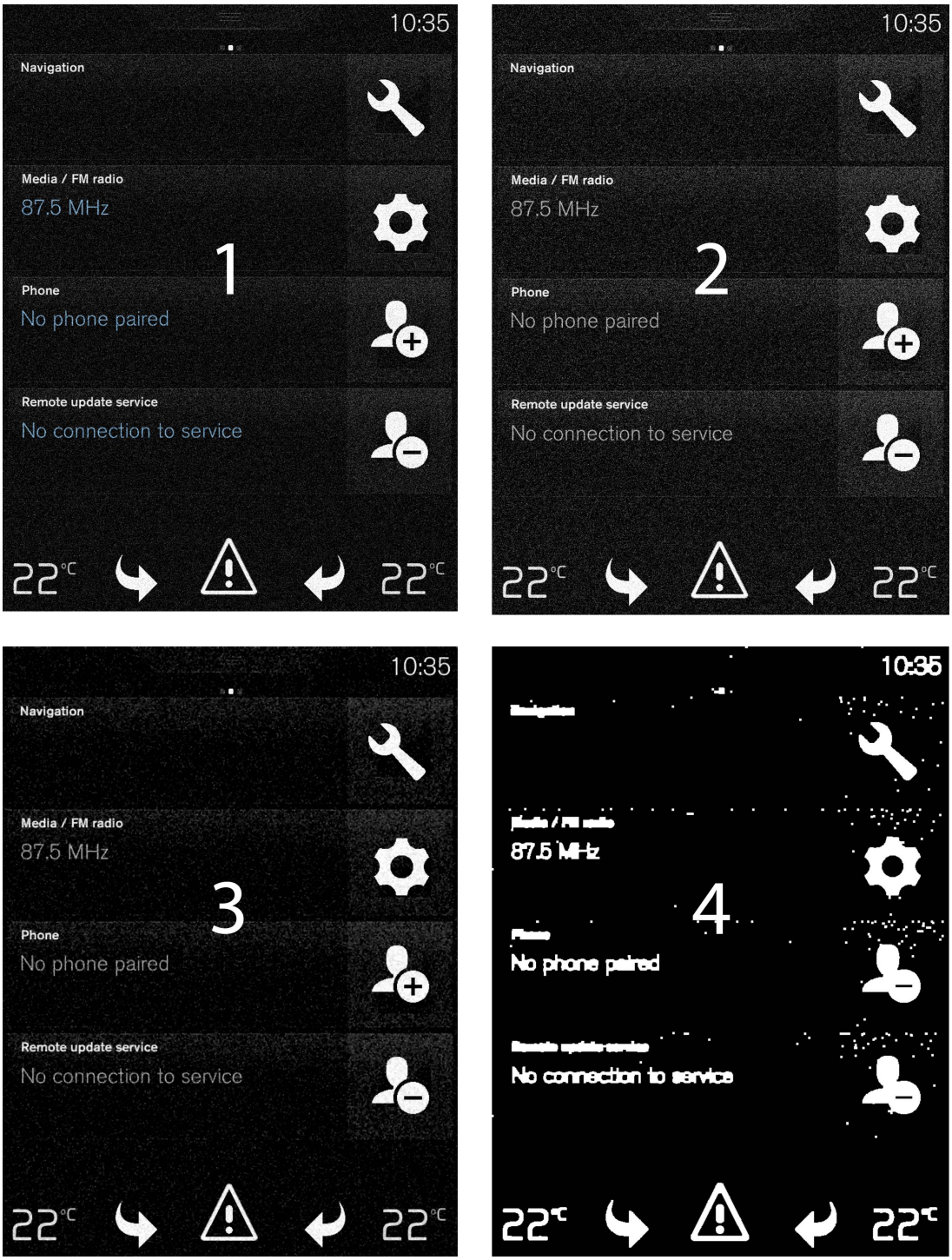



Figure 5.4: *Example of enhancing the blobs in a noisy image. The original image (1) converted to gray (2). Closing and Opening is applied with a small SE to remove any noise(3). The image converted to binary and then closed is applied again to connect characters and symbols into blobs (4)*

### Segment Identification

After the primary image processing is completed the blobs consisting of words and symbols are enhanced. Next step is to localize the blobs and extract their positions to later extract the segments. It is done by applying a contour algorithm [36] on a closed binary image which can be show showcased from Figure 5.4 below. The algorithm finds all the blobs and encloses them by contours. OpenCV's *Bounding Rectangle* algorithm is then used to bound each contour into the smallest possible rectangles. All rectangles which are extremely large or extremely small are considered to be insignificant, and will be removed for efficiency reasons. The remaining rectangles' positions are inserted into a *result list* (5.4) and represent the positions of each segment which later is recognized by Tesseract. See Figure 5.5 where the segment positions is extracted from a image without any noise.

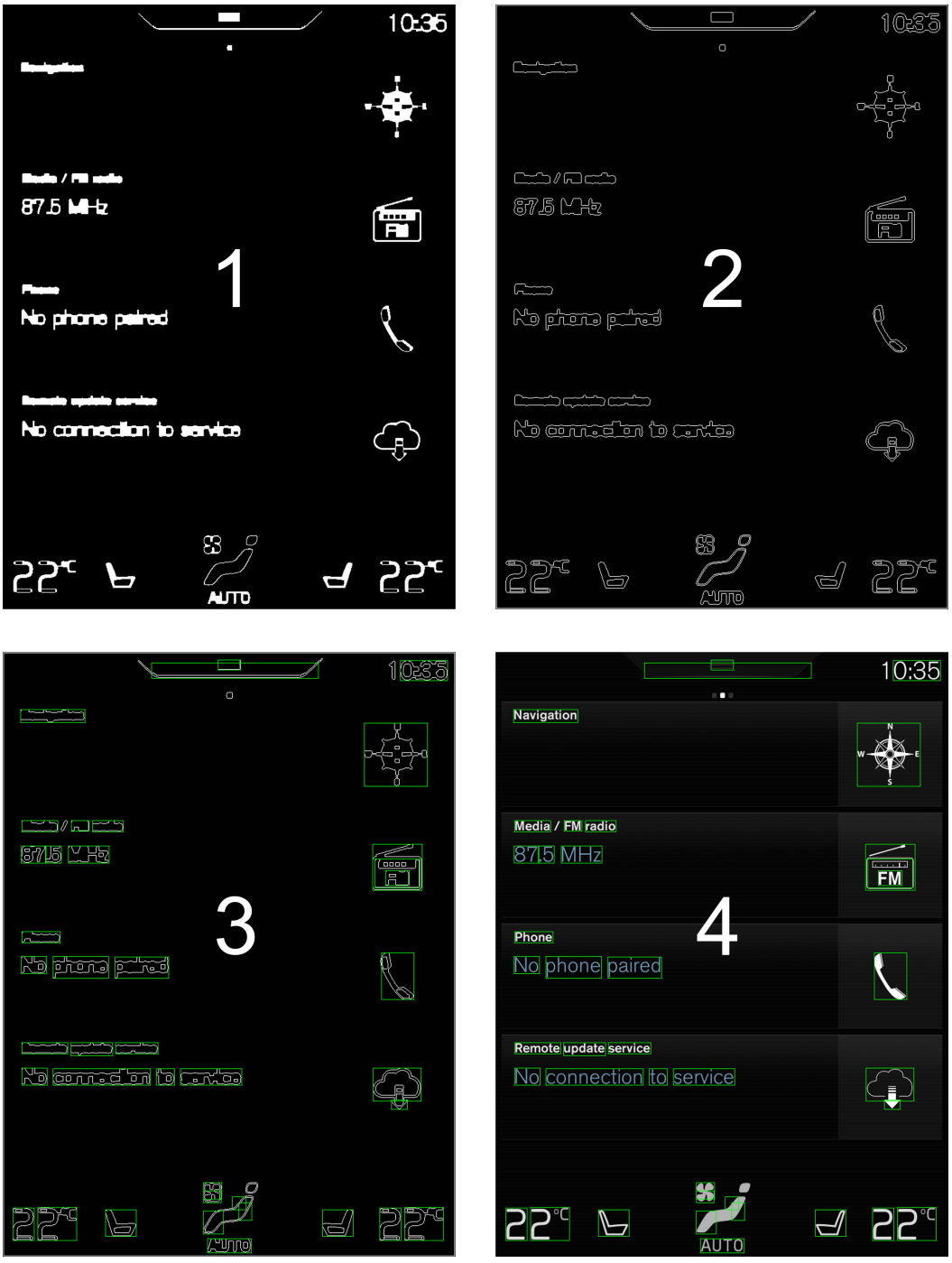



Figure 5.5: *Example of identifying segment positions (displayed as green rectangles). First the closed binary image (1) is used to extract the contours (2). The contours are then bounded by rectangles (3), the rectangles are positioned onto the original image to showcase the positions of each segment(4).*

## 5.4 Recognition

By letting Tesseract recognize the whole image at once recognition results will be poor. This is because of the different text and symbol sizes causing problems for Tesseract with structuring the information. To address this problem isolation of each respective segment is essential and will provide a more accurate recognition. This is also practical because it can pinpoint the exact location of the segment which is required for localization during system testing.

Each and every segment is extracted from the original image into small images containing only the information of the segment, see Figure 5.6. The segments are then fed into the Tesseract Engine one by one. Tesseract receives them and starts to recognize them, and as a result Tesseract will return an answer of the recognition along with a confidence of the analysis. This will be added to the already existing *result list* where segment positions were inserted.

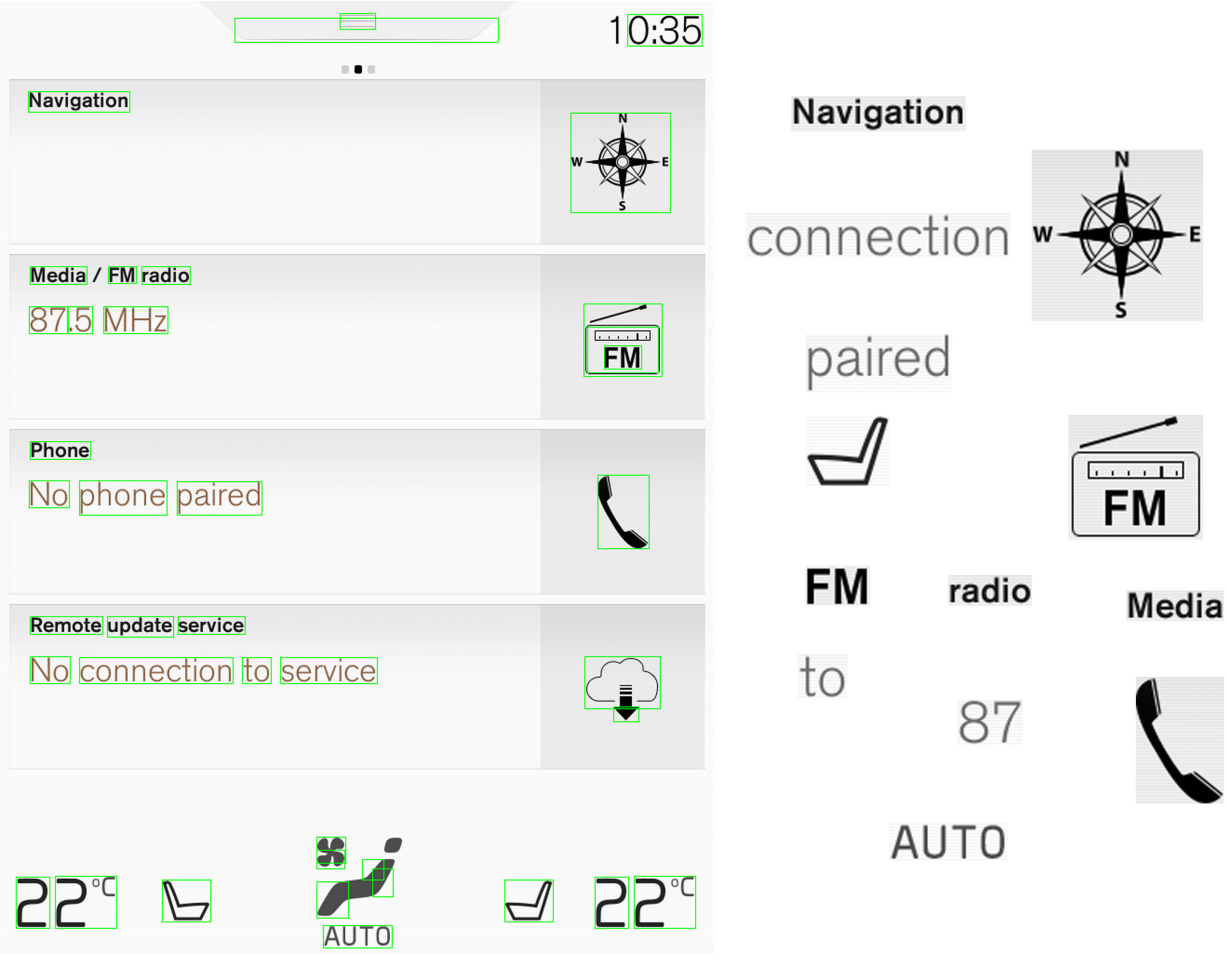



Figure 5.6: *Example of some segments extracted from the original image, later to be sent into Tesseract for separate recognitions.*

### The Result class

The result class contains all necessary information of a segment:three types of positions(top left, bottom right and center), the recognized word or symbol along with the confidence and finally an image displaying where the blob is located.

The top left corner and bottom right corner positions are used by CameraVision to project the location of the recognized segments. The center position is used for testing by a touch panel testing robot. The recognized word/symbol is used for verification and the confidence can be used to set thresholds on the reliability of the recognition. The image with a highlight over the current blob is presented as visual feedback for the tester during troubleshooting.

The result class is presented as a *result list*, where each and every result in the list represent the information of specific segment. The list is the final building block to be analyzed and return an answer to respectively function call.

# CHAPTER 6

# Experimental Procedure

This chapter covers the experimental procedure explaining how the implementation was tested.

The original intention was to use Volvo Cars own interface symbols. However, they were not available in vectorized format. But the study is not focusing on Volvo's specific symbols and any types of symbols works. Figure 6.1 displays the symbols that were used for experimentation. There are both unique symbols and a mixture of symbols sharing the same outlines. The various symbols should cover the general symbol aspect; where some are fully connected like a typical character, some are disjointed into smaller symbols, others are hollow and some are full.

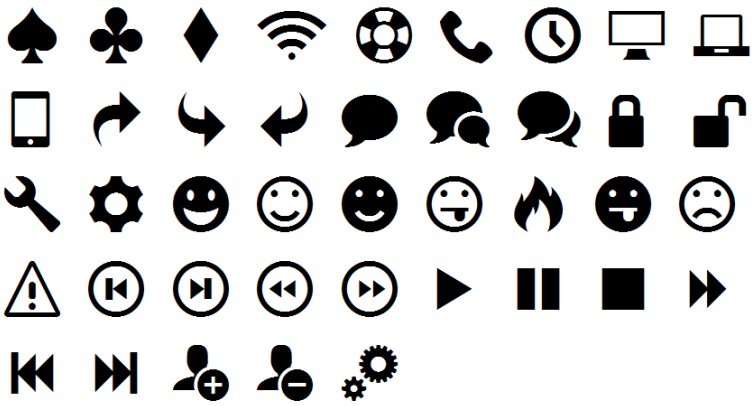


Figure 6.1: *Symbols trained to be recognized by Tesseract*

## 6.1 Symbols in perfect conditions

The first experiment was done in perfect conditions, meaning that the recognized image is the same as the one used for training Tesseract. It is essential to be able to draw any conclusions of the engine's capabilities of handling different and similar symbols. This is to ensure the engines capability, and it will be shown what types of symbols Tesseract have problems with recognizing. The result here helps to understand the other result in other conditions.

Table 6.1 presents the result from some of the symbols recognized in perfect conditions. To see the complete table see Appendix A.2. The table is sorted by recognition confidence where the best confidence comes first, and the symbol with lowest confidence comes last. The table consist of the recognized **symbol** followed by the symbol **name**, the **recognition** is the returned answer from the recognition and the **confidence** is Tesseracts expression for the grade of the recognition. The first 7 symbols in the table represent the symbols which will be used for all the upcoming tests.

| **Symbol** | **Name** | **Recognition** | **Correct** | **Confidence** % |
|---|---|---|---|---|
| | WRENCH | WRENCH | YES | 86 |
| | COG | COG | YES | 91 |
| | USERPLUS | USERPLUS | YES | 85 |
| | USERMINUS | USERMINUS | YES | 84 |
| | WARNING | WARNING | YES | 96 |
| | REPLY | REPLY | YES | 78 |
| | FORWARD | PHONE | **NO** | 46 |
| | PHONE | PHONE | YES | 95 |
| | DISPLAY | DISPLAY | YES | 99 |
| | MOBILE | MOBILE | YES | 98 |
| | CLUBS | CLUBS | YES | 95 |
| | CLOCK | CLOCK | YES | 91 |
| | NEXT | FORWARD | **NO** | 87 |
| | PAUSE | MOBILE | **NO** | 73 |

| Symbol | Name | Recognition | Correct | Confidence % |
|---|---|---|---|---|
|  | TOUNGE2 | TONGUE | **NO** | 73 |
|  | LOCK | MOBILE | **NO** | 71 |
|  | HAPPY | TOUNGE | **NO** | 71 |
|  | WIFI | FIRE | **NO** | 49 |
|  | COGWHEELS | SMILE2 | **NO** | 48 |

Table 6.1: *Recognition samples of symbol recognition in perfect conditions, see Appendix A.2 for full table*

The clear trend in the recognition Table A.1 is that the recognitions below 73% confidence are not able to properly recognize correct symbols. Some of them however, were quite close though to matched a similar symbol. An important observation from the table shows that symbols with high recognition confidence can be wrong where the NEXT is recognized wrongly with an 87% confidence. The table also shows that the OCR engine is capable of recognizing different symbols sharing the same outlines, e.g the clock, smiles and the play buttons. Disjoint symbols like the pause or the wifi symbols seems to be very hard to interpret by the engine. Normally characters are connected and this might be a reason why the engine can not handle disjoint symbols.

The most surprising result presented in Table 6.1 above is the forward symbol, it is very similar to the reply symbol but is incorrectly classified into a phone. One would think that it is trivial if reply can be correctly classified then forward also would be correctly classified as well. Abstracting the outlines of the two symbols there are some similarities between the two, both sharing the same curve. However, the outlines are completely different which makes the result questionable. It is safe to conclude that OCR works to recognize symbols to some extent in perfect conditions, but the conditions needs to be bended into proper real use case scenarios to for further evaluation for real scenarios.

## 6.2 Varying conditions

The next upcoming experiments utilized Volvo's interface image but instead of its original symbols in the image, they were replaced by some of the trained symbols from Figure 6.1. See Figure 6.2 for the new *experimental image*.

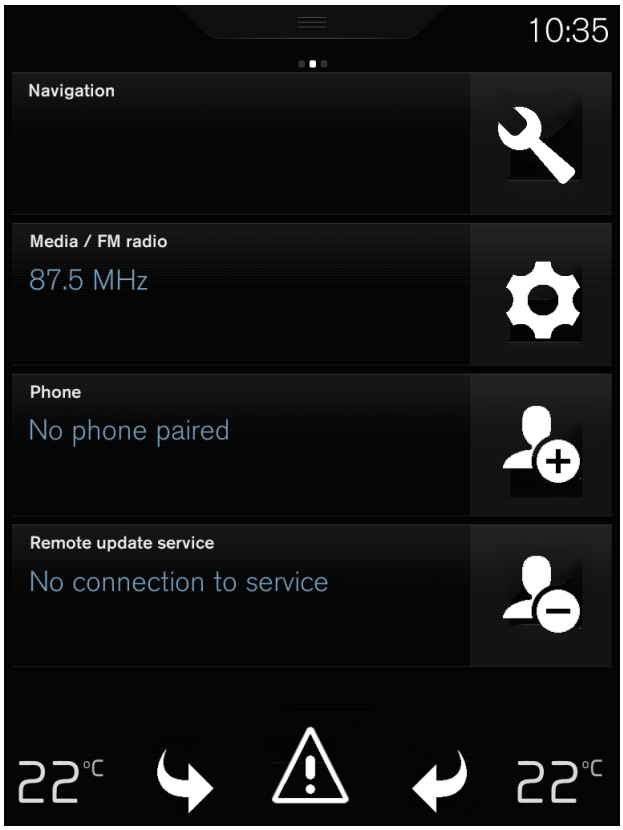



Figure 6.2: *Volvo's interface with replaced symbols*

### 6.2.1 Different Filters

The first experiment were testing the robustness of the recognition with different filters applied on the *experimental image* (Figure 6.2). The filters enabled the possibility to see what happened when similar symbols were recognized instead of the exact match (as in the perfect condition experiment). The different filters applied were: a pixel filter, a more intense pixel filter, glass filter, cutout filter and spray filter. A human can clearly see the symbols correlate to one and another. The table will showcase the OCR recognition performance of how the engine can regonize symbols similar to the trained, but not quite the same. If a symbol is incorrectly classified it is marked with "ERR", this does not mean the confidence is 0. As displayed in previous table (Table 6.1) the engine can incorrectly recognize a word and still have a high confidence.

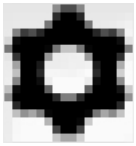
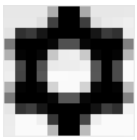
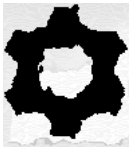
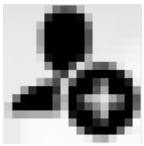
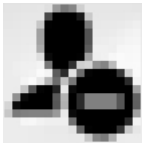
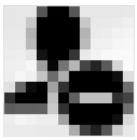
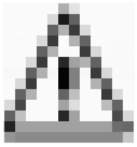
| **Name** | **Original** | **Pixelated 1** | **Pixelated 2** | **Glass** | **Cutout** | **Sprayed** |
|---|---|---|---|---|---|---|
| Wrench |  |  |  |  |  |  |
| Confidence % | 83 | 74 | 54 | 81 | 83 | 83 |
| Cog |  |  |  |  |  |  |
| Confidence % | 92 | 73 | 75 | 89 | ERR | 74 |
| User-plus |  |  |  |  |  |  |
| Confidence % | 84 | 78 | 62 | 79 | 86 | 62 |
| User-minus |  |  |  |  |  |  |
| Confidence % | 84 | 70 | ERR | 81 | 84 | 59 |
| Warning |  |  |  |  |  |  |
| Confidence % | 94 | 60 | 55 | 87 | 94 | 81 |
| Reply |  |  |  |  |  |  |
| Confidence % | 80 | ERR | ERR | 87 | 77 | 65 |
| Forward |  |  |  |  |  |  |
| Confidence % | ERR | ERR | ERR | 43 | ERR | 52 |

Table 6.2: *interface symbols with different filters*

The table is not obvious to correlate the results, to justify and put them in another perspective they are arranged in Figure 6.3 below. The dots with black background indicate incorrect classification.

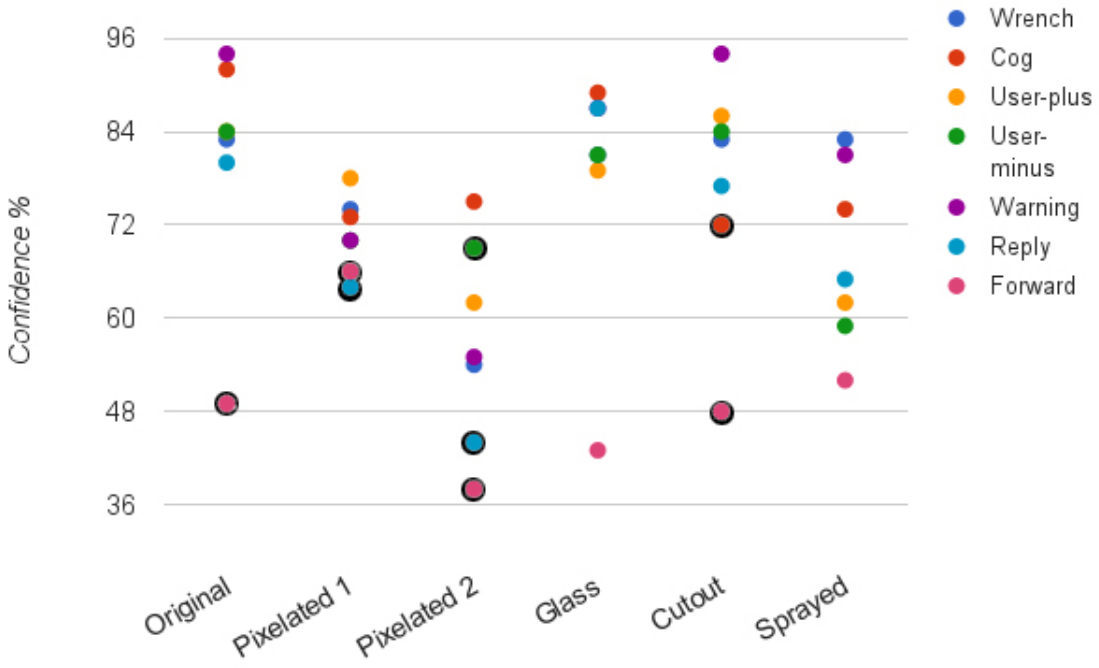



Figure 6.3: *Table 6.2 compiled into a diagram, dots with black background indicate incorrect recognition*

The result from Table A.2 and Figure 6.3 shows promising results, meaning that the engine manages to recognize symbols similar to the trained reference even though they are put through heavy filtering. The polynomial approximation really shine in this type of experiment, even though the symbols are very different from a pixel-to-pixel comparison they still share the same outlines. This could be an advantage over image registration functions built on pixel-to-pixel verification.

The most surprising fact is the the forward symbol, it has been shown in previous perfect conditions test that it was not recognizable. However, it is able to match the correct symbol with different filters applied. This indicate that the training lack credibility, or simply more training data is needed since the forward symbol cannot be recognized in perfect conditions, but filtered. The correlation between confidence and wrongfully recognized symbols shows to be very spread, from the results conclusions can be drawn that it is capable of correctly classify symbols above 73% confidence.

### 6.2.2 Decreasing Resolution

Naturally one can assume that a decreasing resolution resolves into inferior result, but had to be verified with an experiment. The experiment showed the recognition confidence in correlation to the decreasing image resolution.

The experiment started by taking the *experimental image* (Figure 6.2) which was 92 Pixel Per Inch (PPI), equals to the resolution of 768x1024 pixels. The resolution and PPI share the same ratio, meaning that dividing the PPI by half means to divide the resolution by half as well.

#### Symbols

The diagram presented in figure 6.4 displays the correlation between symbol recognition and decreasing image resolution on the *CSD FRAME* (Figure 6.2) interface. The dots with black background indicates incorrectly classification.

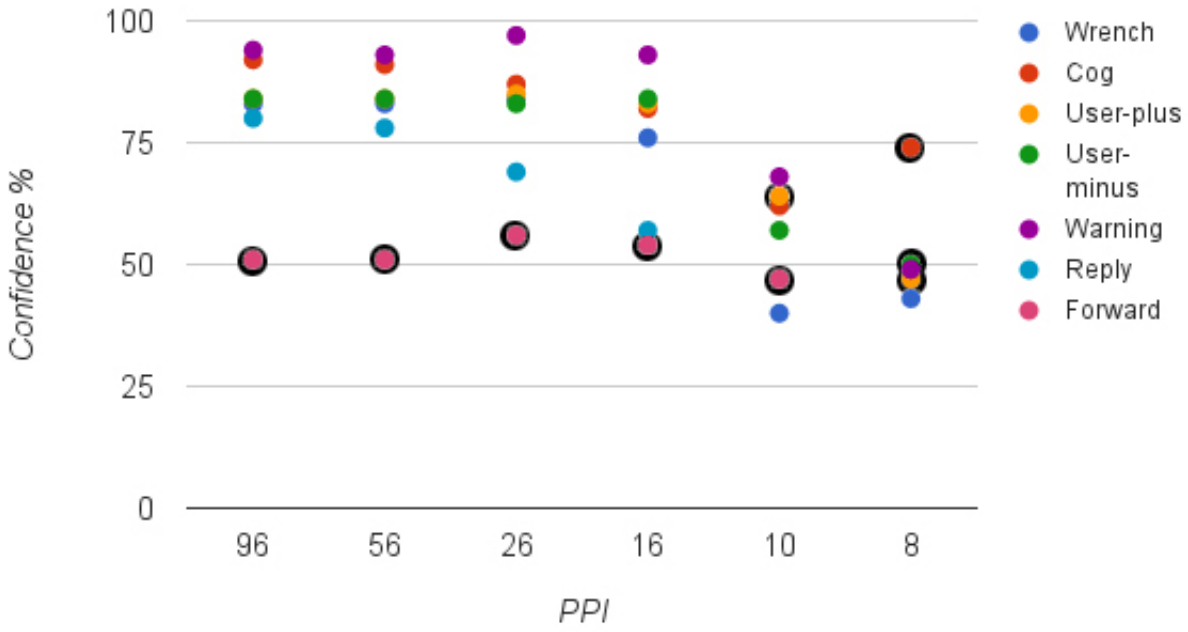



Figure 6.4: *Symbol Recognition vs Image Resolution*

The result shows that symbol recognition is not affected to much by the resolution, eventually the result will be affected once the PPI reaches somewhere between 16-10 PPI. The Forward symbol were not recognizable at any point which were expected, as seen from the previous experiment in perfect conditions it was not reconizeable either.

### Words

To be able to evaluate and compare the results from decreasing resolutions and symbol recognition, the same experiment was conducted with word recognition. The diagram presented in Figure 6.5 displays the same as previously presented in Figure 6.4, but in this case it is the correlation between image resolution and word recognition. This could give an indication of how well symbol recognition is compared to word recognition in decreasing resolutions.

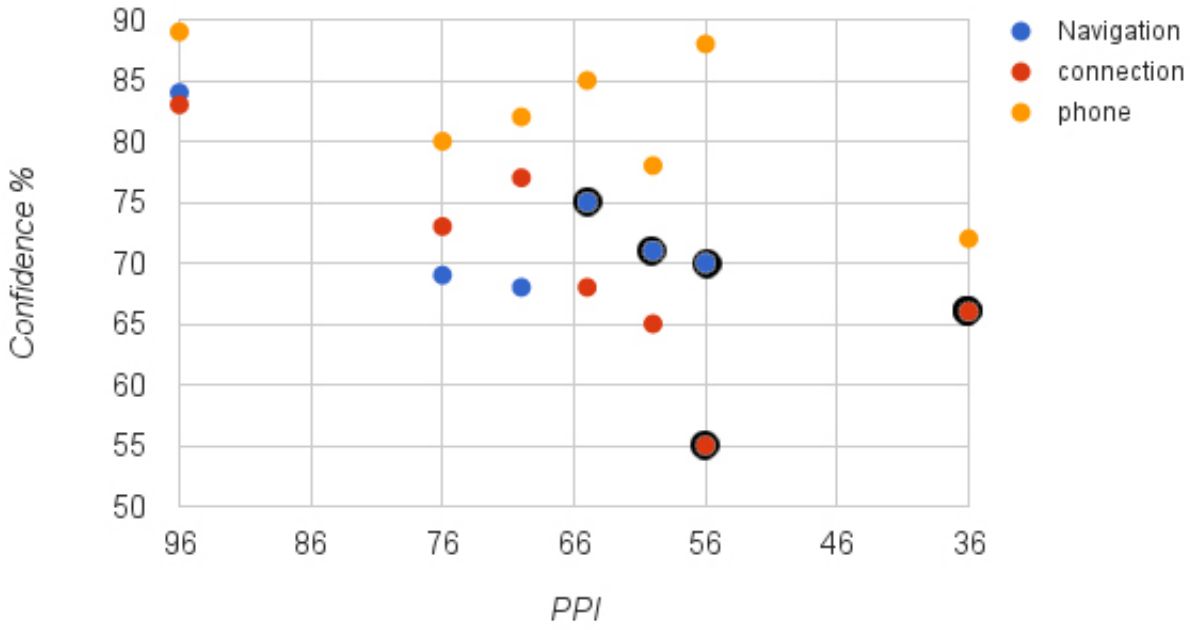



Figure 6.5: *Word Recognition of "Navigation", "connection" and phone vs Image Resolution*

The results show that words are very dependent on the resolution and will not be recognized at all after 36 PPI which coan be seen in Figure 6.5. Compared to the symbols recognition which withstood a decrease of 8 PPI (Figure 6.4).

## 6.2.3 Increasing Noise

The next experiment was similar to the previous decreasing resolution experiment. Instead of testing the resolution the experiment tested an increasing noise level. Image noise is a very common problem in computer vision, testing this issue is very important to show the recognition capabilities with an increasing noise level. The noise in the image was artificially produced with the help of a Gaussian noise filter.

The diagram presented in figure 6.7 displays the correlation between the recognition and increasing Gaussian noise levels. The noise level starts from 0% and increases up to 40%, where 40% noise level corresponds to a standard deviation of 43.36. A visual example can bee seen in Figure 6.6. Above 40% Gaussian noise level, the recognition were not able to recognize any symbol.

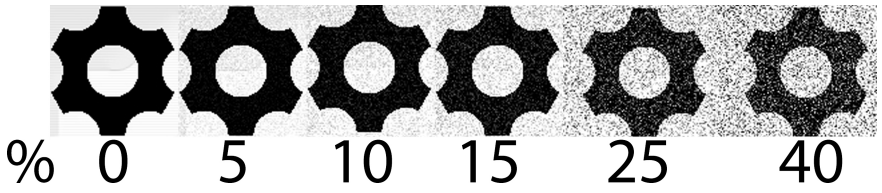



Figure 6.6: *Example of the COG symbol with increasing Gaussian filtering*

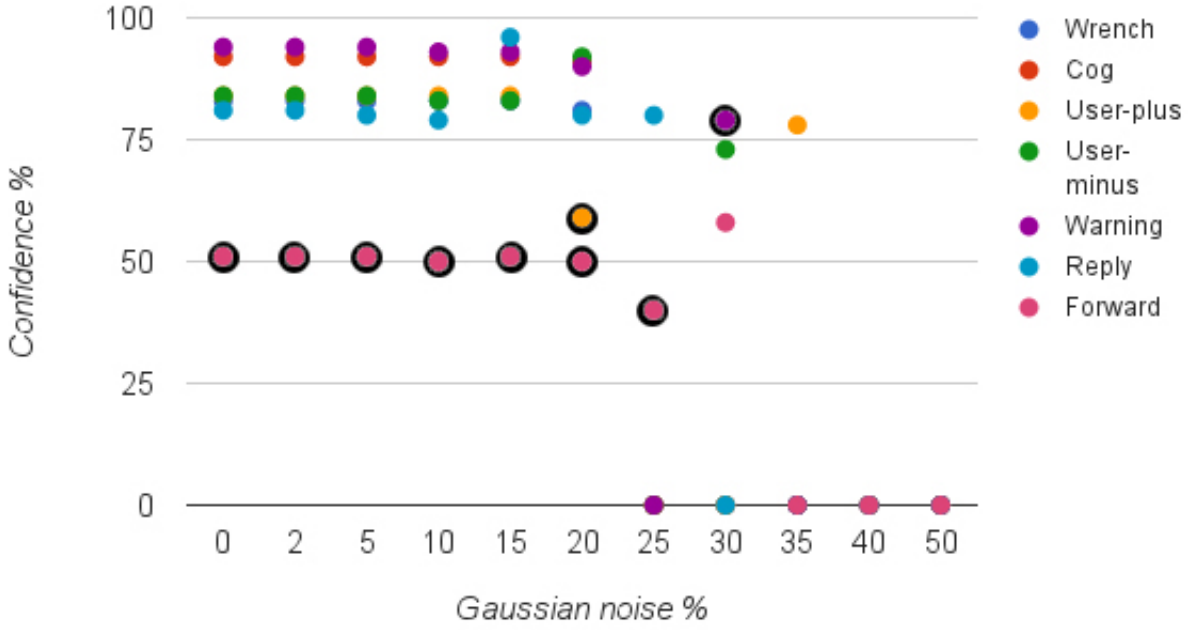



Figure 6.7: *Symbol Recognition vs Gaussian Noise*

It clearly shows that the recognition is affected by the noise levels after 15%. But still it provides information that the recognition is rather robust and can perform well on noisy images.

## 6.3 Real Conditions

The next experiment was conducted on the Driver Information Module (DIM) (Figure 6.8), another type of interface used in testing at Volvo. This image is featured in testing today where the symbol in the center top position is located using *image registration*. However, the words below in the same image cannot be recognized and verified. In the experiment the implementation is tested to do a real test scenario, where it can locate and verify both words and symbols at the same time. A proof of concept test showcasing symbol recognition along with word recognition for verification purposes.

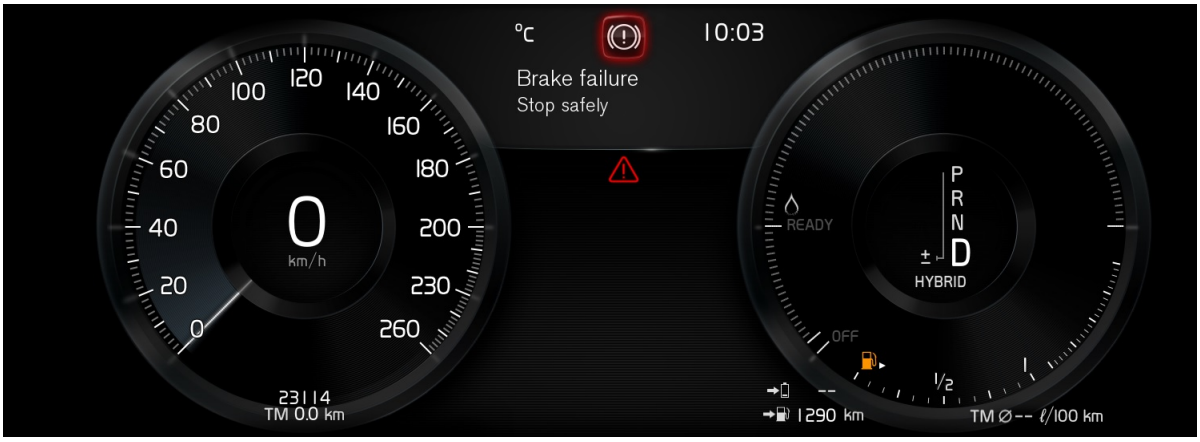



Figure 6.8: *DIM interface, used today to verify the symbol in the top center position of the image by image registration.*

The table 6.3 presents the result from the DIM interface. It can be considered to be a proof of concept of the potential and ability to use OCR for symbol verification.

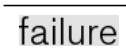
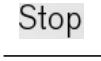
| Segment | Name | Recognition | Correct | Confidence % |
|---|---|---|---|---|
|  | Brake | Brake | YES | 78 |
|  | failure | failure | YES | 84 |
|  | Stop | Stop | YES | 77 |
|  | safely | safely | YES | 81 |
|  | WARNING | WARNING | YES | 90 |

Table 6.3: *Recognition of words and a symbol*

The table clearly shows that all the words and the symbol searched for were found with a satisfying result. A satisfied result in this case is because of the real conditions of the test case, and during this conditions the engine is able to come out on top.

## 6.4 Efficiency

To test the efficiency of the different processes a timer was started before a process began and stopped after it was completed. To validate the efficiency every experiment was run 15 times and the result presented is the average of these 15 results.

### Initialization process

The initialization process was tested with and without skew calculations on the white themed image from the previous Chapter (Figure 5.3). This types of images is most likely to be tested from Volvo and can provide a realistic result. The diagram presented in figure 6.9 displays the efficiency of the *Initialization process*.

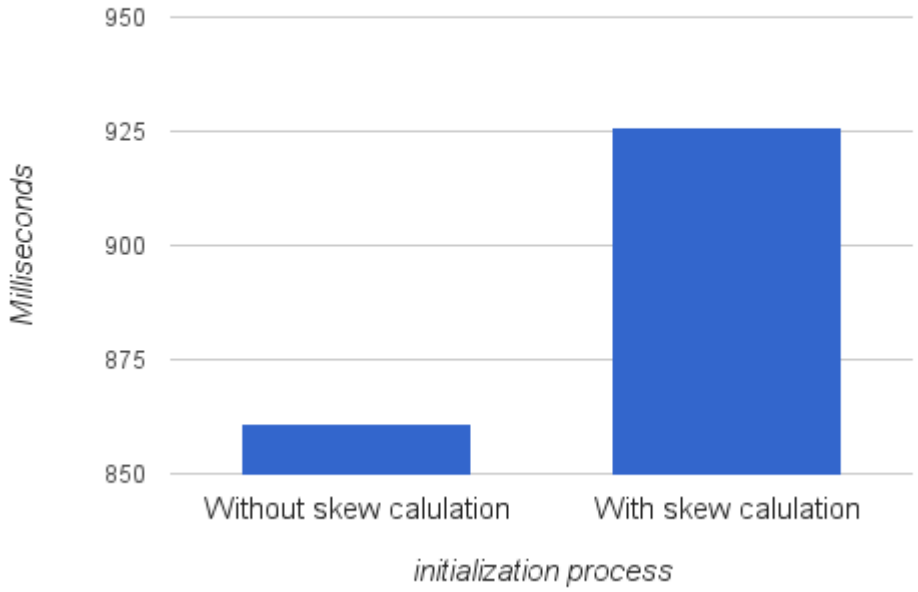



Figure 6.9: *Efficiency experiment of the initialization process, with and without skew calulation.*

The diagram clearly shows that there is a difference between the two, but it also show that the Hough Line transform algorithm is effective, considering the heavy computing for calculating the skew angle.

### Different interfaces

This experiment shows the efficiency of different types of interface images which can occur at Volvo's interface testing. Some of them have low resolution, some have high. Other have a complex interface while others are quite unadorned. The efficiency was tested on four different image:

- CSD Frame (Figure 6.2)
- CSD Camera (Figure 5.3) from Chapter 5 (Implementation)
- DIM Frame (Figure 6.8)
- DIM, A Region Of Interest (ROI) (Figure 6.10)

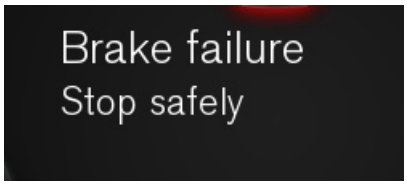



Figure 6.10: *Region of interest, a region of the DIM interface (6.8)*

The diagram presented in figure 6.11 display the efficiency on different types of interfaces. The diagram shows how the efficiency differentiate depending on the different interfaces. The *DIM FRAME* interface have the largest resolution and contain most information to processes and can be assumed to be the slowest, and it is verified by the diagram. The *DIM ROI FRAME* is an extracted region from the normal *DIM FRAME*, very small with limited content making it very efficient to analyze. The most interesting result is the CSD frame and CSD Camera; they have identical interface and share equal amount of content to be analyzed. However, *CSD FRAME* is taken from a frame grabber with a high resolution and the *CSD CAMERA* is taken by a camera with lower resolution.

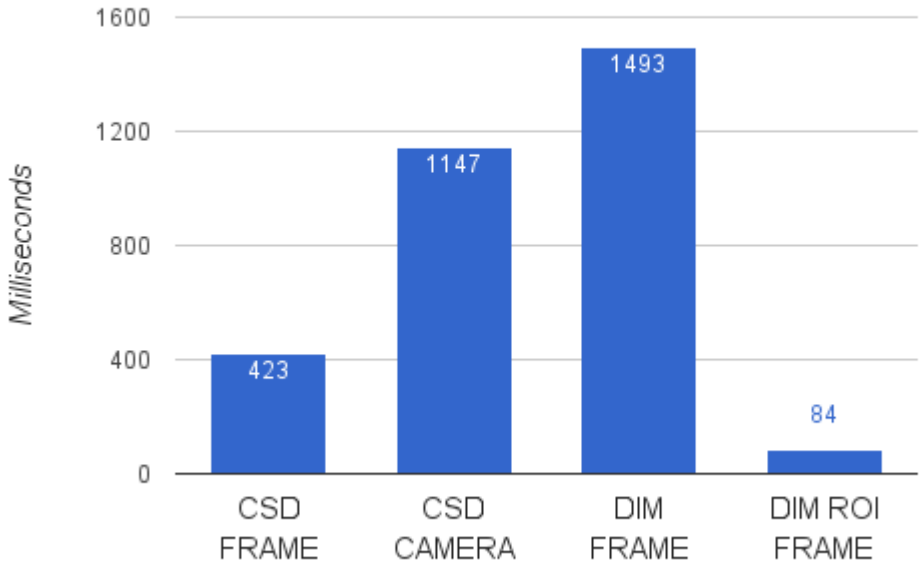



Figure 6.11: *Efficiency test with different types of interfaces*

The diagram shows that there are different processing times depending on what image is tested. The *DIM FRAME* is slowest of them all, but it contain most content and have the biggest resolution so the result is expected. The *CSD FRAME* have larger resolution

then *CSD CAMERA* but is much faster. Booth CSD interfaces contains the same amount of information, only the resolution and quality varies. The result shows clearly that lower quality equals higher processing time. The *DIM ROI FRAME* is a small piece of the *DIM FRAME*.It shows the fastest efficiency which is expected because it contains the least amount of information and the lowest resolution.

### Different Noise levels

In the last experiment the efficiency of the image processing is compared to the amount of noise in the image. This experiment will show in Figure 6.12 how the noise affect the image processing. The experiment was utilizing the same *experimental interface* image to see how symbols are recognized with the increasing noise.

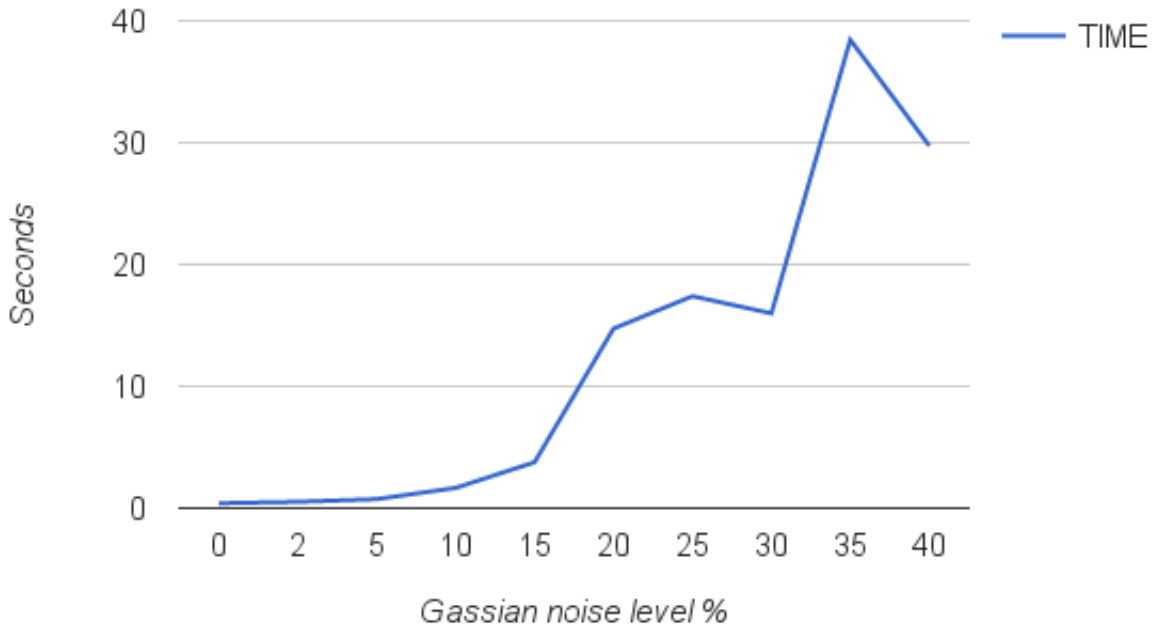



Figure 6.12: *Processing efficiency vs increasing Gaussian noise*

The diagram clearly shows that the processing time increases when the noise levels increases. However, in the beginning between 0-5% noise level increment the processing time is not changed dramatically. This shows that noise below 5% does not affect the efficiency remarkably.

# CHAPTER 7

# Evaluation

## 7.1 Analysis of the result

The results from recognizing symbols with Optical Character Recognition point out that it really is possible. Even though Tesseract engine were put through a vast majority of different images with different filters, sizes or noises it still proved to be able to recognize most content. This proves that at least some symbols can be identified and recognized with great accuracy. But the problematic side of the results shows that symbols which can be assumed to have been easily be recognized were not. Even though the conditions for recognition were perfect, some symbols were not recognizable. With this information it can surely be stated that something is faulty, either the training data or the training process. Retraining the engine with the same data did not make any difference in the upcoming recognitions. Both the training and recognizing seems rather static where every result came out the same no matter how many times they were tested.

Even though the engine received the minimum amount of training data, the experiment have proved that OCR can be used for verification and testing, but should be bounded to verify simple symbols without too much complexity inside. It is difficult to recommend this for verification testing because of the randomness of which symbols were recognized properly. Two similar symbols with clear outlines features can be trained but only one may be recognized later. E.g two arrow symbols: The arrow pointing in the left direction is easily recognized but the other arrow pointing right is not recognized at all. The symbols clearly share the same outlines and with a 180 rotation of one of the symbols they would be assumed equal. It is difficult to see the reason for this and points to something is still not working properly inside the training process. In order to use OCR for symbol verification,

each and every symbol needs to be tested individually to be verified if it is recognizable in perfect conditions. If it is not it will most probably never be recognized later during testing either.

OCR is built to examine the outlines of each object in the image, whether it is a symbol, a character or a word etc. If the outlines can be identified it is considered to be a match. Image registration on the other hand analyze the whole image as one, it does not matter if the image is binary, colored or vectorized like OCR does. It locates feature points and translate them into a map of feature coordinates, if the map can be aligned to a reference-map a match is considered. This means that recognizing smaller symbols inside a bigger image could be very difficult using image registration, possible the smaller symbols could consist of only one feature point each to map the image, making it impossible to find a symbol match. OCR on the other hand work very well with smaller symbols in an image, it does not consider the full image as an object. Instead it locate the blobs (objects) and analyse them separately to find a match, it does not matter what sizes they are, as long they are disjointed from each other. Considered this scenario, symbol recognition using OCR have a slight edge compared to image registration but the technology have its own flaws regarding sustainable and correct recognitions. This currently makes image registration the best contender of the two counterparts.

## 7.2 Discussion

OCR has a great potential for recognizing symbols in the future, the reason for this is mostly because of the *Neural Network Classifier*(NN). Hopefully it will be the missing piece to be able to distinguish and recognize similar symbols with the same outlines but different insides. Developers are working frequently on keeping the engine up to date for more accurate character recognition and adding support for new languages. But the real struggle here from a symbol recognition perspective is all the tools surrounding the engine:

• The training process: it is undeniable that this is an extremely important feature for the future of symbol recognition. But it lacks proper documentation and tools today to satisfy an outside developer. The process needs to be more usable and give more feedback to the developer about the training results, for instance regarding which symbols were successfully recognized and which were not. As of now, there is no such information and you can only hope for the best during training. It is up to the developer to later distinguish whether or not the recognition of the specific symbol worked as intended.

• Fonts: unfortunately since OCR is for characters, it assumes characters will be recognized and therefore it requires a Truetype font to be used for training. This is very problematic if you have symbols, naturally symbols are not found inside fonts but stand alone images in a library. To solve this issue all the separate symbols have to be converted into a vectorized format, once that is complete it can be used to build a new type of font with symbols. This is extremely time consuming and is not correlated any how with any of the available tools for Tesseract. Thus, to solve converting symbols into a font you have to rely on other programs which is time consuming.

• The box file: To train Tesseract it requires a high definition image with the symbols along with a box file. Inside the box file you have to manually set the parameters of each symbol name and the location on the image. The training process does not tell you if something is corrupt, missing or does not work.

The process for actually inserting the new symbols into the training data is very complicated, this is not a process any developer want to go through in order to use the symbol recognition functionality. Even though developers proceed and complete all the required steps to train OCR engines with symbols. Each and every symbol need to be properly evaluated separately to see if it can be recognized later. The Tesseract training tool is built to be used for character recognition. Even the process to train a normal character font can be challenging for a less experienced developer. This is why the process of symbol recognition through OCR is not mature enough to be used for this purpose. The results have displayed the robustness and the actual recognition is rather accurate once you have verified that the engine can recognize the symbols. But the process to get there today is way too time consuming.

## 7.3 Conclusion

Optical Character Recognition *can* be used for symbol recognition and it proves to have a robust and accurate recognition under various circumstances with different resolutions, noise-levels and filters. The technique is special in a way, because it can handle recognition of characters and symbols at the same time. But the processes to be able to do so is not satisfying. Due to all the different amount of steps that is required to be able to train the engine to recognize symbols. Also the training appears to be too random to understand why the same symbols with different angles have a different outcome from the recognition, where one can be recognized and the other not even though they share the same outlines. Tesseract OCR classifies the symbols depending on outlines, naturally characters have different outlines and makes this a natural classification technique. However, the symbol recognition is suffering largely from this where many symbols share the same outlines but have different content inside.

A plausible solution for OCR and symbol recognition is under development and might be the lurking factor which is missing today: The Neural Network(NN) classifier. Utilizing the old static and adaptive classifier together with the NN classifier can be of great help in order to determine the content inside each symbol.

Symbol recognition with Optical Character Recognition is very possible, the continuous development of the classifiers accuracy will be benefited for symbol recognition as well as characters. The symbols which were recognized accurately in the perfect conditions also stood out and were recognized properly in the other experiments with various circumstances. Until the day when the NN classifier is implemented and a properly setup of help-tools for training Tesseract, Image registration is the technique to use for recognizing and verifying symbols.

## 7.4 Future work

The dynamic morphological parameters calculations require more test images to be able to produce a more stable formula to be used in the long term. Even though it provide an overall good result, it should be inserted as an optional input for the testers since this is the most essential piece of finding the blobs which the information is segmented into.

A fundamental piece that would make a significant difference is to implement some sort of word translation library. This would provide the possibility to allow one tester to run tests in several languages. Today Tesseract can accurately recognize and interpret many different languages, but as a result it will return that specific language as well. If the tester does not understand the specific language it is very difficult validate or to build tests. However, if a translation package were available the tester could build and validate tests in their preferred language, translated from the foreign language.

To improve the robustness and force the recognition error rate to be even lower the segmentation of words should be broken down and segment the characters as well. It is shown in the previous theory chapter that character recognition error rate is lower than word recognition. To support this implementation, if every word can be broken down into characters a language library is required. This would be used to verify that the segmented characters actually create a word. If not, Tesseract word recognition could be used as a second alternative.

# Bibliography


[1] Matthew J Pitts, Gary Burnett, Lee Skrypchuk, Tom Wellings, Alex Attridge, and Mark A Williams. Visual–haptic feedback interaction in automotive touchscreens. *Displays*, 33(1):7–16, 2012.

[2] Number of touch screens installed in cars worldwide in 2011 and 2019. `http://www.statista.com/statistics/261697/availability-of-touch-screens-in-vehicles/`.
Accessed: 2016-03-24.

[3] G Waloszek. Interaction design guide for touchscreen applications (experimental). *SAP Design Guild, Version 0.5*, 2000.

[4] Donald A Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[5] Shunji Mori, Hirobumi Nishida, and Hiromitsu Yamada. *Optical character recognition*. John Wiley & Sons, Inc., 1999.

[6] On a type-reading optophone. `http://rspa.royalsocietypublishing.org/content/90/619/373`.
Accessed: 2016-02-05.

[7] Omnifont ocr, for blind and visually impaired users. `http://everything.explained.today/Optical_character_recognition/#Ref-5`.
Accessed: 2016-02-05.

[8] Jim R Parker. *Algorithms for image processing and computer vision*. John Wiley & Sons, 2010.

[9] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and vision computing*, 21(11):977–1000, 2003.

[10] Divakar Yadav, Sonia Sánchez-Cuadrado, and Jorge Morato. Optical character recognition for hindi language using a neural-network approach. *JIPS*, 9(1):117–140, 2013.

[11] Aamir Khan, Devesh Pratap Singh, Pushpraj Singh Tanwar, and Amit Rajput. Vehicle license plate number recognition and segmentation system. *International Journal on Emerging Technologies*, 2(2):75–79, 2011.

[12] Muhammad Tahir Qadri and Muhammad Asif. Automatic number plate recognition system for vehicle identification using optical character recognition. In *Education Technology and Computer, 2009. ICETC'09. International Conference on*, pages 335–338. IEEE, 2009.

[13] Ivan Kastelan, Sandra Kukolj, Vukota Pekovic, Vladimir Marinkovic, and Zoran Marceta. Extraction of text on tv screen using optical character recognition. In *Intelligent Systems and Informatics (SISY), 2012 IEEE 10th Jubilee International Symposium on*, pages 153–156. IEEE, 2012.

[14] Jisheng Liang, Ihsin T Phillips, Vikram Chalana, and Robert Haralick. A methodology for special symbol recognitions. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 4, pages 11–14. IEEE, 2000.

[15] Markus Schreiber, Fabian Poggenhans, and Christoph Stiller. Detecting symbols on road surface for mapping and localization using ocr. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 597–602. IEEE, 2014.

[16] Ray Smith. An overview of the tesseract ocr engine. In *icdar*, pages 629–633. IEEE, 2007.

[17] Ocr software comparison tesseract vs abbyy finereader, 2015. `http://lib.psnc.pl/dlibra/docmetadata?id=358&from=publication&showContent=true`. Accessed: 2016-02-27.

[18] Tesseract optical character recognition engine. `https://github.com/tesseract-ocr`. Accessed: 2016-02-05.

[19] Leptonica, useful for image processing and image analysis applications. `http://www.leptonica.com/`. Accessed: 2016-04-06.

[20] Apache license 2.0. `http://www.apache.org/licenses/LICENSE-2.0`. Accessed: 2016-02-26.

[21] Javacpp- presets module contains java configuration and interface classes for widely used c/c++ libraries. `https://github.com/bytedeco/javacpp-presets`. Accessed: 2016-02-26.

[22] Lookup ocr, java library. https://github.com/axet/lookup.
Accessed: 2016-02-26.

[23] Tess4j, jna wrapper for tesseract. http://tess4j.sourceforge.net/.
Accessed: 2016-02-26.

[24] Java ocr, java library. https://sourceforge.net/projects/javaocr/.
Accessed: 2016-02-26.

[25] Ocr software comparison tesseract vs abbyy finereader,2010. http://www.splitbrain.org/blog/2010-06/15-linux_ocr_software_comparison.
Accessed: 2016-02-26.

[26] Abbyy finereader engine lets developers,integrate ocr technologies into their applications. http://www.abbyy.com/.
Accessed: 2016-02-26.

[27] Asprise, global imaging and ocr leader since 1998. https://asprise.com/home/.
Accessed: 2016-02-26.

[28] Open source computer vision. http://opencv.org/.
Accessed: 2016-02-19.

[29] Ray smiths slides from tutorial on tesseract, presented at das2014. https://github.com/tesseract-ocr/tesseract/wiki/Documentation.
Accessed: 2016-02-05.

[30] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[31] A Rajavelu, Mohamad T Musavi, and Mukul Vassant Shirvaikar. A neural network approach to character recognition. *Neural Networks*, 2(5):387–393, 1989.

[32] Reza Safabakhsh and Shahram Khadivi. Document skew detection using minimum-area bounding rectangle. In *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on*, pages 253–258. IEEE, 2000.

[33] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[34] Wenshuo Gao, Xiaoguang Zhang, Lei Yang, and Huizhong Liu. An improved sobel edge detection. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 5, pages 67–71. IEEE, 2010.

[35] J. Serra. *Image analysis and mathematical morphology*. Academic Press, London, 1982.

[36] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[37] Tesseract supported languages. `https://github.com/tesseract-ocr/tesseract/blob/master/doc/tesseract.1.asc#languages`.
Accessed: 2016-04-15.

# A P P E N D I X A

# Appendix

## A.1 Tesseract Supported Languages

**Supported languages 2016-04-15** [37]

**afr** (Afrikaans)
**amh** (Amharic)
**ara** (Arabic)
**asm** (Assamese)
**aze** (Azerbaijani)
**aze_cyrl** (Azerbaijani-Cyrilic)
**bel** (Belarusian)
**ben** (Bengali)
**bod** (Tibetan)
**bos** (Bosnian)
**bul** (Bulgarian)
**cat** (Catalan; Valencian)
**ceb** (Cebuano)
**ces** (Czech)
**chi_sim** (Chinese-Simplified)
**chi_tra** (Chinese-Traditional)
**chr** (Cherokee)
**cym** (Welsh) dan (Danish)
**dan_frak** (Danish-Fraktur)
**deu** (German)
**deu_frak** (German-Fraktur)
**dzo** (Dzongkha)
**ell** (Greek, Modern (1453-))
**eng** (English)
**enm** (English, Middle (1100-1500))
**epo** (Esperanto)
**equ** (Math / equation detection module)
**est** (Estonian)
**eus** (Basque)
**fas** (Persian)
**fin** (Finnish)
**fra** (French)
**frk** (Frankish)
**frm** (French, Middle(ca.1400-1600))
**gle** (Irish)
**glg** (Galician)
**grc** (Greek,Ancient(to 1453))
**guj** (Gujarati)
**hat** (Haitian;Haitian Creole)
**heb** (Hebrew)
**hin** (Hindi)
**hrv** (Croatian)
**hun** (Hungarian)
**iku** (Inuktitut)
**ind** (Indonesian)
**isl** (Icelandic)
**ita** (Italian)
**ita_old** (Italian-Old)
**jav** (Javanese)
**jpn** (Japanese)
**kan** (Kannada)
**kat** (Georgian)

**kat_old** (Georgian-Old)
**kaz** (Kazakh)
**khm** (Central Khmer)
**kir** (Kirghiz; Kyrgyz)
**kor** (Korean)
**kur** (Kurdish)
**lao** (Lao)
**lat** (Latin)
**lav** (Latvian)
**lit** (Lithuanian)
**mal** (Malayalam)
**mar** (Marathi)
**mkd** (Macedonian)
**mlt** (Maltese)
**msa** (Malay)
**mya** (Burmese)
**nep** (Nepali)
**nld** (Dutch;Flemish)
**nor** (Norwegian)
**ori** (Oriya)
**osd** (Orientation and script detection module)
**pan** (Panjabi;Punjabi)
**pol** (Polish)
**por** (Portuguese)
**pus** (Pushto;Pashto)
**ron** (Romanian; Moldavian; Moldovan)
**rus** (Russian)
**san** (Sanskrit)
**sin** (Sinhala;Sinhalese)
**slk** (Slovak)
**slk_frak** (Slovak-Fraktur)
**slv** (Slovenian)
**spa** (Spanish;Castilian)
**spa_old** (Spanish;Castilian - Old)
**sqi** (Albanian)
**srp** (Serbian)
**srp_latn** (Serbian-Latin)
**swa** (Swahili)
**swe** (Swedish)
**syr** (Syriac)
**tam** (Tamil)
**tel** (Telugu)
**tgk** (Tajik)
**tgl** (Tagalog)
**tha** (Thai)
**tir** (Tigrinya)
**tur** (Turkish)
**uig** (Uighur;Uyghur)
**ukr** (Ukrainian)
**urd** (Urdu)
**uzb** (Uzbek)
**uzb_cyrl** (Uzbek-Cyrilic)
**vie** (Vietnamese)
**yid** (Yiddish)

## A.2 Full table of recognized symbols

| Symbol | Name | Recognition | Correct | Confidence % |
|---|---|---|---|---|
|  | DISPLAY | DISPLAY | YES | 99 |
|  | MOBILE | MOBILE | YES | 98 |
|  | UNLOCK | UNLOCK | YES | 97 |
|  | DIAMOND | DIAMOND | YES | 97 |
|  | WARNING | WARNING | YES | 96 |
|  | LAPTOP | LAPTOP | YES | 96 |
|  | FIRE | FIRE | YES | 95 |
|  | CLUBS | CLUBS | YES | 95 |
|  | PHONE | PHONE | YES | 95 |
|  | NEXT | NEXT | YES | 93 |
|  | PREVIOUS | PREVIOUS | YES | 92 |
|  | PLAY | PLAY | YES | 92 |
|  | TONGUE | TONGUE | YES | 91 |
|  | STOP | MOBILE | **NO** | 91 |
|  | LIFEBUOY | LIFEBUOY | YES | 91 |
|  | FORWARD2 | FORWARD2 | YES | 91 |
|  | COG | COG | YES | 91 |
|  | CLOCK | CLOCK | YES | 91 |
|  | SAD | SAD | YES | 90 |
|  | BACKWARD | BACKWARD | YES | 90 |
|  | SMILE | SMILE | YES | 89 |
|  | NEXT | FORWARD | **NO** | 87 |

| Symbol | Name | Recognition | Correct | Confidence % |
|---|---|---|---|---|
|  | WRENCH | WRENCH | YES | 86 |
|  | USERPLUS | USERPLUS | YES | 85 |
|  | USERMINUS | USERMINUS | YES | 84 |
|  | BUBBLE | MOBILE | **NO** | 81 |
|  | REPLY | REPLY | YES | 78 |
|  | REDO | REDO | YES | 76 |
|  | PAUSE | MOBILE | **NO** | 73 |
|  | TOUNGE2 | TONGUE | **NO** | 73 |
|  | LOCK | MOBILE | **NO** | 71 |
|  | HAPPY | TOUNGE | **NO** | 71 |
|  | SPADE | MOBILE | **NO** | 69 |
|  | SMILE2 | SMILE | **NO** | 68 |
|  | BUBBLES | MOBILE | **NO** | 66 |
|  | PREVIOUS2 | MOBILE | **NO** | 65 |
|  | WIFI | FIRE | **NO** | 49 |
|  | COGWHEELS | SMILE2 | **NO** | 48 |
|  | FORWARD | PHONE | **NO** | 46 |

Table A.1: *Recognition table of symbols in perfect conditions*