QR-Extraction using OpenCV in Python

Leonhard Feiner, Verena Röhrl, Universidad Pontificia Comillas, Computer Vision

Abstract—This paper describes the Computer Vision project of QR-Extraction using OpenCV in Python. Therefor a set of images taken of QR codes are used to test the software and certain OpenCV features as well as mathematical means help to locate and find the right orientation of the Finder Patterns of the QR code. The general methodology of these operations are explained in this paper. As the set of QR codes vary regarding to different distances and perspectives, the results will be validated in order to show the limitations of the selected approach, that is based on the Finder Patterns.

1 Introduction

QR code is the short form for Quick Response code and is displayed as a 2 dimensional barcode that has been developed by the Japanese company Denso Wave in 1994. The idea back then was to create a code that is easily readable for machines in the automotive industry. This code should contain more information on less space, impose only few requirements on reading devices and it should work if the code is partly soiled or destroyed. Hence, QR codes store data in a pattern of black and white pixels that is not readable for humans.

As already mentioned, these codes were actually developed for the industry, but today they are widely used in the everyday life, for example in advertisements or newspapers to represent complicated or long internet addresses. Furthermore, QR codes are more and more used in museums to quickly get information about a certain object. Therefore, a lot of different QR reader applications for smartphones exist to easily recognize and determine the decoded data.

Fig. 1: Application exampels of QR code

This recognition of QR codes is relevant for Computer Vision, because the taken images are usually not of high quality, but on the contrary they can be distorted due to non-perfect angles and distances of the camera or the lighting conditions might not be optimal and can cause some problems as well. Consequently, appropriate Computer Vision and mathematical techniques, such as image processing, feature extraction and feature matching, should be used in order to detect as many QR codes as possible with respect to their before mentioned non-ideal properties.

The implementation of this QR extractor and reader is done in Python by using OpenCV. OpenCV is a free library that contains a huge amount of Computer Vision algorithms. Intel initiated its development and today it is mainly further developed by Willow Garage. The strengths of OpenCV are the great number of algorithms and its quick performance. Of course, Python is rather slow compared to other programming languages, but nevertheless Python in combination with OpenCV was choosen in order to combine the strengths of both and finally it led to a reasonable good performance.

On the following pages, the main methodology starting with the components of a QR code, followed by the performed Computer Vision and mathematical operations and finally the QR decoding will be explained in more detail. Furthermore, the results are presented and validated. The conclusion helps to get an overview and outlook of the topic and the used references are appended at the end as well.

2 Methodology

2.1 Components of a QR code

First of all, it is important to get an overview of the structure of the QR code and understand its different components.

Fig. 2: Components of QR code

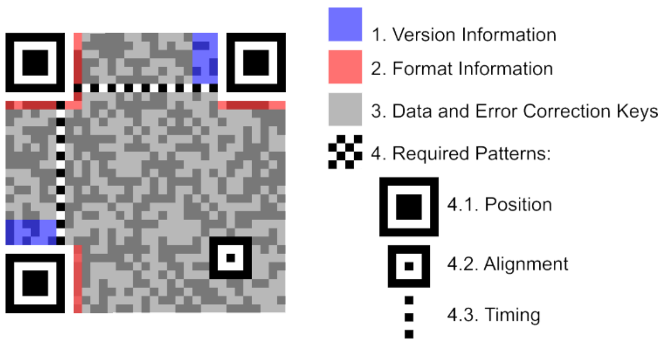


Figure 2 shows the contained information and where in the QR code it can be found. As there are three kinds of obligatory patterns (Position, Alignment and Timing Patterns), there are many different approaches of locating the QR code, depending on what pattern or patterns are the basis of the further processing. This approach is based on the detection of the Position/Finder Patterns.

2.2 Detection of Finder Patterns

After opening an image of a QR code in OpenCV, it is converted to a grayscale picture. This is often done in Computer Vision if the color information is not important at all. After that, the Canny operator is applied to the grayscale image in order to detect the edges in the image.

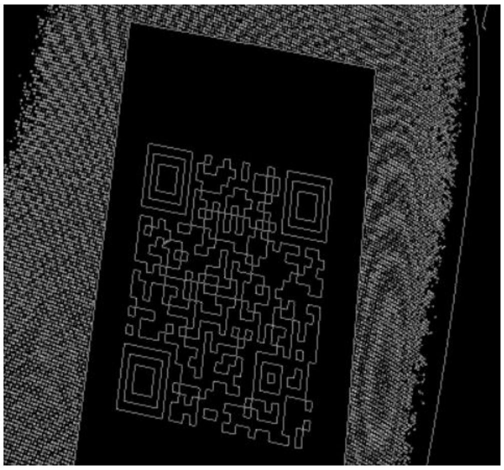


Fig. 3: Application of the Canny operator

Figure 3 is an example of a resulting image. It can be seen, that the detected edges are displayed as white lines on a black background. In order to detect the three Finder Patterns their specific structure and the OpenCV feature Contours is used. This feature looks for contours and enables finding a hierarchy in contours. In other words, it determines whether there are some nested contours and how often they are nested.

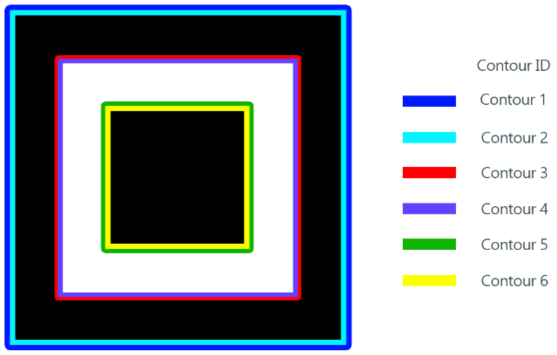


Fig. 4: Contour hierarchy in a Finder Pattern

Because of the Canny operator, the edges are represented as white lines. The Contours feature detects changes from black to white or viseversa as a contour. Consequently, two contours per white line are found and a Finder Pattern consists of 6 contours in total, like shown in Figure 4. Therefor, the developed algorithm looks for five times nested contours, stores all contour points and by use of the function moments, the center points of each Finder Pattern are calculated.

2.3 Matching of Finder Patterns

Subsequently, as the three Finder Patterns and their location have been determined, they have to be matched to the right orientation.

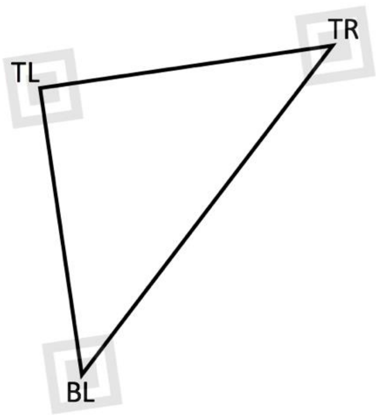
Therefor, the vectors between the center points are used and it can be followed, that the vector with the biggest distance between the two center points describes the vector between the Finder Pattern Bottom Left (BL) and the Top Right (TR). Hence, the Finder Pattern Top Left (TL) can be clearly distinguished.

In addition, the cross following cross product

$$\overrightarrow{BLTL} \times \overrightarrow{TRTL}$$

is computed and with the sign of the result it is possible to differentiate between the Finder Patterns BL and TR as well.

Fig. 4: Matching of Finder patterns using a triangle



2.4 Determining Corners of Finder Patterns

For purposes that are mentioned in the following section, the corner points of each Finder Pattern have to be determined. This is done by two lines for each Finder Pattern, displayed red in Figure 5. These lines come from these vectors:

$$\overrightarrow{BLTL}\,;\ \overrightarrow{TRTL}$$

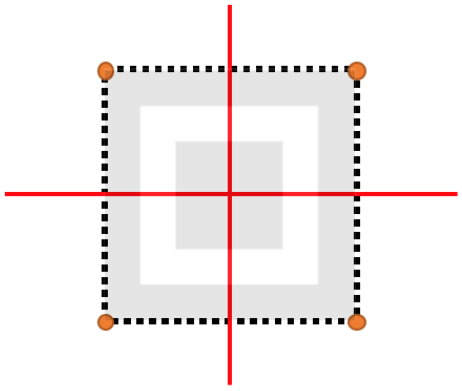


Fig. 5: Determining corners of Finder Patterns using two lines

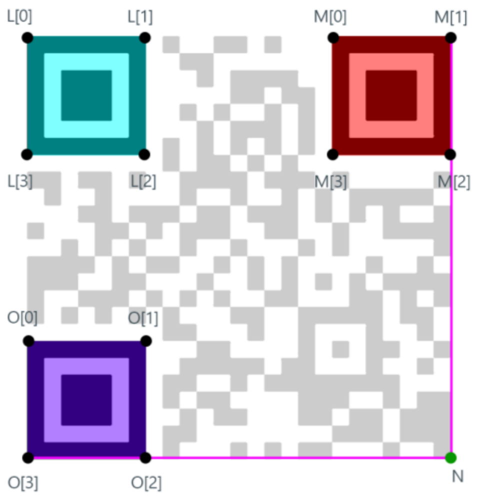
In Figure 5, the outer contour line is displayed by many contour points, that are generated by the Contours feature as described in section 2.2. The two lines divide the contour points in four areas. The corner point of each section is the point that has the biggest distance to the center point.

2.5 Determining 4th corner

In order to get the required information for the following section, the fourth corner point of the whole QR code has to be determined.

This is done by extrapolating the two vectors of the edges of the Finder Patterns BL and TR between the points Q[3] and Q[2] and between M[1] and M[2] in the Figure 6.

Fig. 6: Extrapolation to determine fourth corner



This extrapolation sometimes seems to be quite inaccurate. Hence, another approach is used to improve this.

Therefor, the vector between the center points of the Finder Pattern BL and TR is used to find the midpoint between them (yellow in Figure 7). Furthermore, a new vector between the bottom right corner point of the Finder Pattern TL and the midpoint calculated before is computed and appended to the mentioned midpoint. This leads to the area where the Alignment Pattern is located.

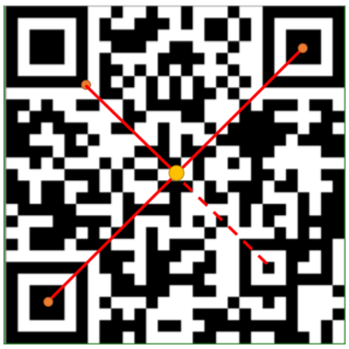


Fig. 7: Extrapolation of Alignment Pattern

In this area, the contours image is taken again and in order to get the center point of the Alignment Pattern, a three times nested contour is required.

This second approach leads to much better results than the first one. But in QR codes of version 1, there are no Alignment Patterns contained. Hence, in most of the cases it is possible to use the better second approach, only in some cases it is necessary to use the first method.

2.6 Amount of pixels/Version

In order to decide which version is used, the following computations are needed.

First of all, the number of pixels are calculated by taking the means of the pixels horizontally and vertically of the whole image and by taking the means of the pixels of all Finder Patterns horizontally and vertically as well. As a Finder Pattern in an original QR code has a length of seven pixels, the final length is computed with the rule of three. The version is represented as an integer value and can be determined by this formula, where v is the version and number of pixels represent the number of pixels horizontally or vertically. The number of pixels is rounded in order to get an integer number for the version.

$$number\ of\ pixels = 17 + v * 4$$

2.7 Warping the perspective

As the images taken of the QR codes are usually not perfectly fitted and somehow twisted or rotated, the QR code has to get rewarped to the desired squared form. This is done by taking the four corner points of the whole QR code as the Figure 8 shows.

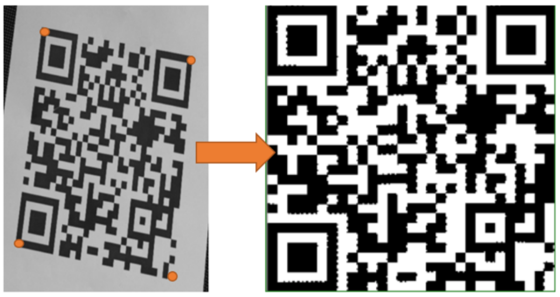


Fig. 8: Rewarping using the corner points

In order to get desired black and white images, a thresholding is performed. After that, it can be seen in Figure 8, that the resulting image is quite good, but there are some artifacts remaining.

2.8 Bit extraction

Finally, the QR code is resized to the desired dimensions that have been computed in section 2.7. Performing thresholding again removes the artifacts mentioned in the section before. As a result, an extracted bool matrix is obtained.

2.9 Decoding

As common QR readers contain a certain decoder with error correction, it is helpful to implement a decoder for this project as well to be able to validate the results. This required a lot of research effort as this is done by many mathematical algorithms and as this is not related to Computer Vision at all, it will not be further explained.

3 Results

This implementation of a QR reader is tested using a set of images taken of different QR codes of different perspectives, angles and distances. Therefor, the algorithm that has been explained in section 2 is applied on all of these images of this dataset. Furthermore, the number of total iterations and successful iterations are counted, the resulting probability of successful decoding is calculated and the decoded information contained in the QR code is displayed if the

decoding was successful, otherwhise an error message is printed. This test set results in a probability of successful decoding of **72%** which is a reasonable good result when considering that there are images taken in quite extreme angles and perspectives.

The main aim of the developed algorithm is to get a quick overview of the validation, so it is not very failsafe in general.

When taking a closer look at the failed iterations, it can be seen that one of the main problems is caused by the finding of contours, that has been described in section 2.2. Sometimes there are not exactly three Finder Patterns detected and as a consequence the algorithm fails and the error message "Detected X Finder Pattern. Exact 3 are required!" is printed on the console.

In addition, if the image of the QR code is not flat on the surface, the picture of the QR code will appear a bit curved. This leads to some problems of the rewarping and one of the two error message "Could not locate error" or "Too many errors to correct" is displayed. This message comes from the error correction function reed-solomon.

Finally, the assumption that was made in section 2.3 for using a triangle to distinguish between the three Finder Patterns can lead to problems when there are acute angles of the camera.

In the remaining part of this section, the above-mentioned problems should be visualized with real images of the used data set.

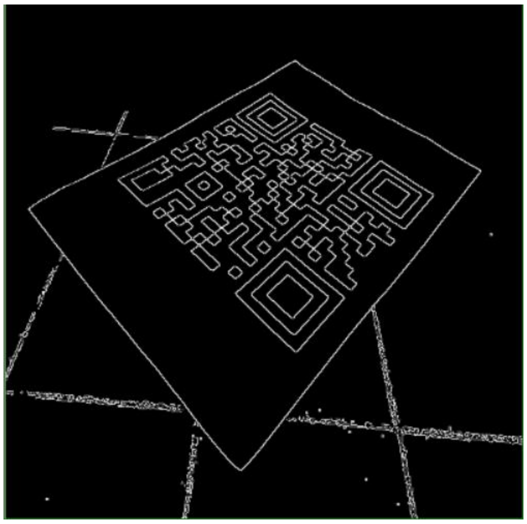


Fig. 9: Example of a warping error – Canny image

In the figure 9 there can be seen, especially in the top left edge of the sheet that contains the QR code, that the sheet is not totally flat and so the rewarped image in figure 10 exhibits a clear warping error.

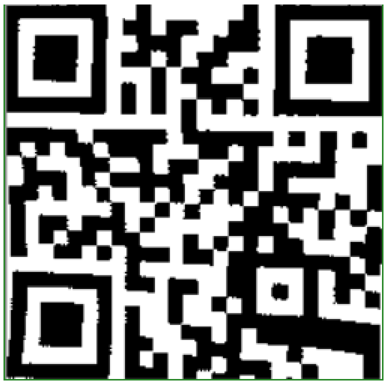


Fig. 10: Example of a warping error – rewarped image

Regarding the problem with contours, in figure 11 it can be seen that only the Finder Patterns TL and TR are detected. Furthermore, in the Finder Pattern BL there is an irregularity in the outer edge. This is because of some light/reflection effects of the original image, shown in figure 12. Hence, the algorithm is quite sensitive to such irregularities and sometimes this leads to problems determining the three Finder Patterns.

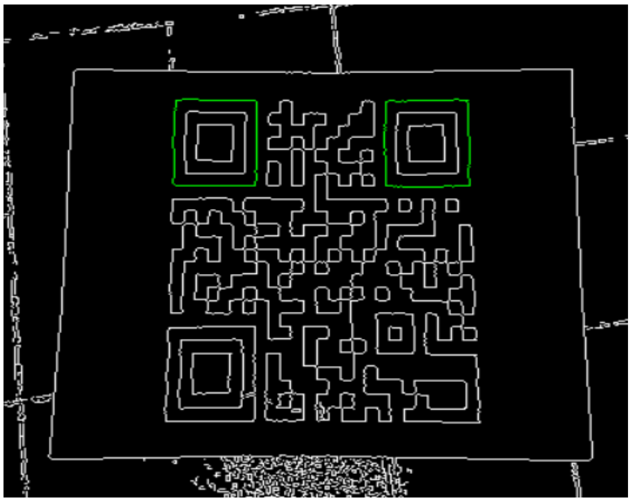


Fig. 11: Example of Finder Pattern error – Canny image

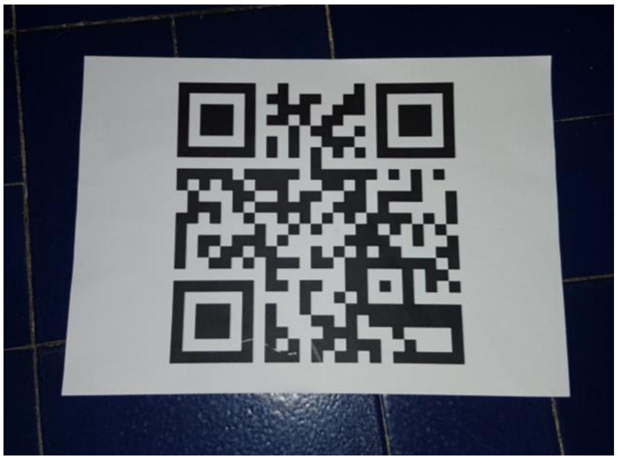


Fig. 12: Example of Finder Pattern error – original image

But on the other hand in figure 14 can be seen, that despite of a nonhomogeneous environment (figure 13) in which the image of the QR code is taken, the result of the warping is very good even for QR codes that contain a lot of data.

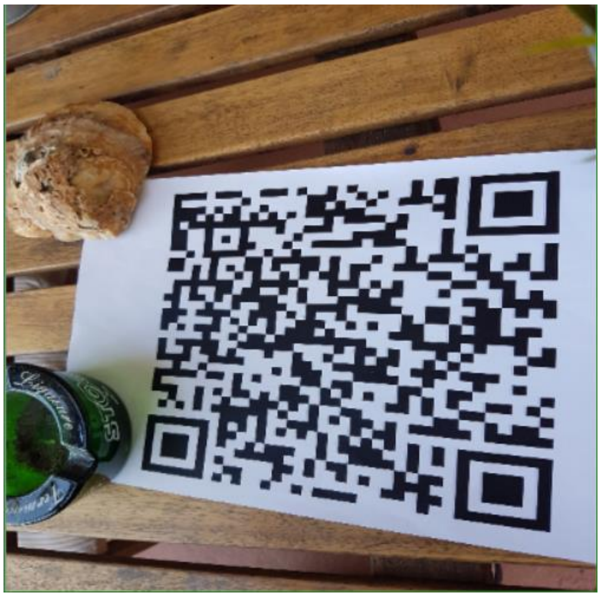


Fig. 13: Very good result – original image

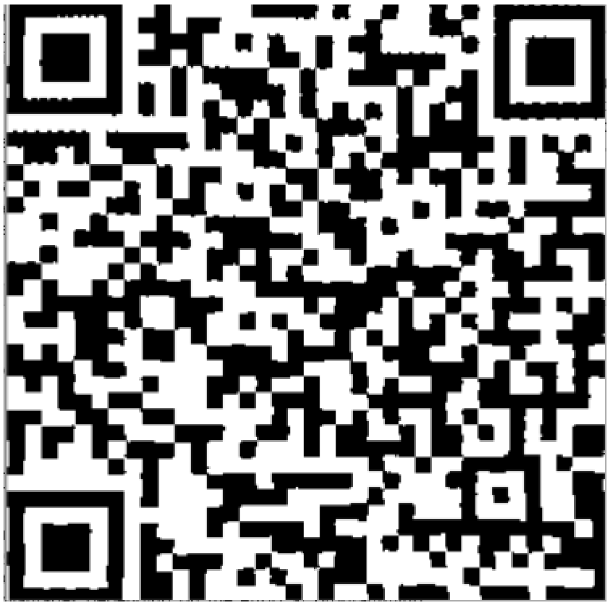


Fig. 14: Very good result – rewarped image

4 Conclusion

When searching online, a lot of different implementations of QR readers can be found. Hence, many various approaches are used, that lead to different advantages and disadvantages depending on the effort at programming. Most implementations use a lot of pixelwise computations and mathematic algorithms and only few Computer Vision algorithms. Consequently, the very first approach used in this project was to reimplement the ZXing library. But soon it could be realized, that this is not an appropriate approach for a Computer Vision project. Finally, in one of the later mentioned resources a Computer Vision approach is mentioned and has been used further. This approach is the root of the described algorithm. But solely a reimplementation of this led to very poor results. As a consequence, a great majority of the described algorithms has been developed without any example source code. The described methodology is the result of many, many different approaches. For example, there was an approach to do the extrapolation of the 4th corner (section 2.5) by using the Hough transform. The algorithms, that are presented in this paper, led to the best result. When taking a look at common implementations of QR readers like for example ZXing the most obvious difference is the only use of pixelwise computations. In addition, ZXing for instance contains thousands of code lines, whereas this implementation only covers about 600 code lines. Consequently, it seems like pixelwise processing of QR codes lead to different advantages and disadvantages than Computer Vision approaches.

5 References

[1] A. Hengstbach, "QR-Codes: lessen, erzeugen, verstehen", Computerbild, http://www.computerbild.de/artikel/cb-Tipps-Wissen-QR-Codes-Barcode-EAN-6122468.html

[2] OpenCV Team, "About", OpenCV, http://opencv.org/about.html

[3] Bharath P., "OpenCV: QR Code detection and extraction", Dsynflo Blogspot, http://dsynflo.blogspot.com.es/2014/10/opencv-qr-code-detection-and-extraction.html

[4] Bharathp666, "OpenCV: QR Code detection and extraction", Github, https://github.com/bharathp666/opencv_qr

[5] OpenCV Team, "Contours Hierarchy ", Docs OpenCV, http://docs.opencv.org/trunk/d9/d8b/tutorial_py_contours_hierarchy.html

[6] Krasnaya Ploshchad, "Official ZXing ("Zebra Crossing") project home", Github, https://github.com/zxing/zxing

[7] A. Fuller, "Decoding small QR codes by hand", Solder and Flux, http://blog.qartis.com/decoding-small-qr-codes-by-hand/

[8] Robomatics, "How to Decode a QR Code by Hand", YouTube, https://www.youtube.com/watch?v=KA8hDldvfv0

[9] Wikipedia, "QR Code", https://en.wikipedia.org/wiki/QR_code

[10] Carolyn Eby, "QR Code Tutorial", Denso Wave, http://www.thonky.com/qr-code-tutorial/

[11] Fig. 1: Application examples of QR code https://upload.wikimedia.org/wikipedia/commons/7/7f/Qr-projekt-taunusanlage-beethoven-denkmal-2011-ffm-029.jpg

[12] Fig. 1: Application examples of QR code https://upload.wikimedia.org/wikipedia/commons/3/32/Japan-qr-code-billboard.jpg

[13] Fig. 1: Application examples of QR code https://upload.wikimedia.org/wikipedia/commons/f/f9/Munzee.jpg

[14] Fig. 1: Application examples of QR code

https://upload.wikimedia.org/wikipedia/commons/a/af/Japan_Visum.png
[15] Fig. 2: Components of QR code http://2.bp.blogspot.com/-gb52V580Pms/VEu10EYnqsI/AAAAAAAACk8/5I9Koh7XZsQ/s1600/qr-code-parts.png
[16] Fig. 4: Contour hierarchy in a Finder Pattern http://4.bp.blogspot.com/-WDDpcW4qCMY/VEjlP0hf3tI/AAAAAAAACkI/GnO9CaSFt0Y/s1600/markers.png
[17] Fig. 6: Extrapolation to determine fourth corner http://3.bp.blogspot.com/-1sGsWJKNCtY/VD_gygx1MKI/AAAAAAAACjU/qoIH32Ia5Y0/s1600/naming.png