Ca' Foscari
University
of Venice

Master's degree
in Computer science
Software dependability and Cyber Security

Final Thesis

A quantitative evaluation of the QR code detection and decoding performance in the zxing library

Supervisor
Ch. Prof. Filippo Bergamasco

Graduand
Daniele Barzazzi
Matriculation Number 863011

Academic Year
2021 / 2022

Abstract

In the year of the global pandemic there was an increment of the usage of QR code due to the development of the EU Digital COVID Certificate, among others type of certificate of other countries. This lead us to the problem of mobile phone having difficulties to reading the QR code. In this thesis, we evaluate the QR code detection and decoding performance of a popular open-source library by applying different image noise models. Our approach works by simulating several image degradation factors like thermal noise, perspective distortion, defocus, and Moirè patterns originated when capturing an LCD screen. Experimental results show that the detection part plays a significant role and, surprisingly, the error-correction capability of the marker might be inversely proportional to the decoding rate.

Contents

1 Introduction

In recent years, QR codes have gained widespread usage. A growing number of states have implemented the use of QR codes for payments in an effort to shift away from cash and promote electronic transactions. In Europe, the increased adoption of QR codes may have been accelerated by the global COVID-19 pandemic and the development of the EU Digital COVID Certificate. This specific QR code was necessary to access many places like cinema or theater, in order to ensure that the people participating where vaccinated, had recovered from the disease, or at least had been tested negative from a COVID test. This situation in which lots of people's QR code had to be scanned showed that in multiple occasion the scanning was not immediate and in some cases was almost never working.

The main idea on why there was such problem was because of the implementation. The structure of a EU Digital COVID Certificate is complex and is composed by lots of data which means that the QR code generated is large and dense. The density of a QR code makes it difficult for a library to distinguish between different squares in the matrix. Two previous thesis from Marco Carfizzi[6] and Giacomo Arrigo[5] discussed about how to improve the DCC by reducing the version of the QR code by removing unnecessary data and trying different data structure to save the payload, plus reducing the error correction level for the generation of the QR code.

1.1 Objectives

In this thesis we will discuss how we simulated the scanning of a QR code from a smartphone camera by applying different type of image noise models. Usually people used to carry the DCC printed on paper or saved in their smartphone so we divided the project in two type of camera simulation. In the first we applied image degradation factors like thermal noise, perspective distortion, defocus. In the second, to simulate the QR code on a smartphone, we added a Moirè patterns originated when capturing an LCD screen. The goal it to understand which condition or which parameter were most influencing the good result of a scan. In particular, we studied the trade-off between the lowest and highest error correction values, as there is a significant difference in the versions generated between the two. This was because we wanted to determine if the trade-off would be valuable, considering both QR codes had the same encoded data inside them.

2 QR Code

Quick response (QR) codes, developed by Denso Wave in 1994, are a type of barcodes. QR codes are a license-free standard specified in ISO/IEC 18004:2006[10]. Data are encoded in a matrix formed from black square placed on a white background which encode a string of text(Figure 1).

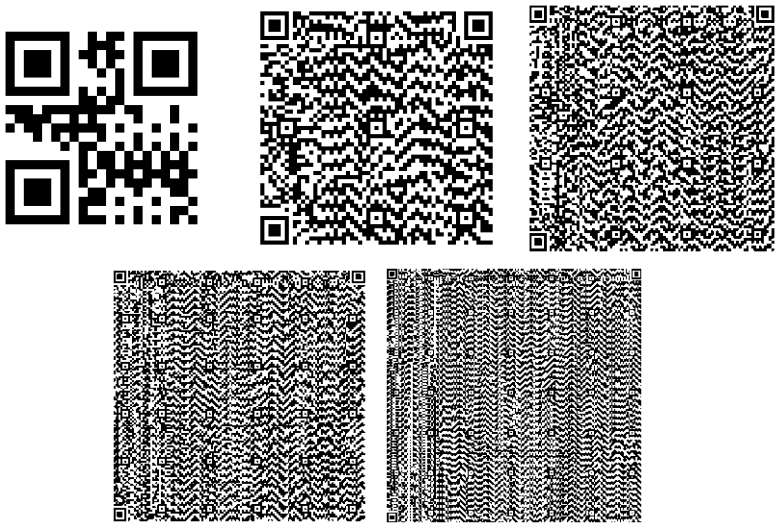

Figure 1: Different versions of QR codes

There are four modes of encoding:

- Numeric mode is for decimal digits from 0 to 9
- Alphanumeric mode is for decimal digits, letters only in uppercase and the symbols $, %, *, +, -, ., /, : plus the blank space
- Byte mode is for characters from the ISO-8859-1 character set, 8 bit for char.

- Kanji mode is for double-byte, 16 bit, characters from the Shift JIS character set.

A QR code has a standard structure (Figure 2). It contains three big markers, called position marker positioned on the top-left, top-right and bottom left corner used for detecting the position and the size. Some smaller marker on the inside are used as alignment pattern used to correct distortion of the QR code and the timing pattern that are two dotted line that connect position markers. On the outside, a quiet zone is used to make easier to detect the QR code. The remaining area is called data area but some regions, called reserved area, are used to insert version information, format information and data and error correction keys.

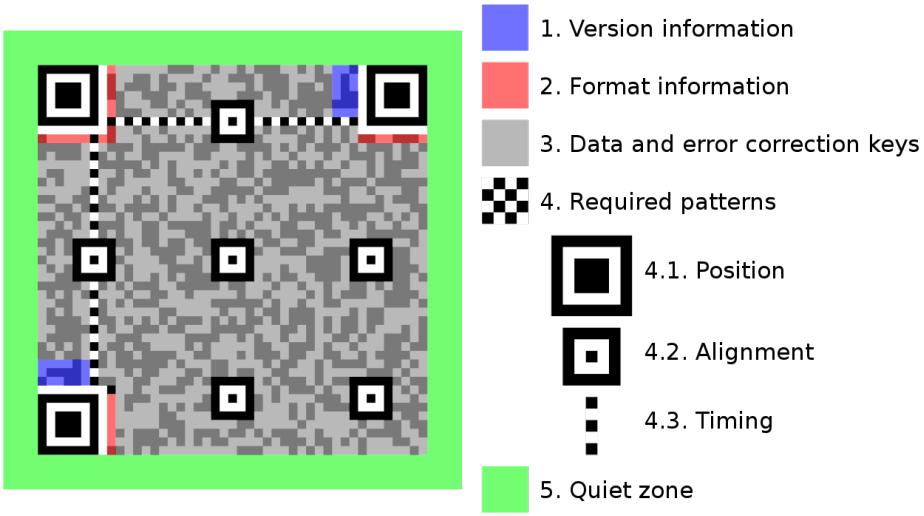

Figure 2: QR code structure[4]

QR codes have different possible versions(Figure 1) that goes from version 1 with size 21×21 modules, a module is a single square of the matrix that it will not be called pixel because it can differ in size from a real screen

pixel, to version 40 that is 177×177 modules. Every version differs from the previous one by 4 modules. The number of character that can be encoded in each version depends on the mode in use. All the space not covered from the reserved area is filled by codewords, 8 bit long, that are divided between data codewords and error correction codewords. Depending on the version, the total number of blocks is fixed from 26 of version 1 to 3706 of version 40. Data are inserted starting from bottom right to top left, first the data codewords and then the error correction ones, Figure 3.

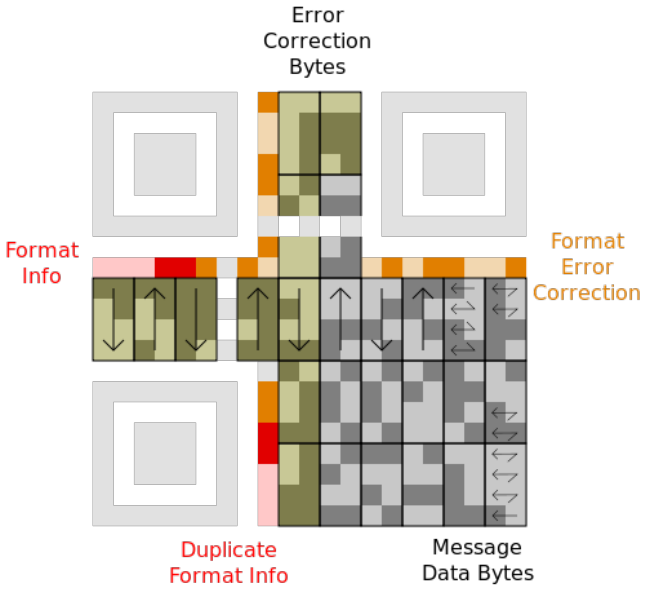

Figure 3: Data and error correction codewords[3]

2.1 Error Correction

The error correction blocks allows QR code decoder to detect and correct errors. There are four level of error correction:

1. level low, L, that cover up to 7% of the damage
2. level medium, M, that cover up to 15% of the damage
3. level quartile, Q, that cover up to 25% of the damage
4. level high, H, that cover up to 30% of the damage

The error correction is based on the Reed–Solomon codes[11] that are able to detect and correct half of the number of error correction codeword. For example, in a QR code of version 2 with 44 codewords (level Q) we can recover up to 11 of them but we will need 22 of them for the error correction. For larger versions the codewords are divided on group and block, even so for high error correction level and long sequence of data the QR code becomes very large and dense.

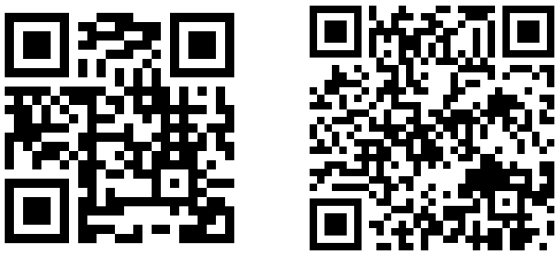

Figure 4: Same data with error correction level low and high

2.2 Data Masking

Data masking is an operation in which some modules are switched from black to white and from white to black, in order to make the QR code more readable for a decoder. The QR code specification gives eight possible mask patterns with their relative formula (see Figure 5). Each formula tell us if for the given i and j the respective module has to be flipped or not, the % operator stands for module that return the remainder of a division between two numbers.

Those patterns have to be applied only on data and error correction modules not on marker, version information, and format information. To evaluate the best mask, all patterns must be tested with all penalty rules. Then, the one with the lower score is picked. To calculate the penalty score there are four rules to be applied:

1. gives a penalty for each rows of 5 consecutive modules of the same color;
2. gives a penalty for each square 2×2 of same color modules;
3. gives a penalty if there are patterns similar to the position pattern;
4. gives a penalty if there are more than double modules black then white or vice versa.

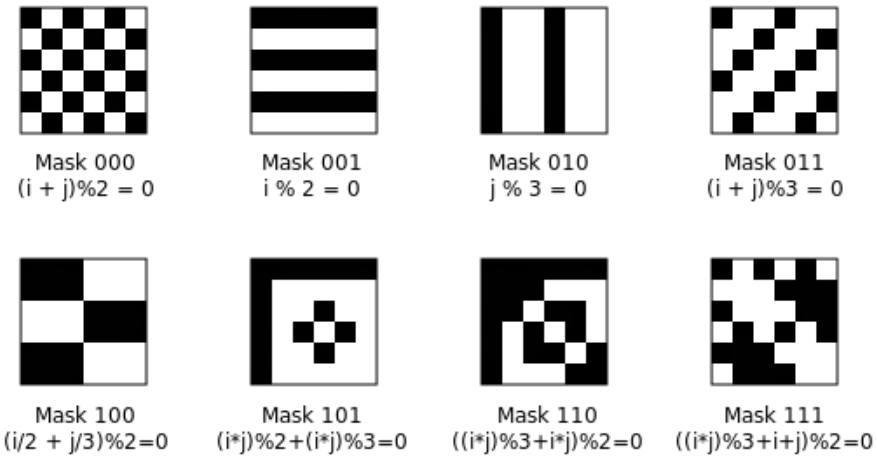

Figure 5: Data mask patterns[1]

3 Digital Image

In the context of computer vision a digital image is a function $f(x, y)$ where the x and y are the coordinates of a single point in the image, called pixel, and it returns the intensity of that single pixel. Taking in example an image of 800×500 pixels we have:

$$f(x,y) = \begin{pmatrix} f(0,0) & f(0,1) & \cdots & f(0,499) \\ f(1,0) & f(1,1) & \cdots & f(1,499) \\ \vdots & \cdots & \ddots & \vdots \\ f(799,0) & f(799,1) & \cdots & f(799,499) \end{pmatrix} \tag{1}$$

In the case of a gray scale image the intensity just need a values form 0 to 255, but since we want to study a scenario with a simulation of a real smartphone camera we have a color image as input. Visible colors can be represented by a mixture of three primary color: red, blue and green. So, the function $f(x, y)$ will return a triple i.e. a black pixel will be $(0, 0, 0)$, the most common order is RGB but sometimes even other variant can be found like BGR.

3.1 Perspective Distortion

When a person take a photo of a planar object, like the QR code, the image look distorted, if the angle of view differs form the perpendicular viewpoint. Another factor can be the distance from the object, when closer it looks bigger and when far it looks smaller. In Figure 6 we can see an example of perspective distortion applied to a square. On the left there is the normal

object and the right the same object distorted. The square is formed with four edges (A, B, C, D) and in the distortion they become (A', B', C', D'), this is to represent the fact that they have been translated by the change of the point of view. We can describe the translated point as:

$$A' = A + dA \tag{2}$$

In general, lengths and angles change depending by the point of view while straight lines remain straight (The transformation is linear in the 2D projective space).

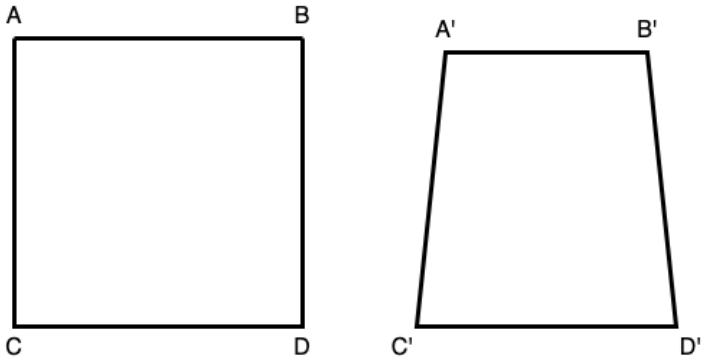

Figure 6: Example of distortion of an object

3.2 Defocus

Defocus is a type of blur that occurs in a image when the camera is not correctly focused on the subject. It can happen because of lens defocus on a stationary subject or because the subject is moving. What we see on the resulting image is blurred image in which it could be difficult to

understand what it has been captured, depending on the level of the blurring. Mathematically, it can be defined as the convolution of the ideal clear image f and the point-spread function h as : $g(x, y) = (f * h)(x, y)$ [8] in our approach we will apply a Gaussian function.

Figure 7: Comparison of a clear image and a blurred one

3.2.1 Gaussian Blur

Gaussian blur is a smoothing lowpass spatial filter using a bell shaped Gaussian (Normal) distribution. To compute the resulting image, a $2D$ Gaussian

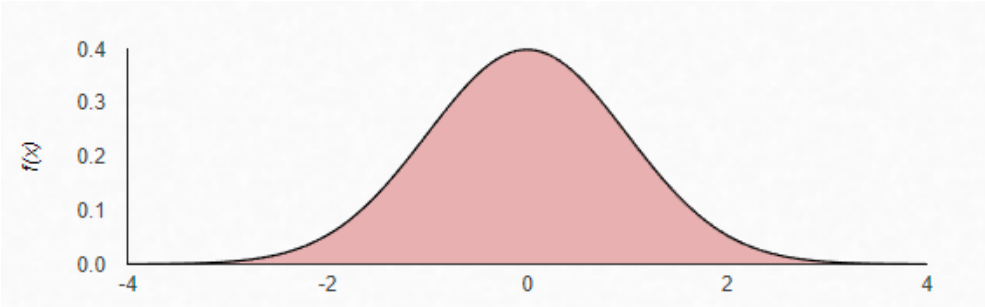

Figure 8: Gaussian or Normal distribution with $\mu = 0$ and $\sigma^2 = 1$

kernel is used, which is a square matrix that has to be convolved with the clear image. The kernel is computed from a zero-mean $2D$ Gaussian function:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

where x and y are the coordinate and σ^2 is the variance of the distribution. As we can see from equation 3, the grater is σ^2 the more blurred the output image will be. Having the kernel, we can define the resulting image $b(x,y)$ as the convolution of the base image $f(x,y)$ with dimension $H \times W$ with the Gaussian filter:[12]

$$b(x,y) = \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} G(i,j) f(x-i, y-i) \tag{4}$$

3.3 Thermal Noise

Noise in a image means a variation of intensity of each pixel and it can be defined as:

$$g(x,y) = (f * h)(x,y) + n(x,y) \tag{5}$$

where the convolution of the clear image $f(x,y)$ and the filter $h(x,y)$ is discussed in section 3.2, while $n(x,y)$ is called additive noise and represents how much the intensity of the single pixel is being altered. This alteration can be caused by various environmental factor such as high or low luminosity exposure. The matrix $n(x,y)$ has the same dimension of the clear image and the values can be simulated by generating intensity values form a probability density function. In our approach we will take in consideration a Gaussian

noise.

Figure 9: Comparison of an image without noise and with noise

3.3.1 Gaussian Noise

The Gaussian noise is a type of noise that arise when the camera is capturing the image, this could happen due to the illumination of the environment or the temperature of the sensor. As said before the noise is additive and it is applied to the single pixel taking in consideration the sampling of a probability density function. In the case of Gaussian noise the sampling is done on the normal distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{6}$$

Where μ is the mean and σ is the standard deviation, the more we increase sigma and the more noise will be applied on the resulting image. When dealing with additive noise, we typically consider it to have zero mean.

3.4 Moire Pattern

A moiré pattern is a visual artifact that can happen when two grid of the relative same dimensions are overlapped creating a pattern with frequencies not present in either of the original patterns[16].

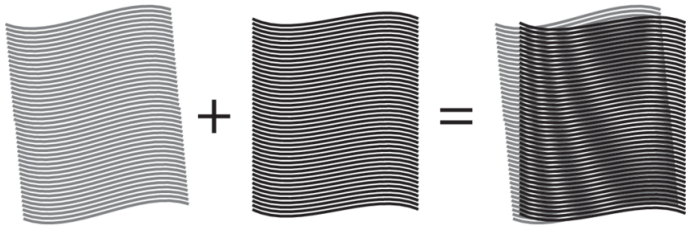

Figure 10: Moiré pattern example[9]

It usually occurs when someone try to take a photo of a screen because both the display and the camera have a grid system. In a LCD display each pixel is composed by three RBG sub-pixel, that can represent each possible visible color by incrementing on lowering the values of the sub-pixels(Figure 11).

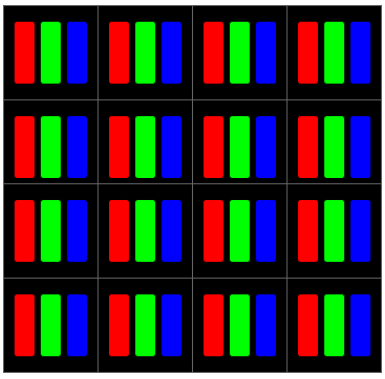

Figure 11: Example of the sub-pixel of a LCD display

Camera sensor is not able to capture colors directly, so a Bayer color filter array is applied in order to be able to understand which color has to be

assigned to each pixel. This operation is possible by using a matrix with a specific arrangement of RGB cells (Figure 12). To evaluate the RGB value of the resulting image, an operation called demosaicing is used. It performs an interpolation of the pixels from the filter to reconstruct the intensity of the pixel in the output image.

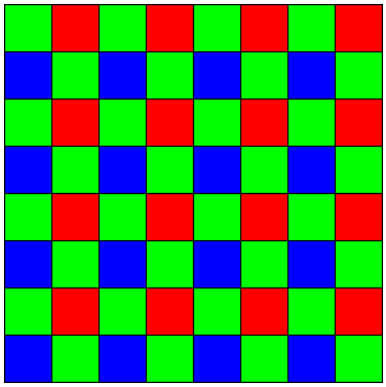

Figure 12: Example of a Bayer CFA

The moiré pattern occurs when the projected pixel grid of the screen have similar size of the pixel grid of the camera. This creates a typical periodic noise pattern that can severely hinder the marker detection.

4 Analysis of the Libraries

We started by choosing a QR code decoding library. After looking at different possibility, we decided for ZXing library because is widely used with a good accuracy, open-source and it was the library used by the android application *VerificaC19*, the Italian customization of the EU Digital COVID Certificate Verifier App[7]. The fact that the library was the one used on the DCC decoder in Italy made a huge impact on the final decision.

4.1 ZXing Library

ZXing, zebra crossing, is an open-source, multi-format 1D/2D barcode image processing library implemented in Java, with ports to other languages[14]. The library support different type of barcodes:

1D product	**1D industrial**	**2D**
UPC-A	Code 39	QR Code
UPC-E	Code 93	Data Matrix
EAN-8	Code 128	Aztec
EAN-13	Codabar	PDF 417
UPC/EAN Extension 2/5	ITF	MaxiCode
		RSS-14
		RSS-Expanded

The library is divided in components:

- core: The core image decoding library, and test code;
- javase: JavaSE-specific client code;

- android: Android client Barcode Scanner;
- android-integration: Supports integration with Barcode Scanner via Intent;
- android-core: Android-related code shared among android, other Android apps.

QR code decoding involves the following steps:

- Image acquisition: an image is captured using a camera or passed to to the library;
- Image preprocessing: The image is converted in grayscale and resized to the dimension in which a single module is equal to a screen pixel, in order to have a standard that make easier the detection;
- Binarization: The image is transformed into a binary form to make it easier to detect and decode;
- QR code detection: The library parse the bit matrix for QR code patterns, such as the alignment patterns and the finder patterns, then extract the format and version information;
- QR code decoding: The library decodes the QR code by reading the encoded data and converting it into the original information, applying the error correction to detect error and correct them if possible.

4.1.1 Change on the Library

To be able to test lots of QR code it has been decided to adopt the Java core module and made it run on IntelliJ IDEA. Being an open-source library it

made possible to change some class and function in order to extract more information on the decoding. The vanilla version of the library given a QR code it returns the decoded text if all goes right and throws an exception if somethings go wrong. Since one of the goal was to understand how much the error correction affects a good scan it was needed to extract information about it.

The *DecoderResult* class in package *com.google.zxing.common*, that encapsulates the result of decoding a matrix of bits:

```
1 DecoderResult(byte[] rawBytes, String text, List<byte[]>
    byteSegments, String ecLevel, int saSequence, int saParity
    , int symbologyModifier)
```

it has been changed to accept also the byte value of the codewords before the application of the ReedSolomon algorithm and store them together with the raw byte corrected:

```
1 DecoderResult(byte[] rawBytes, byte[] notCorrectedBytes,
    String text, List<byte[]> byteSegments, String ecLevel,
    int saSequence, int saParity, int symbologyModifier)
```

The DecoderResult object is being created as result of the *decode* method on the class Decoder at package *com.google.zxing.qrcode.decoder*:

```
1 private DecoderResult decode(BitMatrixParser parser, Map<
    DecodeHintType,?> hints)
```

in which it has been made a copy of the array containing the parsed raw byte before being passed the the method *correctErrors* in order to have both the codewords raw and corrected passed to the DecoderResult.

Having if the error correction was applied or not when a code is decoded

without error it was needed to handle when the detection or decoding failed. To do so it was decided, instead of modifying the library, to just catch the exception on the java file in which the test was run:

- *com.google.zxing.NotFoundException*: the QR code not detected;
- *com.google.zxing.FormatException*: error on the decoding of the QR code;
- *com.google.zxing.ChecksumException*: number of codewords to be corrected grater than maximum number recoverable, based on error correction, see section 2.1.

The function on the test file takes a directory as input and parse all the QR codes inside it. To test different QR codes, they are divided on different directory. In return it gives a number of *.txt* equals to the different level of scaling and noise. In each file each raw is a different level of blur and a raw has:

- total number of QR parsed;
- total number of success;
- total number of fail;
- total number of unsuccessful detection;
- total number of unsuccessful decoding;
- total number of unsuccessful application of error correction, too much errors to repair;
- total number of success with the application of the error correction.

4.2 OpenCV

OpenCV, Open Source Computer Vision Library, is an open-source library specialized in computer vision and machine learning. The library boasts an extensive collection of over 2500 optimized algorithms, comprising classic and cutting-edge computer vision and machine learning techniques. These algorithms enable tasks such as face detection and recognition, object identification, and a lot more[2]. The library is mostly written in *C++*, but it is ported in JAVA, Python and MATLAB.

For our case we decided to opt for Python, to implement the noise model on the images. This choice was made due to our will to learn it better and because there was a lot of different library useful for our study.

5 Implementation of the Models

The software has to take in input two QR code and apply the various model in parallel so that the resulting image could have the same amount of noise in order to be comparable. Given the two QR codes it is possible to choose if the test has to generate the moiré effect or not. The steps performed that will be discussed in the following sections are:

1. generate different scaling of the QR code to simulate different distance from the camera;
2. apply moirè or not to the QR code(Figure 13);
3. generate different homography of the QR code;
4. generate blur and noise to the images.

In the case of the application of the moiré pattern, steps 2 and 3 are fused together.

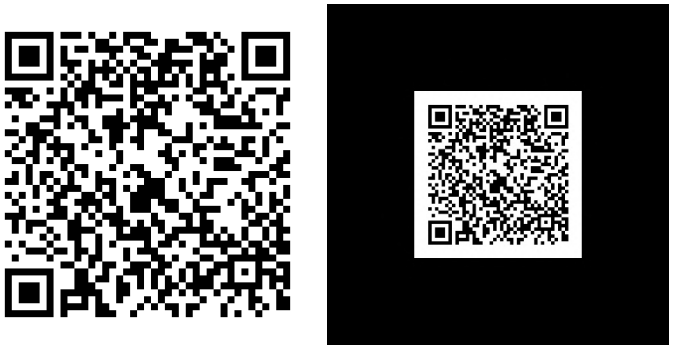

Figure 13: Example of scaling

5.1 Implementation of the Moiré

To simulate a moiré pattern, described in section 3.4, is needed to execute the following steps[13]:

1. to simulate a LCD screen is needed to resample the input image, splitting each pixel into a representation of a RGB sub-pixel of the display. The image is represented in a multidimensional matrix in which at each cell of the $W \times H$ matrix contain a triple of the BGR value of the pixel. So in the output image each pixel needs to be transformed in a 3×3 sub-matrix, Fig 14

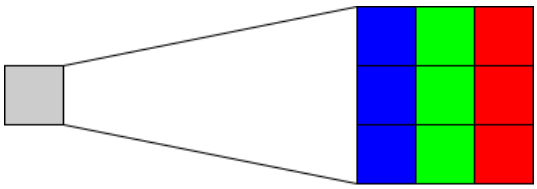

Figure 14: Resample of a pixel

In general, if the pixel of the input image is on $f[i, j]$ then the resulting submatrix will be in position:

$$\begin{pmatrix} [i \cdot 3, j \cdot 3] & [i \cdot 3, j \cdot 3 + 1] & [i \cdot 3, j \cdot 3 + 2] \\ [i \cdot 3 + 1, j \cdot 3] & [i \cdot 3 + 1, j \cdot 3 + 1] & [i \cdot 3 + 1, j \cdot 3 + 2] \\ [i \cdot 3 + 2, j \cdot 3] & [i \cdot 3 + 2, j \cdot 3 + 1] & [i \cdot 3 + 2, j \cdot 3 + 2] \end{pmatrix} \tag{7}$$

In which in the first column there are the intensity value of blue, in the center column green and red in the last. The resulting matrix will be three times the one in input, $3W \times 3H$.

Figure 15: Resulting image of the LCD sampling

2. Apply a Gaussian filter to simulate anti-aliasing filter (Figure 17). Aliasing is a phenomenon where different signals become indistinguishable upon sampling. In computer graphics, it arises during the rendering and reconstruction of an image. To enhance the visual quality of polygon edges, anti-aliasing is commonly employed in computer graphics, making the jagged edges appear smoother on the screen[15].

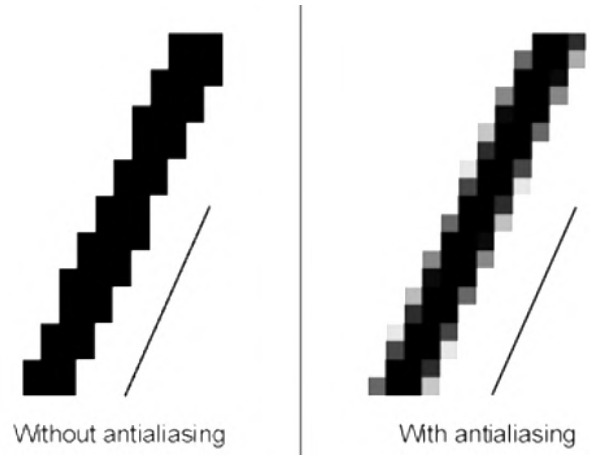

Figure 16: Aliasing vs Anti-alising[15]

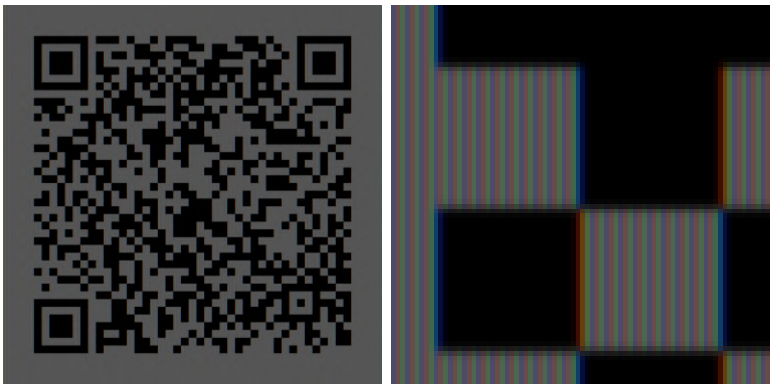

Figure 17: Simulation of the anti-alising filter

3. Generate a perspective distortion, to simulate that the image is captured from a point of view that is not perpendicular with the display. It will be discussed in section 5.2.

Figure 18: Example of perspective distortion

4. Simulate a Bayer CFA filter, to reproduce the capturing of the image from a camera sensor. This operation is done by, following the schema from Figure 12, setting, for each pixel, to zero the intensity value of the two color not corresponding with the one on the filter. If a pixel has to be set to green, then it just need to turn off blue and red.

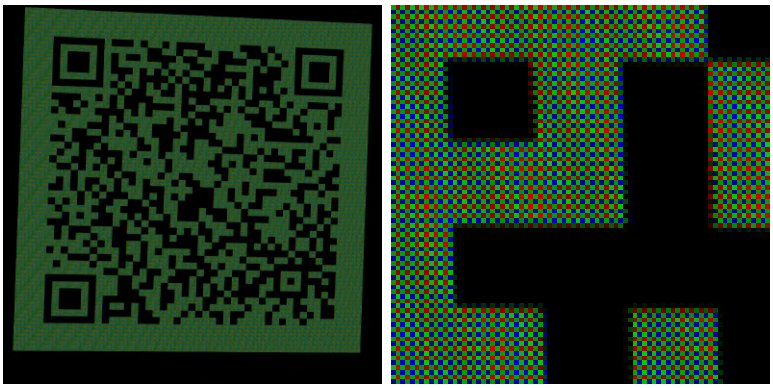

Figure 19: Simulation of the Bayer CFA

5. Apply the demosaicing process by interpolating the intensity values from the Bayer filter. Each pixel, of the resulting image, get its BGR value from a 2×2 sub-matrix of the filter in which the intensity values of blue and red are taken as they are and the green is calculated with the mean of the two green pixels intensity.

Figure 20: Final result

5.2 Implementation of the Perspective Distortion

The implementation takes the four edges of the image and shifts their position. The new edges should not be outside the original image area, in order to do so the program follows this schema, given the edges form Figure 6:

1. $A' = A + (dx, dy)$
2. $B' = B + (-dx, dy)$
3. $C' = C + (dx, -dy)$
4. $D' = D + (-dx, -dy)$

For each edge the value of dx and dy are randomly sampled form a discrete Uniform distribution between an interval from 0 to a 10% of the dimension of the original image. Then the old and new edges are passed to an OpenCV function to calculate the perspective transformation between two planes and finally to another function that computes the transformation.

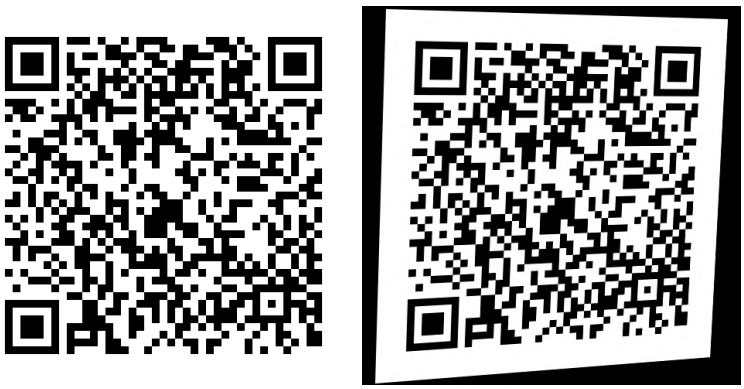

Figure 21: Example of distortion

5.3 Implementation of the Noise

In section 3.3 it has been discussed that the noise, in a image, can be defined as the blur effect, the convolution of the image and a filter, plus the additive noise. OpenCV gives a function to generate a Gaussian blur directly on the given image. The noise has been sampled from a Normal distribution, with zero mean and input standard deviation, and added to the blurred image. At the end the value of the matrix are clipped between 0 and 255.

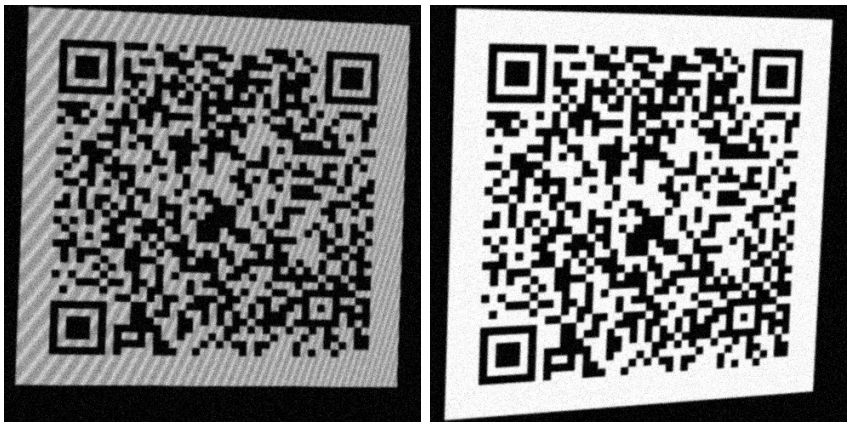

Figure 22: Example of noise with moiré and without

6 Test Result

Tests are performed on two case of studies. The first is centered about error correction, it compares two QR codes with the same data encoded with different level of error correction. The second, compares the EU Digital COVID Certificate versus the optimized version proposed in Arrigo's master thesis[5]. As previously said, each test is divided between the application of the moiré effect or not. Both test have a hierarchical directory structure divided on three level of noise, three level of scaling and twenty-one level of blur all starting from zero. Inside the different directories of blur levels, there are 1000 different homographies of the QR code, that are equals on each different directory. This structure is not based on the order of the application but just how it was thought it was more simple to parse with the java library. Different homographies can be seen as a sort of collection of frame on the action of scanning the QR code. All parameters values have been selected trying to remain on a realistic level of noise on the images. Putting a large amount of noise in an image to make it unreadable would not have been useful for the purposes of this thesis. All the images generated have been passed to the Java library and got in return the result of the scanning, see Section 4.1.1.

The data have been represented in three different chart:

- Line chart: it shows the percentage of successful decoding over different level of blur;
- Stacked area chart: it show result of all the 1000 homograpies over the different level of blur;

- Pie chart: it show the whole set of data to show how a QR code has sustained the test.

6.1 Low Versus High Error Correction Level

Low level error correction permits the detection and recover of a maximum of 7% data, meanwhile the high level reach a maximum of 30%. From this information it is expected that, with the presence of noise on the image, a QR code with higher error correction level should be more easily decoded. But the higher the level and denser and bigger the QR code becomes. To study the effectiveness of the error correction it has been decided to take two QR with the same data encoded inside it, the string of a EU Digital COVID Certificate, and apply the maximum level of error correction to one and the lowest to the other.

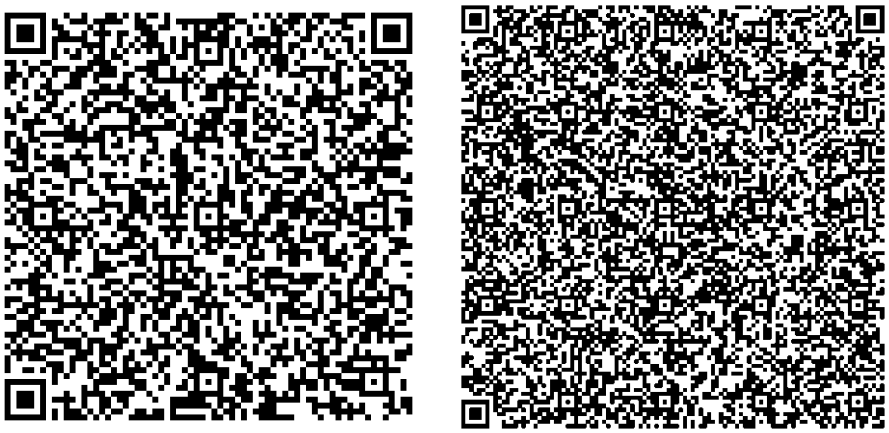

Figure 23: Low and High QR code used for the test

6.1.1 Without Moiré

The test results are presented in a line chart that represent the decoding rate of the QR codes (X-axis) over the different levels of blur (Y-axis). In case of comparison of two different QR codes two lines are putted together in order to see how they perform in a specific context. Knowing that for a QR code to success needs only one successful decoding, the percentage can be interpreted as how fast a specific QR is to be scanned. Taking in consideration the

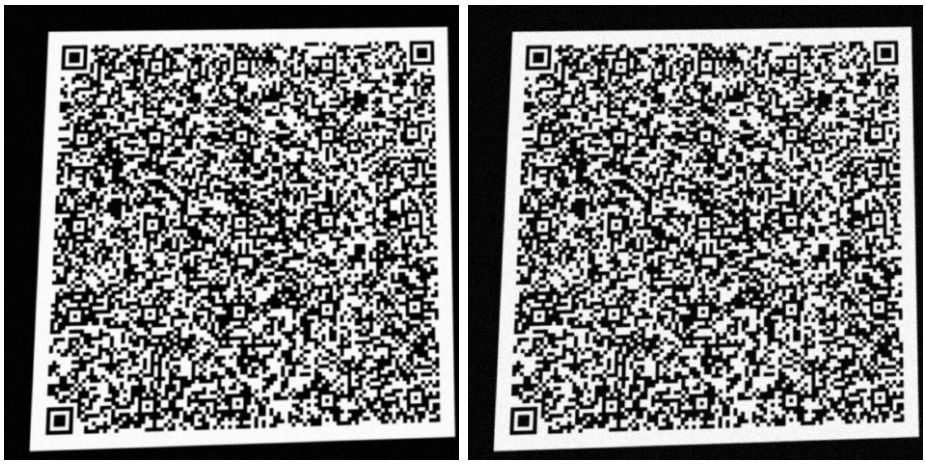

Figure 24: High QR code with medium and high level of noise

decoding rate of the QR codes with no scaling and different levels of noise it is immediately clear that the low level has much better performance than the high level one(Figures 25, 26, 27). Also blur alone seems to not impact much on the good decoding of a QR code, but higher the noise become and more the decoding rate decrease at the increments of the blur.

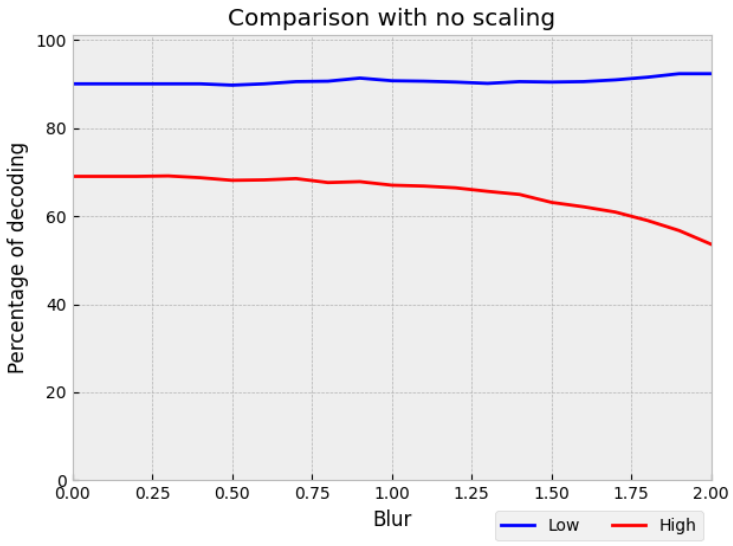

Figure 25: Without noise

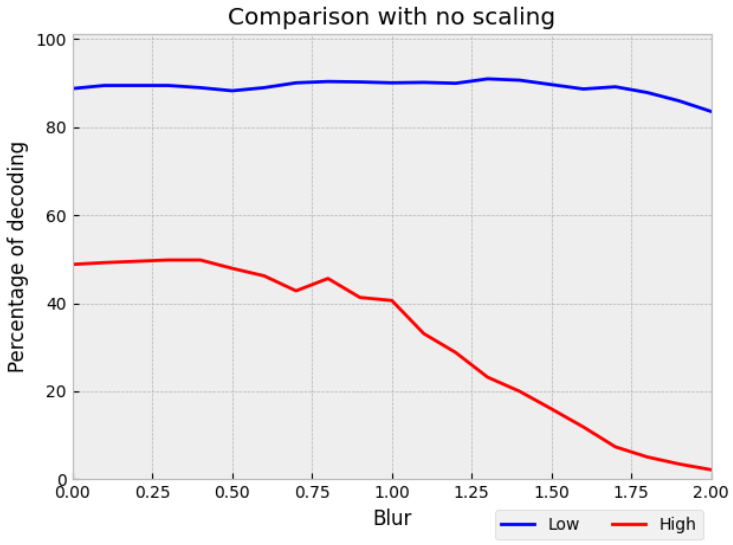

Figure 26: Medium noise

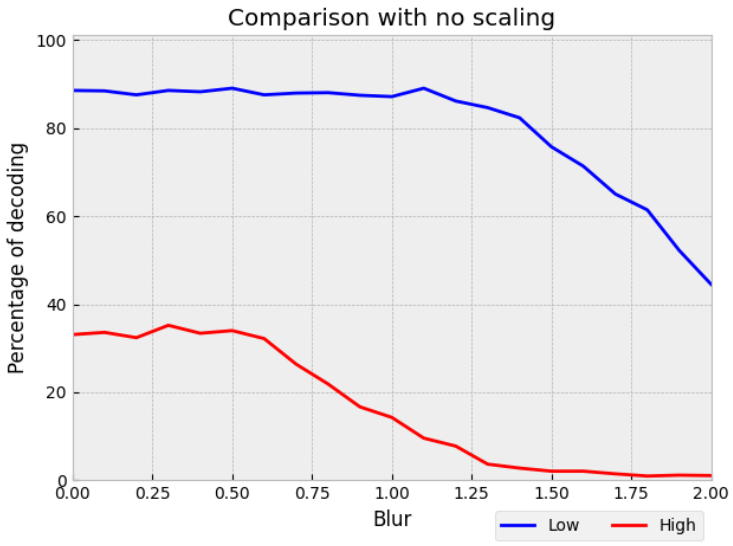

Figure 27: High noise

Looking at the different distances it seems that without noise and blur the library does prefer images with the QR code more distant instead at one covering the whole dimension of the picture. With the addition of noise the result drastically change showing that the low level QR code withstands better those situation. In the line chart representing the three different scaling, of the same QR code, the lines are named *Low_b* for the base image without any scaling, *Low_m* for the medium scaling and *Low_f* for the farthest one.

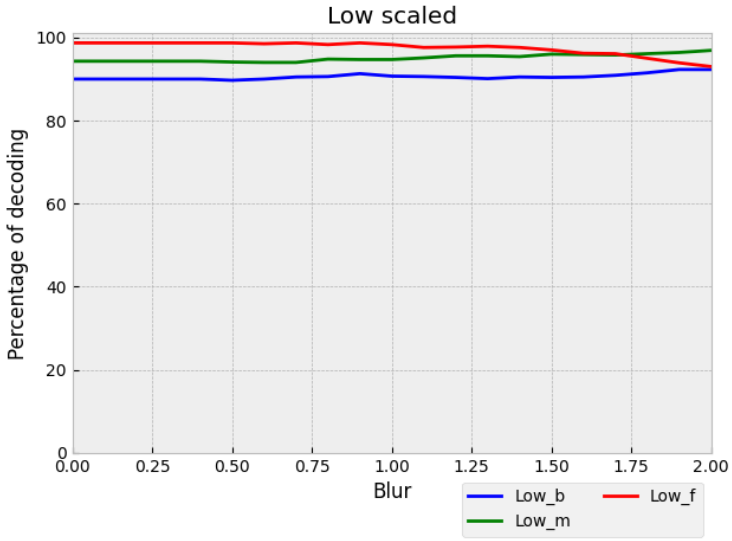

Figure 28: Low - no noise

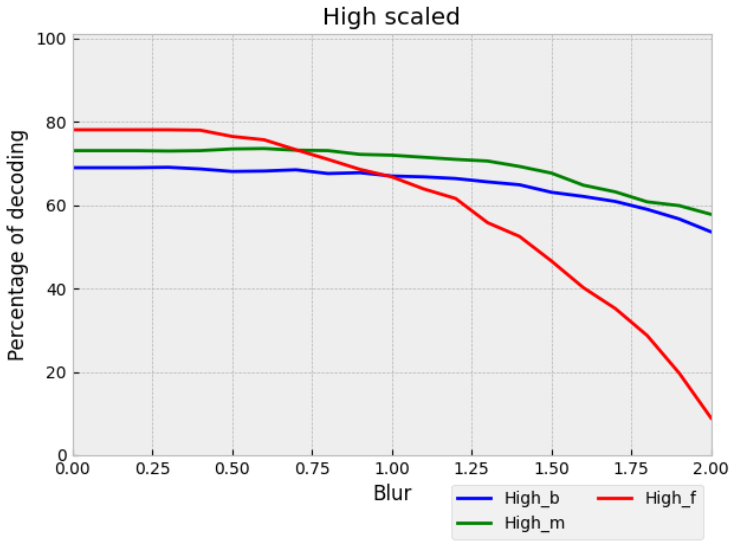

Figure 29: High - no noise

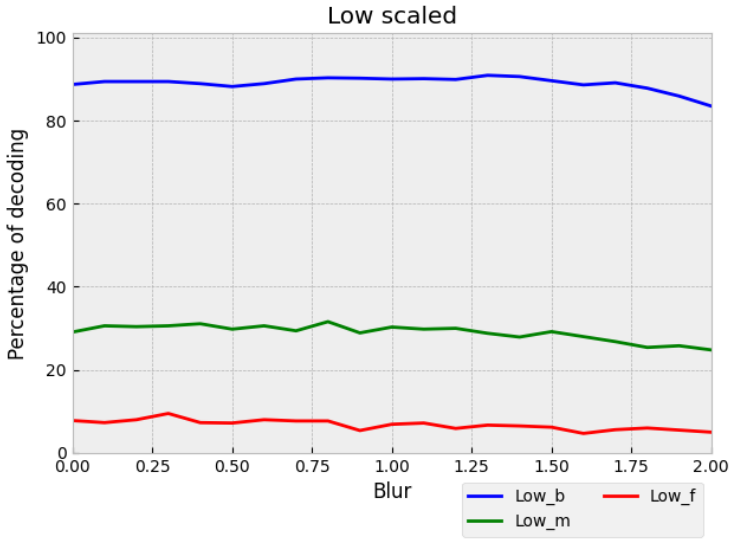

Figure 30: Low - medium noise

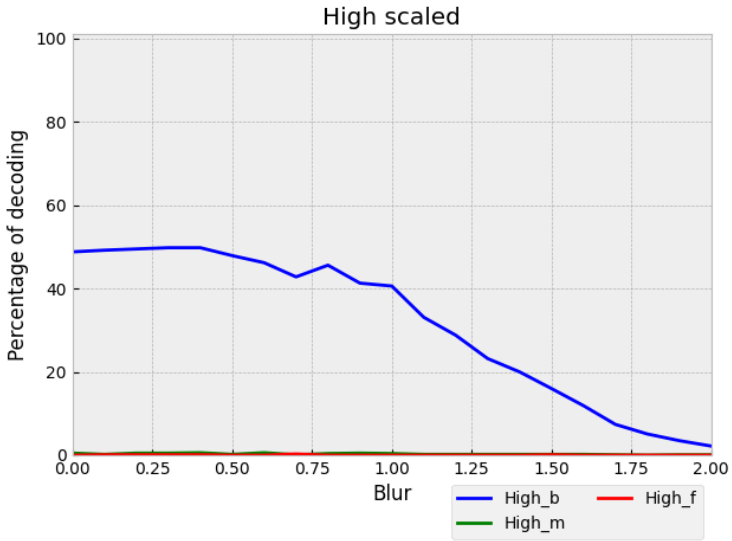

Figure 31: High - medium noise

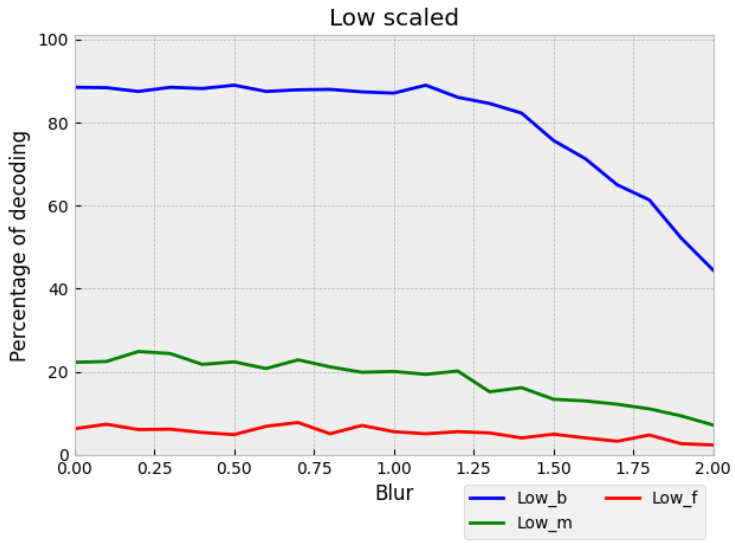

Figure 32: Low - high noise

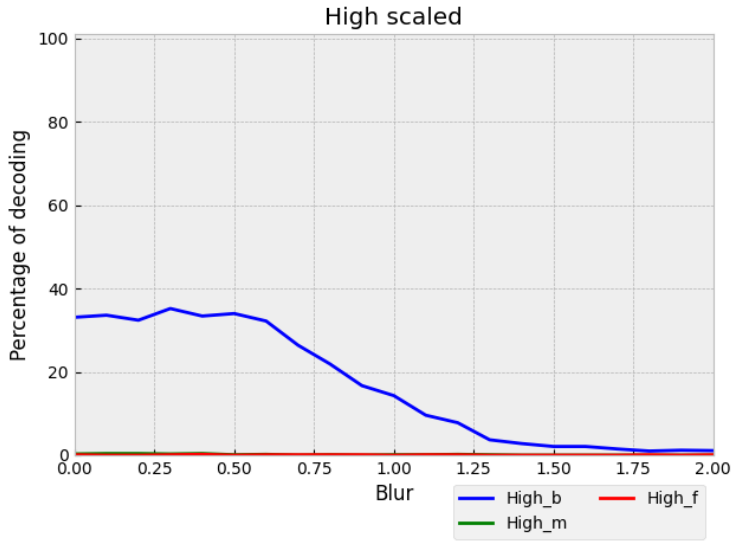

Figure 33: High - high noise

In Figures 34, 35, 36 and 37 is showed a representation of all the results given by a single test, in order to understand why a QR code has been decoded or why it has failed. The presented tests are the ones without noise and with the higher level of noise since the medium is not so interesting. It can be easily seen that, without noise, the low level one is easily decoded without the utilization of the error correction, and the same goes for the higher one with the difference that the second one fails more frequently. In cases of fail the low one does practically always detect the QR code, meanwhile the higher one fails most of the times because it can not be detected. With the increment of the blur level, it is visible that the error correction detects the errors but it cannot correct them because the number of codewords to correct are greater than 30% (Figures 34 and 35).

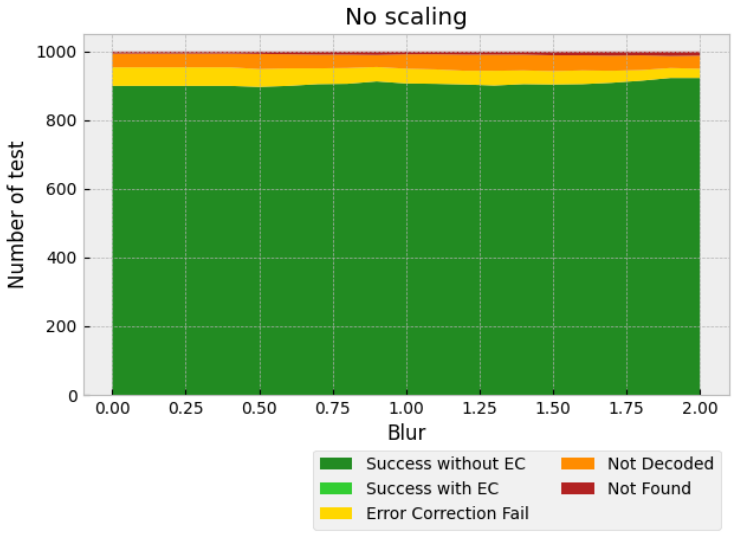

Figure 34: Low no noise

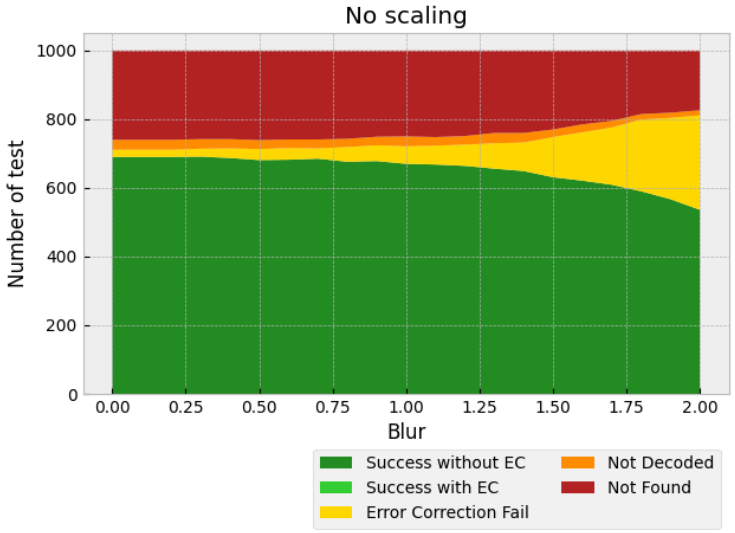

Figure 35: High no noise

With the addition of noise it should be expected that the QR code with high error correction level performs better, instead, as previously seen from the line charts (Figure 33), the decode rate decrease drastically. The reason of this is that the low one is able to recover more errors compared of the high one, due to the lower density of modules. The high one probably has lot of errors equally distributed on the image so, most of the times, the error correction is being applied without success. As the blur increment we see that the low one in no longer able to correct the errors and the high one can not even be decoded, so the modules can not be distinguished one from the others.

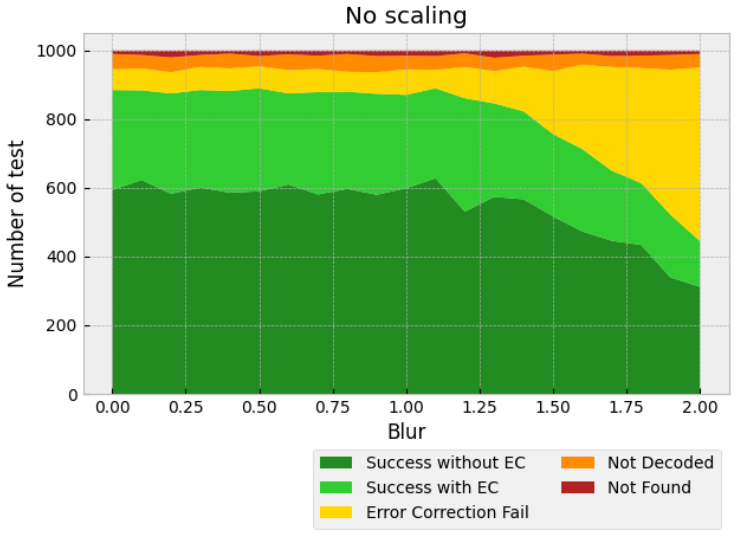

Figure 36: Low with noise

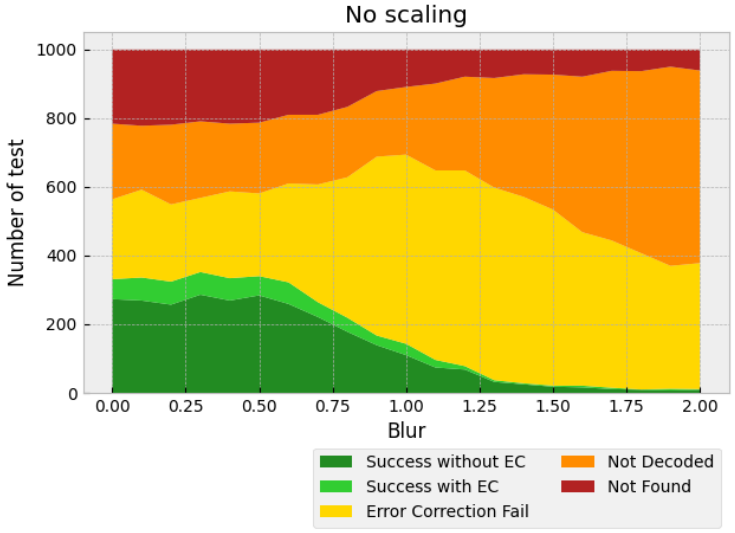

Figure 37: High with noise

To give an easy view of how much the two QR codes have performed in average, all the data has been represented on a pie chart. It is immediately clear that the one with low level error correction performed a lot better than the high one.

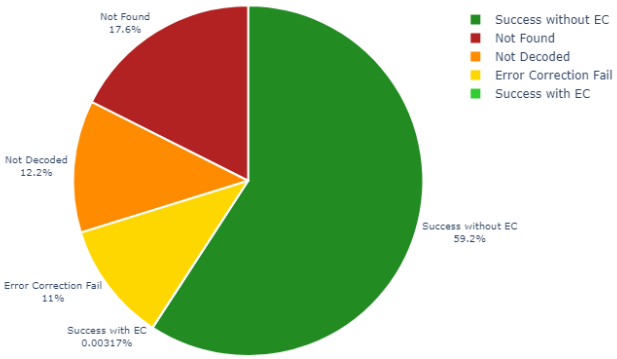

Figure 38: Low

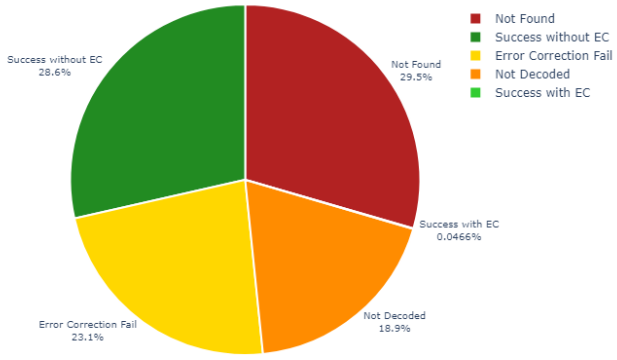

Figure 39: High

6.1.2 With Moiré

In this section will we see the difference on results when the moirè effect is applied in order to simulate the scanning of a QR code on a LCD display. The first notable things it that the overall performances are worse than the simulation without the moirè effect applied. A strange increment, in the line of the low level QR code, is visible on the chart without noise(Figure 40). This happen because as the blur increments the moiré becomes weaker. As per the previous tests, it is immediately clear that the low one seems to have much better performance.

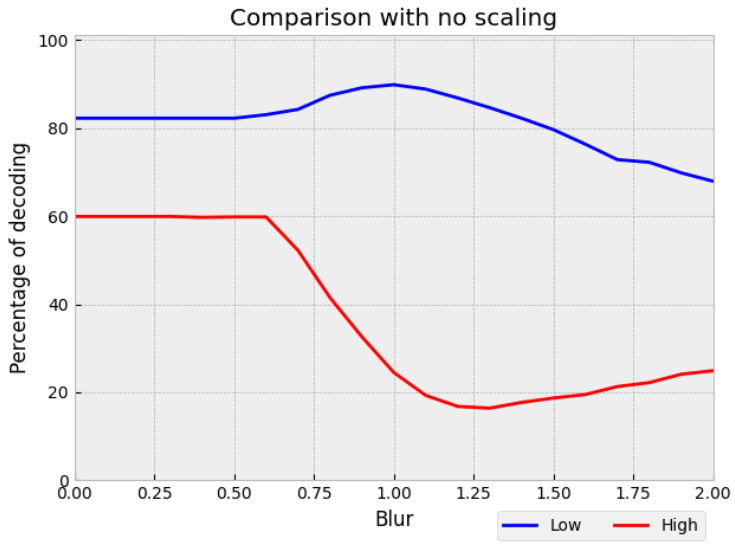

Figure 40: Without noise

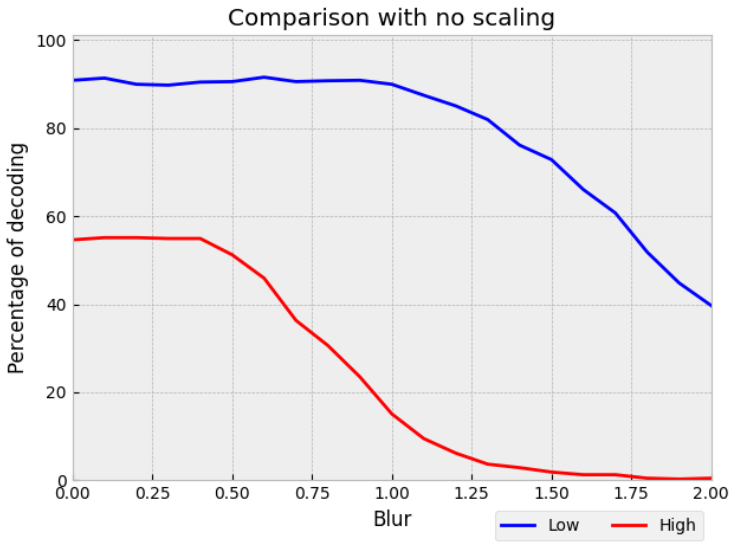

Figure 41: Medium noise

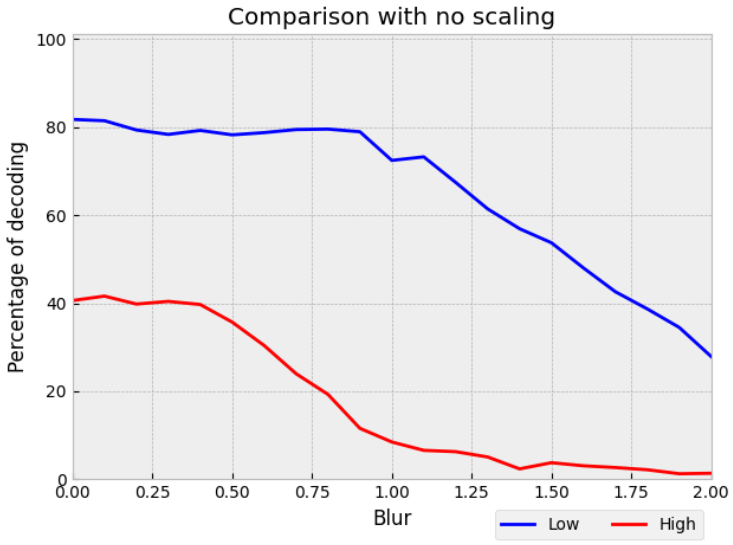

Figure 42: High noise

Looking at what happens with the different distances, the pattern remains the same as before with the QR code with low error correction level withstanding better the different tests. To show this only some charts are presented as an example, and not all the combination. The chart takes in consideration the medium value of noise. It is visible, as said before, that the low QR code can sustain better than the high one. Talking about the distances, both does suffer a lot compared to the test without moiré (Figures 32 and 33). However the high one is practically unusable on higher distance and with higher noise the situation becomes even worse.

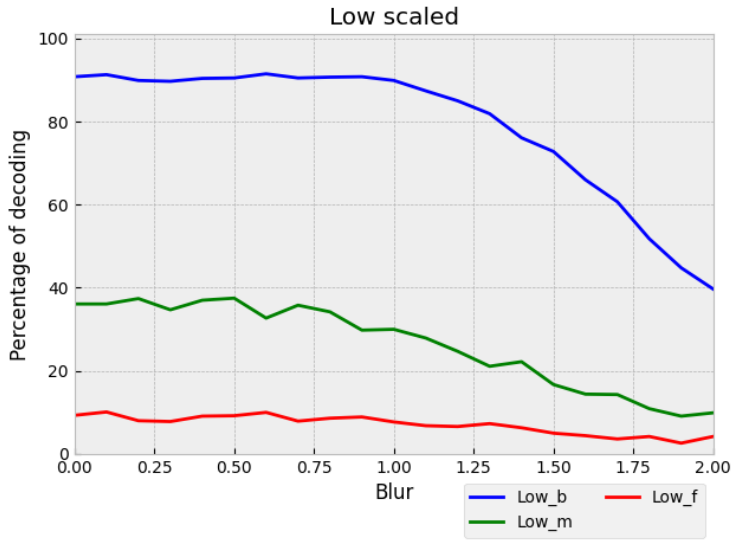

Figure 43: Low scaling

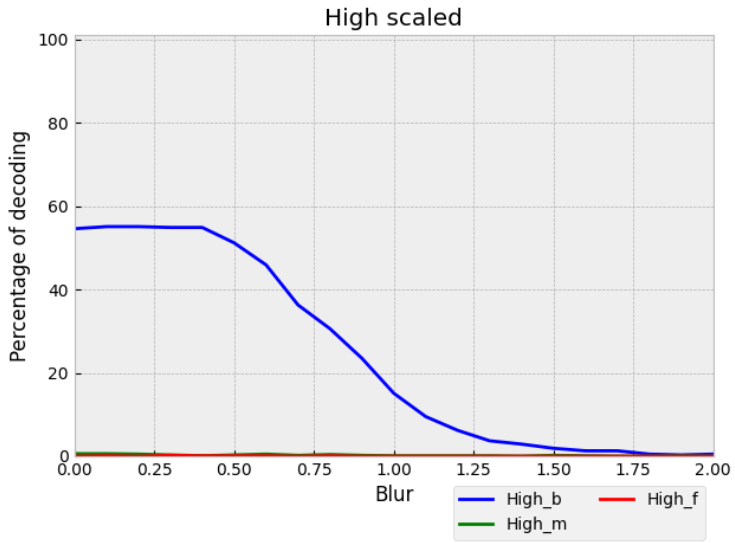

Figure 44: High scaling

The result of the test without moiré has showed that the error correction was almost never successfully applied. For what regards this test instead, the error correction is very useful for successfully scan the QR code. Contrarily at what could be expected, the QR code that is able to recover the majority of errors in order to be readable is the lower one. This is caused by the lower density of modules. In Figure 45 is notable that, without noise, as the blur effect increments the moiré becomes less problematic and the decoding does not need the error correction as much as before. With the addition of the noise the success depends entirely on the error correction application.

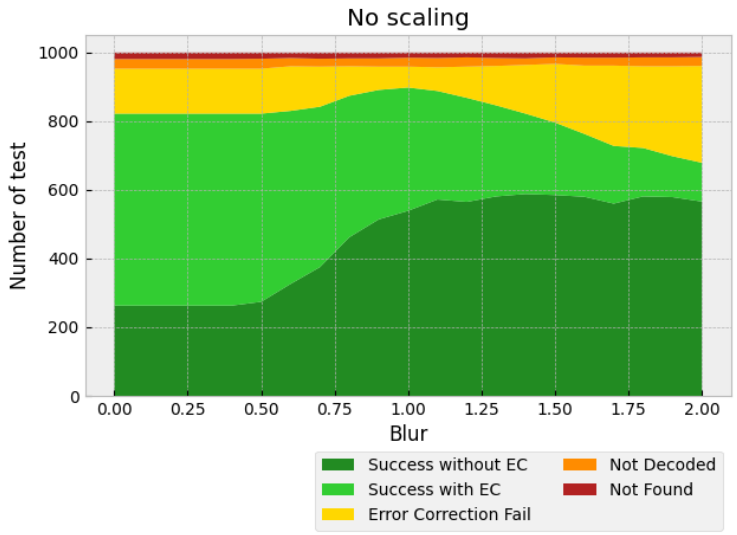

Figure 45: Low - no noise

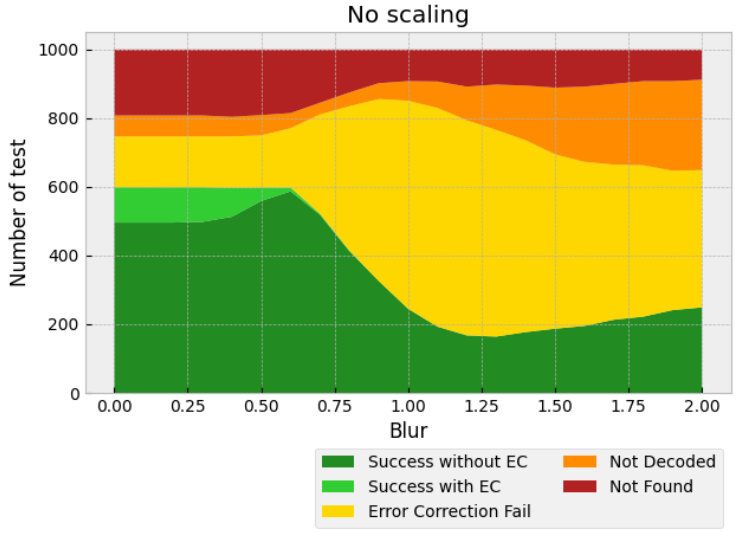

Figure 46: High - no noise

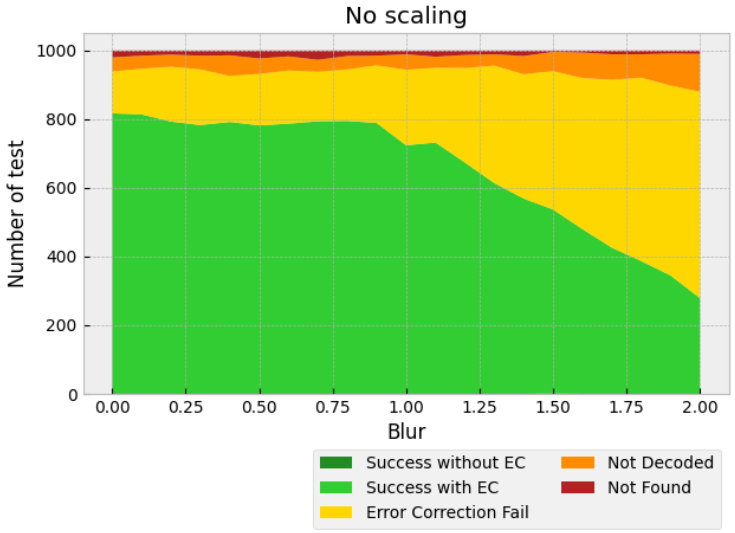

Figure 47: Low - high noise

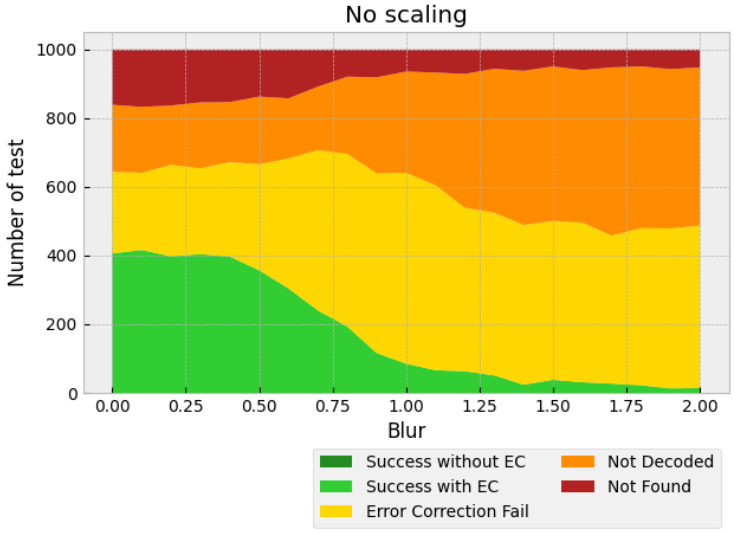

Figure 48: High - high noise

The moiré effect seems to destroy the usability of the QR code with high level of error correction. The pie charts (Figures 49, 50) show that in the entirety of the tests the low level QR code has performed much better than the high one. In this case, the difference between the two is much greater then the one observed on the test without moiré. We can conclude that the trade off of having a grater level of error correction in not worth the loss of usability of the QR code.

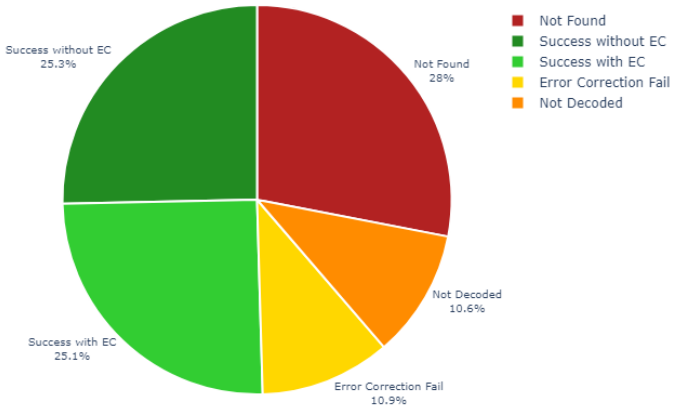

Figure 49: Low

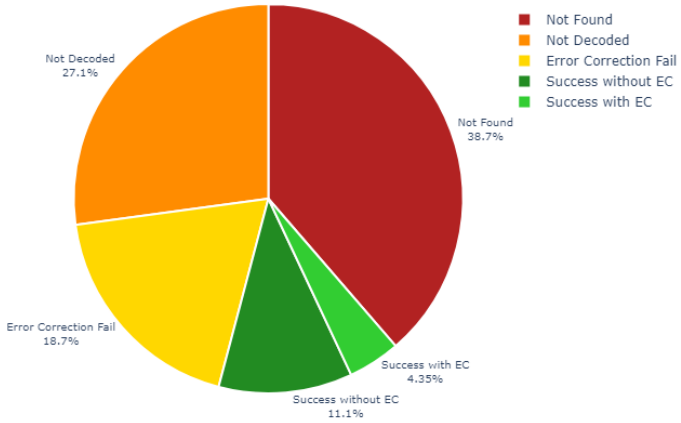

Figure 50: High

6.2 Optimized DCC Versus Base DCC

This case of study start form the implementation of an optimized version of the EU Digital COVID Certificate discussed in Arrigo's master thesis[5]. This version of QR code contains the same essential data as the original DCC. The main differences are the data structure in which the data are stored and some useless data, not used on the validation of the certificate, have been removed. The last main difference is the utilization of the low level of error correction instead of the quartile level used by the official DCC. The type of tests performed on each are the same as the ones performed on the comparison of low and high level of error correction. It is expected that the optimized one will be much better than the base one, but it is interesting to see how well it sustain the different type of disturbances.

Figure 51: Optimized and base DCC used for the test

6.2.1 Without Moiré

It is immediately clear that the difference between the two version is overwhelming. Taking in consideration only the noise level, the optimized version in able to sustain perfectly this level of noise. Adding the distances into consideration, it starts to suffer a bit but it does remain on about 70% of decoding rate even on the worst case (Figure 54). Meanwhile the base DCC is sightly better then the one with the high level of error correction presented before (Figure 54 and 33).

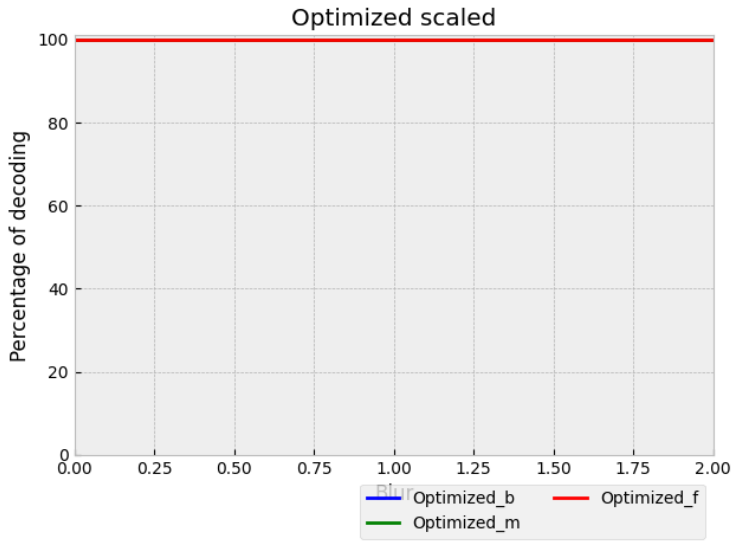

Figure 52: Optimized - no noise

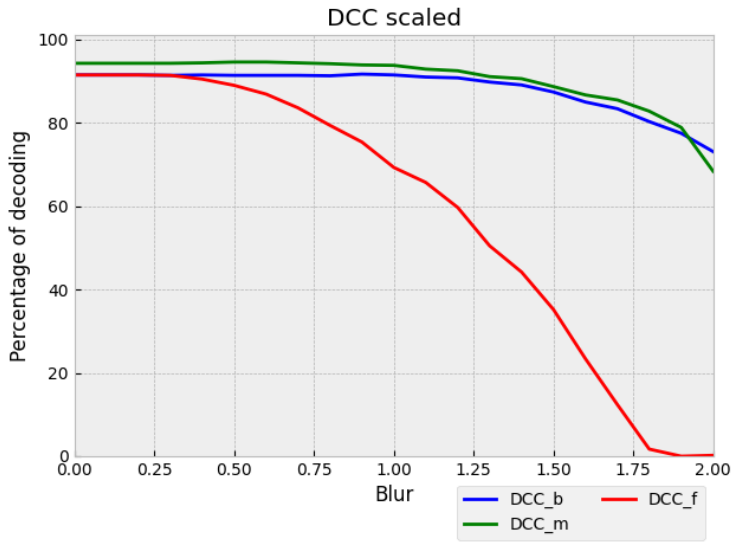

Figure 53: DCC - no noise

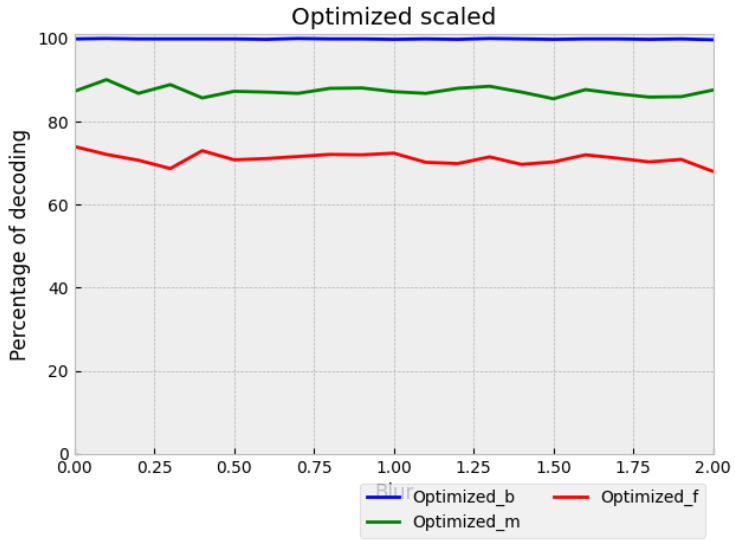

Figure 54: Optimized - high noise

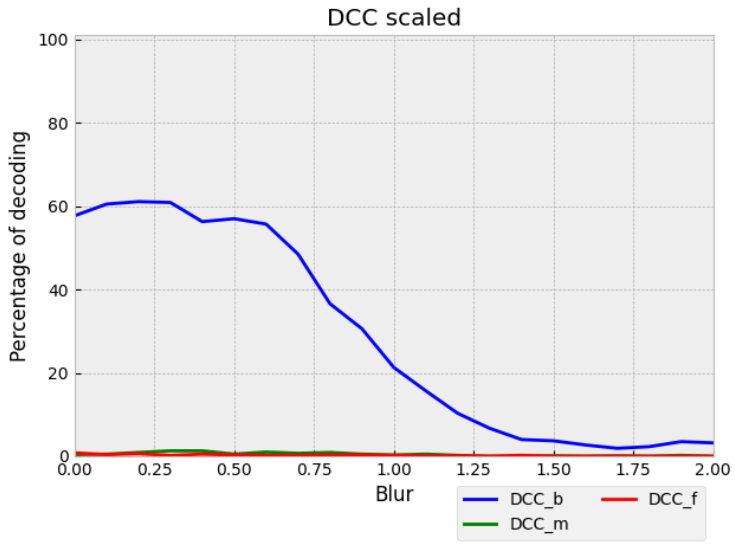

Figure 55: DCC - high noise

Like in the test with low and high, the presence of noise alone does not activate much the error correction. The optimized DCC has been decoded without the needs of correct any error for the cases with no noise and medium noise. Only with a higher level of noise the error correction has been applied in order to recover some data. For what it regards the base DCC it is possible to say that the noise gives to much error to correct and in the other cases it just made impossible the detection or decoding of the QR code.

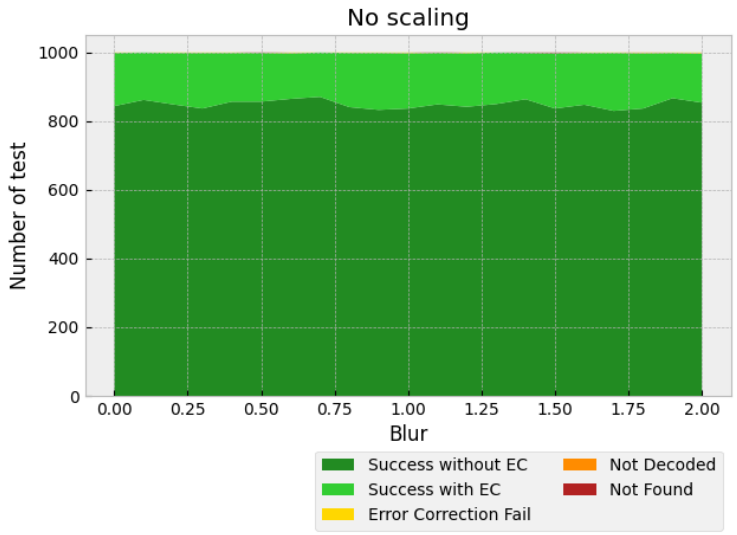

Figure 56: Optimized - high noise

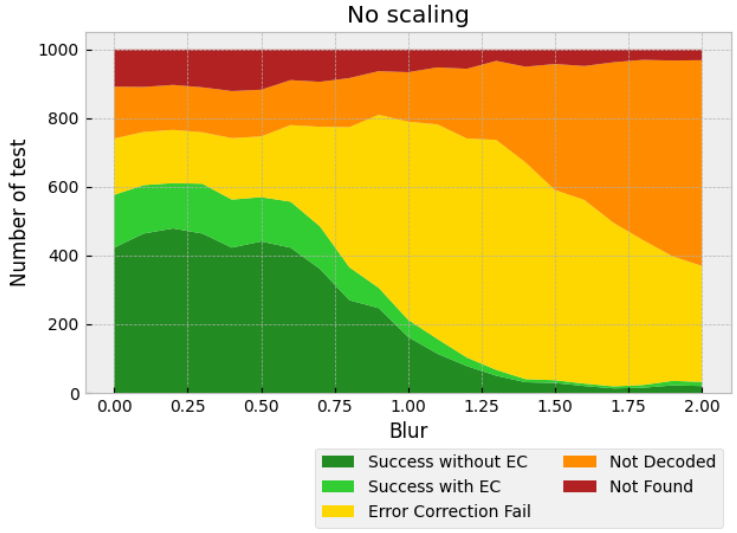

Figure 57: DCC - high noise

In the end, it is possible to claim that, with only noise, the optimized version is practically always readable. Meanwhile the base DCC has a total decoding rate lower than the 40%. So more then half of the time the scanning of the QR code will result in a failed scan.

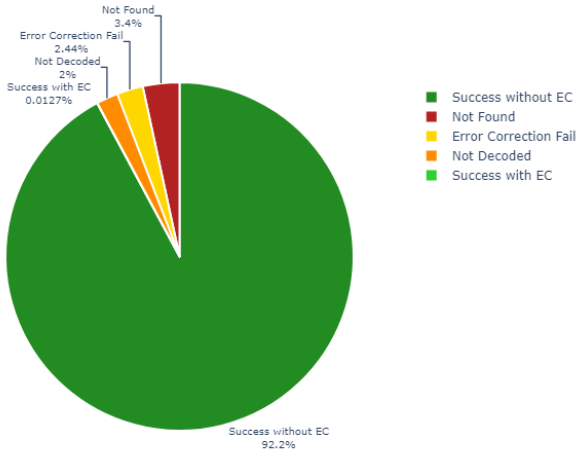

Figure 58: Optimized

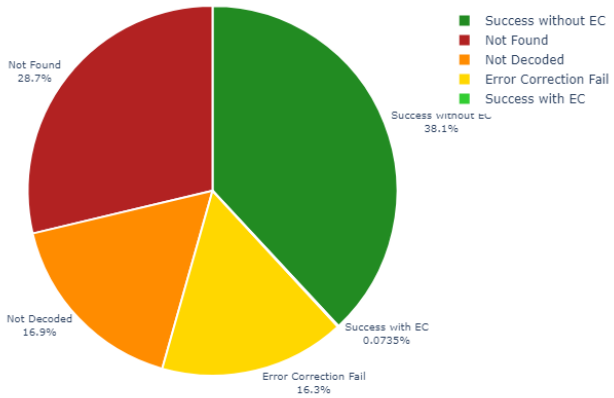

Figure 59: DCC

6.2.2 With Moiré

By applying the moiré effect on the images it is expected to see a drop in decoding rate of the two QR codes and an increment of application of error correction. Looking at the chart is possible to see that without noise the line of the test with no scaling has a lower decode rate at the start, compared to the other two. This is due to the fact that as the blur increment the moiré effect gets weaker and the same goes with higher distances.

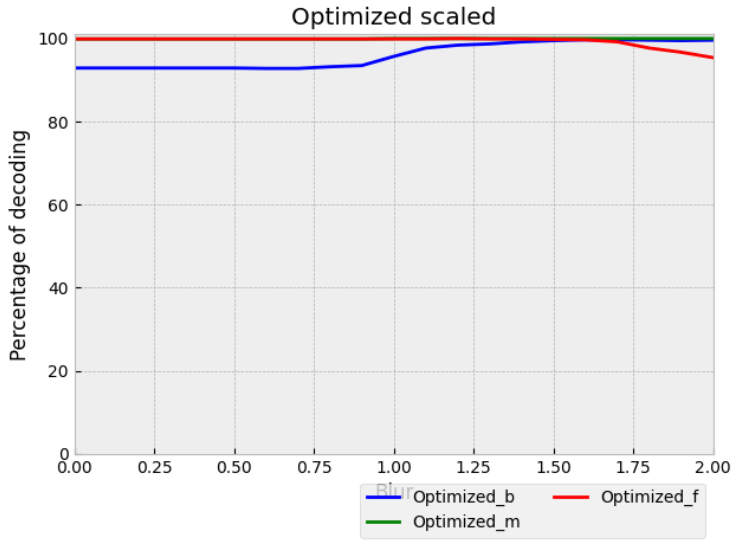

Figure 60: Optimized - no noise

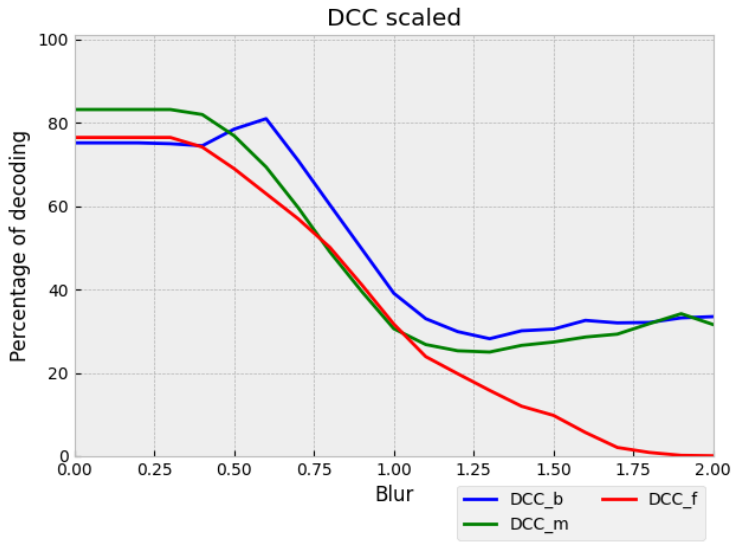

Figure 61: DCC - no noise

With the addition of the noise it is expected that the optimized version will withstand much better than the base DCC. But what is interesting to see is how much the error correction is involved. From the chart it is visible that, without the noise, the defocussing of the camera should be able to remove a lot of disturbance of the moiré. The error correction is applied only on the homograpies with low blur, meanwhile as the blur increases the QR codes have been decoded without its application. If instead the noise level increments, the error correction is essential to have successful scans. The only difference is that with a medium level of noise the blur is able to mitigate a bit the moiré effect, meanwhile with the higher value the QR codes are scanned only thanks to the error correction.

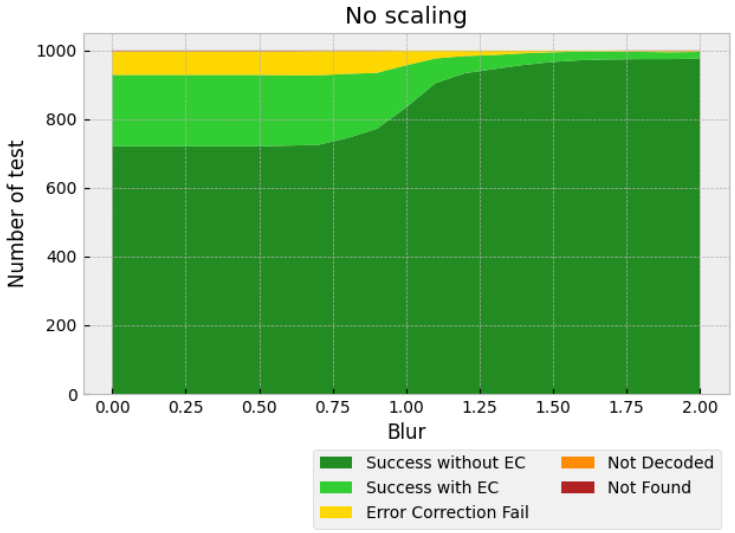

Figure 62: Optimized - no noise

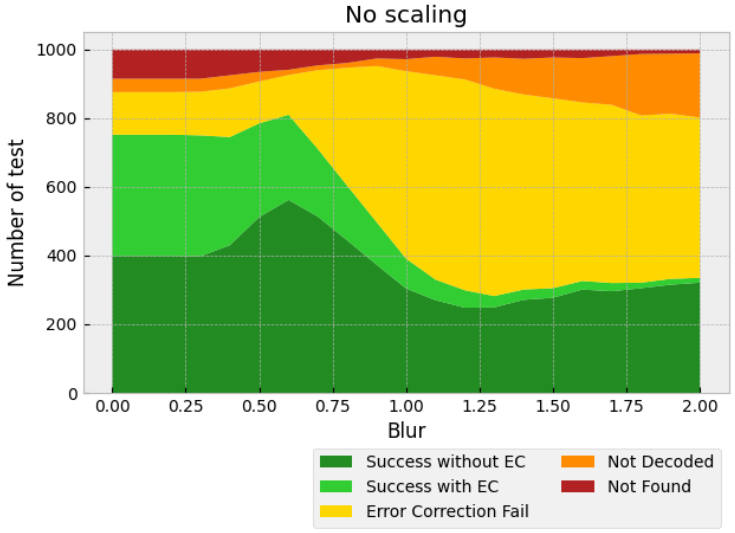

Figure 63: DCC - no noise

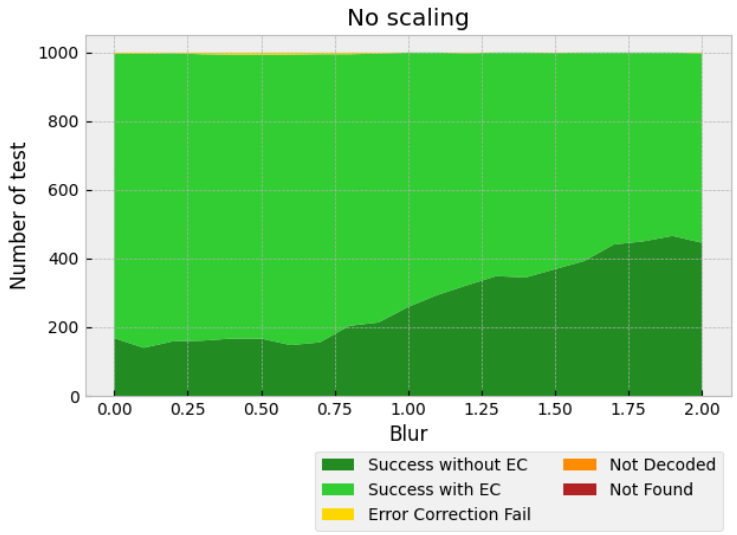

Figure 64: Optimized - medium noise

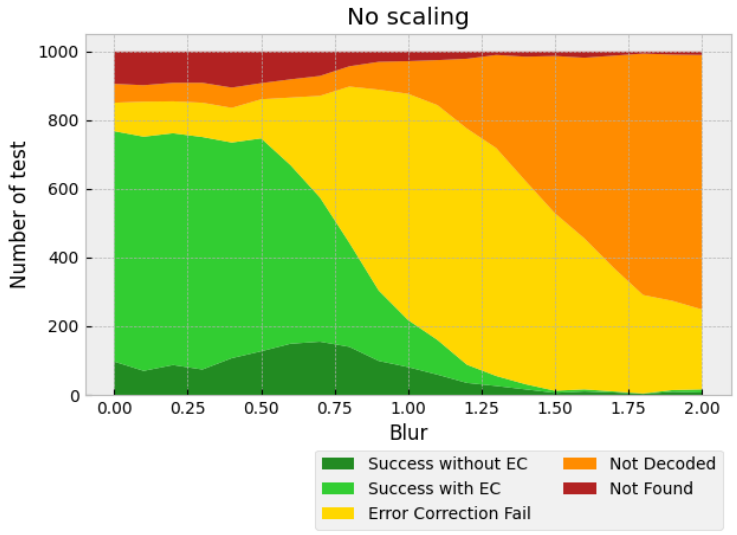

Figure 65: DCC - medium noise

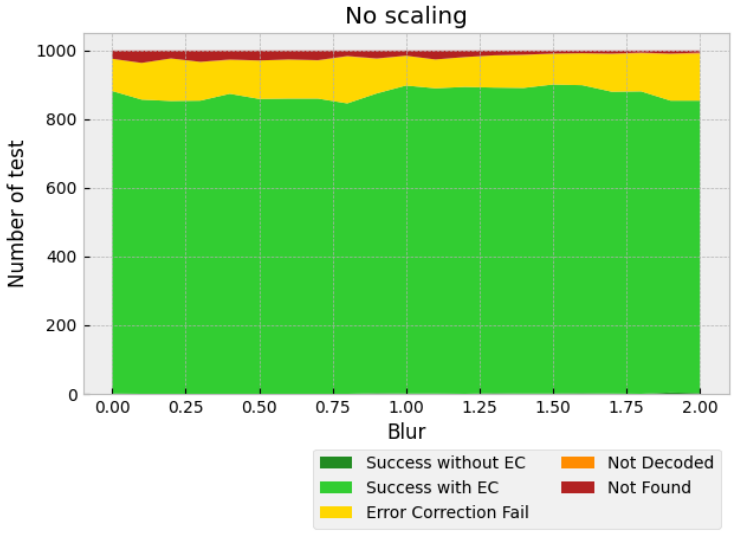

Figure 66: Optimized - high noise

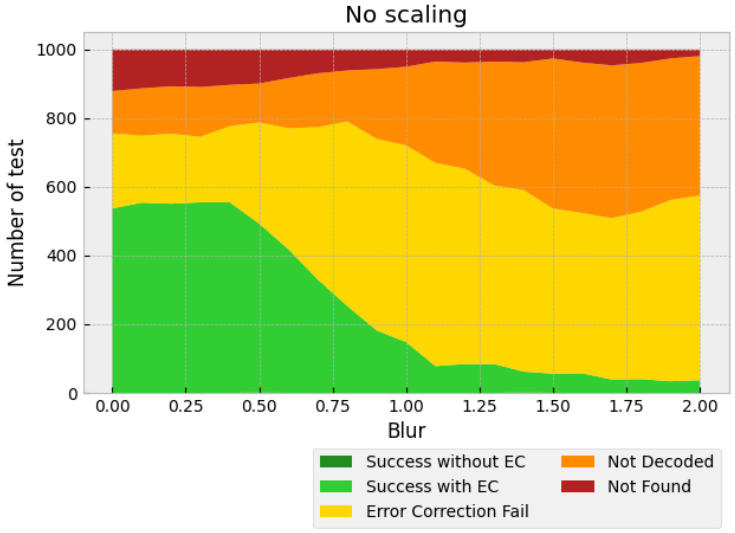

Figure 67: DCC - high noise

Even in this case, is clear that using a better data structure, in order to have a simpler QR code, is necessary for a good scan result. A lower version QR code is able to sustain image distortion a lot better than a more complex one. To give a better understanding of how much better the optimized version has performed the data have been collected together in two pie charts. From this chart is immediately visible the major success rate of the optimized DCC, this is even more greater thanks to the error correction reaching over 90% of success.

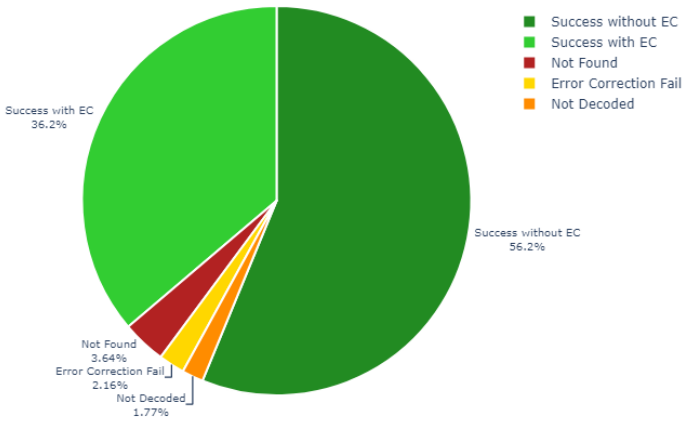

Figure 68: Optimized DCC

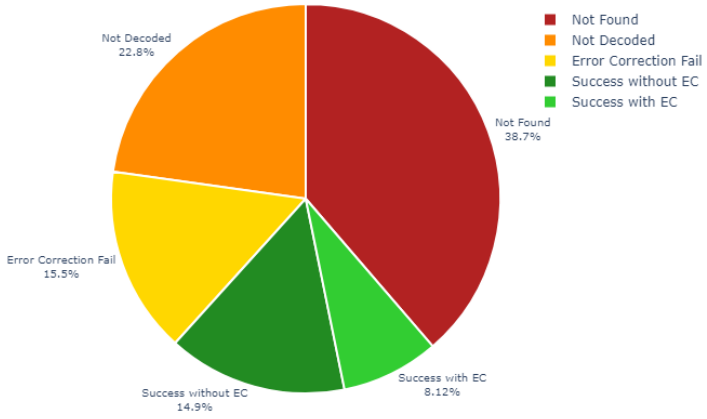

Figure 69: Base DCC

7 Conclusion

In this thesis it has been discussed about the detection and decoding of QR codes. The main discussion is been divided in two type of disturbances that could influence the good result of a scan. First, we simulated a situation in which a camera has to detect a QR code form a surface that is not a display. So the images have been modified to create different levels of camera defocussing and different levels of thermal noise. This is to create a realistic scenario in which the camera try to put on focus the QR code or the object is not much stable, added to the possibility to be in an environment with low or high illumination. To test how a library deals with those problems, we selected two different case of study. In both cases the main point of interest was how a successful scan is dependent of the error correction. On the first case, we used two QRs with the same data encoded inside but with the minimum level selected for one and the maximum for the other. In the second case, we compared the EU Digital COVID Certificate and an experimental optimized version, in order to understand how much is important to use a more compact and efficient data structure. From this two cases it has been found that a lower level of error correction, and even more a good data structure, are essential to have efficient QR codes. Contrary on the expectation an higher level of error correction brings down the decoding rate by a good margin. This is due to the fact that the QR becomes much larger and denser with the higher level, and bigger is the data encoded inside it and more the error correction codewords becomes bigger too. Furthermore, it has been shown that most of the times the error correction was not applied when a QR code was decoded. Instead, it happened frequently that a QR

code was not decoded, because the error correction detected the errors but could not recover the data due to the large amount of them. This was much notable on both the QR code with high level of error correction and the base DCC. The reason is that, by increasing the error correction, the density of the QR becomes much higher. This density makes the noise more impacting causing lots of more errors. Another problem caused by the noise was that the modules became not distinguishable between one from another, making the decoding impossible. A large number of homograpies of the two higher version was not even detected. This happens because the position marker, the alignments marker or timing patterns could not be found due to the noise and the perspective distortion.

The second simulation was the scanning of a QR code on a LCD display. To create this situation it has been implemented a simulation of the moiré pattern to be added to the images. Together with the noise and perspective distortion. Taking in consideration the same two case of studies from before it was clearly visible that the moiré has a huge impact on the performance of the larger QR codes. An interesting result is that, in this case, the error correction is essential to have a successful decoding. On the QR with low level of error correction, and even more in the optimized version of the DDC, the application of the error correction made possible to have a success rate close to the one form the previous simulation.

From all of this interesting result is possible to say that the trade off of having an higher error correction is not worth the lost of usability of the QR code. An high level of error correction is suggested in industrial environment where the QR could be seriously damaged. For an every day utilisation is

better to use the lower version in order to have a better decode rate. From the results of the first test someone could be easily be fouled to think that the error correction is almost useless but this is not the case. The moiré pattern simulation showed that the error correction is essential in some cases. Another interesting result is that some level of blur can help to eliminate some of those artifacts and made the detection more easy. As it has been said when thinking of using the QR codes is important to use a low level of error correction and implement a good data structure to store the data in a safe way but trying to consume the lower number of bytes as possible.

7.1 Future Works

Starting form this study there are multiple new works that can be done. One work could be to study the data masks of the QR code and try to prove if the mask chosen from the lower penalty score is the best one even in practice. This could be done by trying to apply by force all the difference masks to the QR, one at time, and test them like it has been presented in this thesis. To do so it should be needed to take an existing library and change a lot the encoding and decoding of the QR code. This is because the library during the decoding process must know how the values are encoded. The better choice is probably to develop a library anew. By making a new one it could be interesting to try to develop a new mask specific for certain types of data that already have a low entropy, like a digital signature. For whats it regards the detection and decoding it could be interesting to try to apply some preprocessing on the image before it get passed on the library. This could help the decoding of QR codes by eliminating disturbance on the

image. A further extension of this study could be a simulation of different camera resolution and the simulations of different surfaces in which the QR code is placed.

References

[1] Data mask patterns. https://en.wikiversity.org/wiki/File:QR_Code_Mask_Patterns.svg.

[2] Opencv. https://opencv.org/about/.

[3] Qr code codeword. https://en.wikiversity.org/wiki/File:QR_Code_Unmasked.svg.

[4] Qr code structure. https://en.wikipedia.org/wiki/File:QR_Code_Structure_Example_3.svg.

[5] Giacomo Arrigo. Analysis of the eu digital covid certificate system and proposal of design improvements. Master's thesis, Ca'Foscari University of Venice, 2021.

[6] Marco Carfizzi. Proposal of improvements for the digital covid-19 certificate. Master's thesis, Ca'Foscari University of Venice, 2021.

[7] Ministero della Salute. EU digital covid certificate verifier app - android. https://github.com/ministero-salute/it-dgc-verificaC19-android, 2022.

[8] Jan Flusser, Sajad Farokhi, Cyril Höschl, Tomáš Suk, Barbara Zitová, and Matteo Pedone. Recognition of images degraded by gaussian blur. *IEEE Transactions on Image Processing*, 25(2):790–806, 2016.

[9] Dirk Huisman, Glenn Heeres, Bertil Van Os, Willem Derickx, and J.M. Schoorl. Erosion and errors: Testing the use of repeated LIDAR analyses

and erosion modelling for the assessment and prediction of erosion of archaeological sites? *Conservation and Management of Archaeological Sites*, 18:205–216, 07 2016.

[10] ISO. ISO/IEC 18004:2015. `https://www.iso.org/standard/62021.html`.

[11] Lacan Jerome, Roca Vincent, Peltotalo Jani, and Peltotalo Sami. "reed-solomon forward error correction (FEC) schemes. 04 2009.

[12] Leila Kabbai, Anissa Sghaier, Ali Douik, and Mohsen Machhout. FPGA implementation of filtered image using 2D gaussian filter. *International Journal of Advanced Computer Science and Applications*, 7, 07 2016.

[13] Bolin Liu, Xiao Shu, and Xiaolin Wu. Demoiréing of camera-captured screen images using deep convolutional neural network. *ARXIV*, 2018.

[14] ZXing Project. Zxing. `https://github.com/zxing/zxing`, 10 2022.

[15] Kesten Victor. Evaluating different spatial anti aliasing techniques. Master's thesis, KTH, School of Computer Science and Communication (CSC), 2017.

[16] Rafael C. Gonzalez; Richard E. Woods. *Digital Image Processing*. Pearson, 2018.