TNM034 2014

An Automated Pipeline for Extracting and Decoding QR-codes from Arbitrary Images

John Hollén
johho982@student.liu.se

Simon Bergström
simbe109@student.liu.se

December 11, 2014

Abstract

This is an abstract.

Contents

1 Introduction **1**

2 Theory **1**

3 Method **2**
3.1 Binarizing the Image 2
3.2 Localizing the Fiducial Marks 3
3.3 Rotation 6
3.4 Adjustment for Perspective Distortion 6
3.4.1 Corner Detection 7
3.4.2 Projective Transformation 9
3.5 Cropping 10
3.6 Decoding the QR-code 11

4 Result **11**

5 Conclusion and Discussion **12**
5.1 Performance 12
5.2 Corner detection 13
5.3 Transformations 13
5.4 Special cases 13

1 Introduction

This report describes the implementation of an automated pipeline for extracting a QR code out of an image and decoding it to string of text. Problems with decoding QR codes such as spatial and photometric distortion are considered in this report and the solutions in the implementation are discussed together with possible improvements that could be done. The project is restricted to only consider QR codes of version 6. The implementation has been done in Matlab.

2 Theory

QR-code is short for quick response code and is a two dimensional barcode that consist of binary information. Computers can extract this information and make it understandable for humans, or the information could be used to redirect the machine to for example visit a certain webpage or make smart phones open a specific application.

The QR-code version 6 is 41x41 pixels in size. It contains three fiducial marks that are used to localize it and an alignment pattern that is used to determine the orientation of the QR-code. The distance and size specifications are visible in Fig.1. Around the fiducial marks there are quiet zones, at least one bit in size where no information is present.

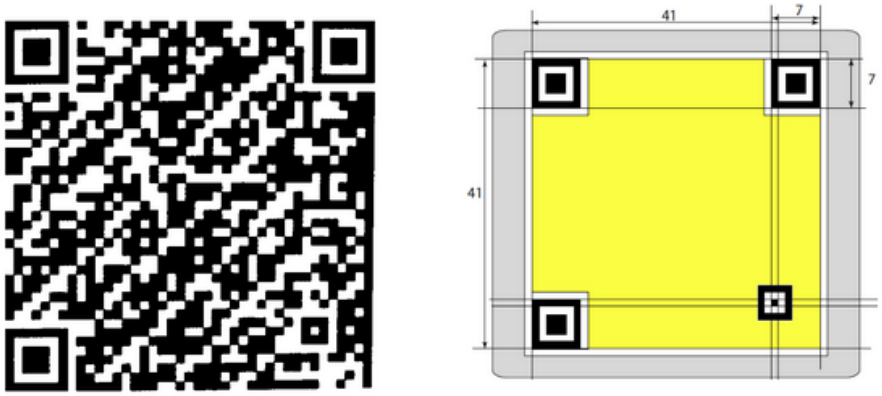

Figure 1: *To the left: an example of a QR-code version 6. To the right is the dimensions of the QR-code.*

3 Method

The implementation is made without any interaction during the execution. The input is an image, containing a QR-code and the output is a string of text. The method for finding and extracting QR-codes is divided into several steps. Each step is described in its own subsection. Something worth mentioning is that operations that require interpolation, such as resizing, rotating and perspective projection are all made on a grayscale version of the original image. All steps in the pipeline are illustrated in Fig.2.

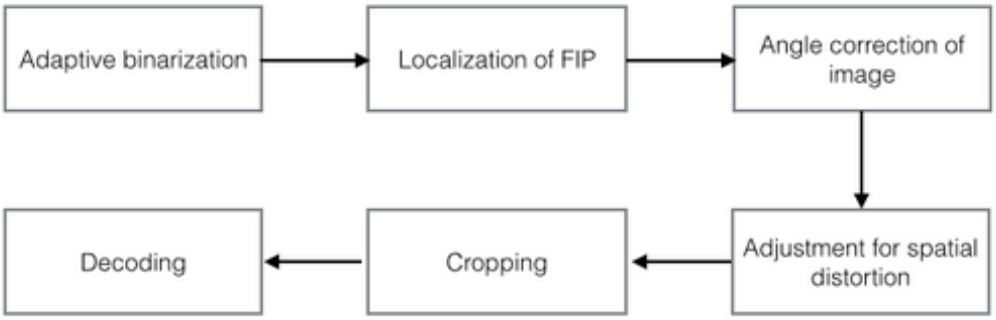

Figure 2: *All the steps in the pipeline for extracting QR-codes.*

3.1 Binarizing the Image

In order to be able to manage images with a wide variation in brightness and contrast, an adaptive binarization function has been implemented. After the image has been converted to only contain grey scale values, the first step in the adaptive binarization function is to calculate an integral image from the original image.

An integral image is also known as a summed area table [6]. It is a data structure where the sum of the pixel values in the original image is stored in the corresponding cell in the integral image. How this is done is illustrated in Fig.3.

4	1	2	2	4	5	7	9
0	4	1	3	4	9	12	17
3	1	0	4	7	13	16	25
2	1	3	2	9	16	22	33

Figure 3: *An intensity image to the left and its calculated integral image to the right.*

When the integral image has been calculated a window of pixels is created. The size of this window is 1/8 of the width of the image. This window is then iterated over the image. During this iteration the average of the pixel values in the integral image that are inside the window is calculated. If the current pixel is a certain percentage lower than the average of the window, the pixel in the original image is set to black. Otherwise the pixel is set to white. The percentage is set to 15% according to [6].

This method is only used in order to find the fiducial marks in the QR-code properly. It is not used in the last step when the decoding of the QR-code is done, because when binarizing the QR-code in the last step, a global binarization method provides a better result.

3.2 Localizing the Fiducial Marks

The so called fiducial marks are three shapes located in three of the corners of the QR-code. The fiducial marks have the shape of a square containing black and white pixels. The black and white segments of the square have the ratio 1:1:3:1:1 [5] in the order black, white, black, white and finally black. By scanning the binarized version of the original image both horizontally and vertically segments of black and white pixels can be extracted. For each segment, the size of the segment and the pixel position where the segment ends are stored in a separate list.

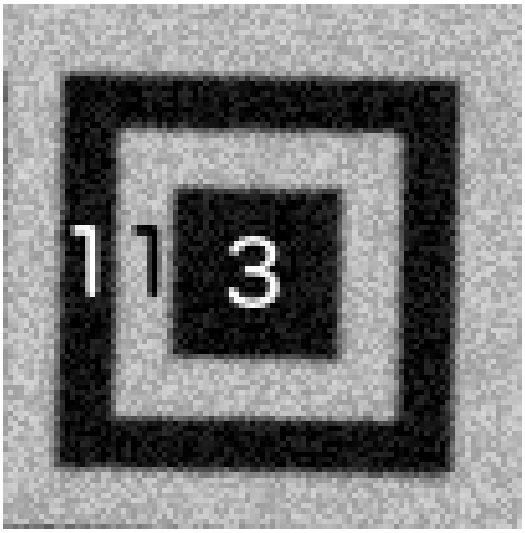

Figure 4: *Image of a fiducial mark showing the ratios between black and white.*

When all the segments in the image have been extracted, the list of segments is searched. First a black segment is looked up. Then the segments upwards, downwards, to the right and to the left are checked in order to determine if the ratio is fulfilled. When a match has been found, the corresponding pixel positions in an otherwise black image with the same size as the original image are coloured white. See Fig.5. The result is a black image with small white fields where the algorithm has found a match. This enables labelling of the white fields in a later stage.

If a match has been found or not is determined by the following code snippet. The snippet shows how it is done when scanning vertically.

```
findpattern = zeros(height, width);
percentage = 0.32;
%Check middle to adjacent
if abs(middleBlack-3*upWhite) <= percentage*middleBlack &&
    abs(middleBlack-3*downWhite) <= percentage*middleBlack
   %Check the outer segments
   if abs(upWhite-upBlack) < percentage*upWhite &&
       abs(downWhite-downBlack) < percentage*downWhite
      %Match is found, color the pixels in the black image white.
      %SegmentsY contains the pixel positions for the segments end.
      findpattern(segmentsY(i-2, 2):segmentsY(i+2, 2),
          segmentsY(i-2, 3):segmentsY(i+2, 3)) = 1;
   end
end
```

The result of this code snippet is shown in Fig.5.

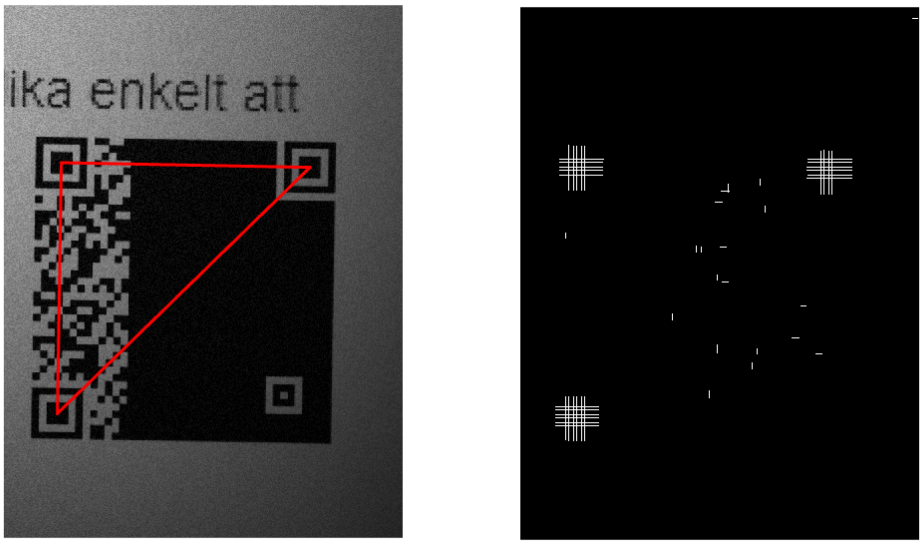

Figure 5: *To the right: the black image with white segments where a match has been found. To the left: the marked fiducial marks.*

When all the white fields in the otherwise black image have been labelled the center of these fields are determined. First a check is made determining if the white fields have a rectangular shape. Then the center of the rectangle is calculated. Sometimes false positives will cause the labelled field to not form a rectangular shape. In such case, the median coordinates of the field are chosen as center point. To make sure the center point is really in the

center of the fiducial mark the pixels in the center of the fiducial mark are counted and the center point is moved to the exact center.

3.3 Rotation

At this stage, the QR-code has been detected. In order to decode the QR-code it needs to be straightened. The first step in the straightening process is to rotate the QR-code so that it has the right orientation. Since the centers of all fiducial marks are now known, they can be used for rotation. In this implementation the two fiducial marks at the top are used. By subtracting one of the fiducial marks at the top from the other the vector between them will be achieved. By normalizing this vector and applying the dot product between it and the x-axis the angle between them can be calculated. This is illustrated in the equation Eq.1.

$$\alpha = cos^{-1}(\hat{v} \cdot \hat{x}) \tag{1}$$

Where α is the achieved angle, $\hat{v}$ is the normalized vector between the fiducial marks and $\hat{x}$ is a normalized vector with the same direction as the x-axis. After the angle has been calculated a check is made whether the vector $\hat{v}$ has a negative x component. If that is the case the resulting angle α will be $\alpha = \alpha - \pi$. The image in Fig.6 shows an image before and after rotation. To not lose the coordinates of the fiducial marks, these coordinate are also rotated by applying a rotation matrix.

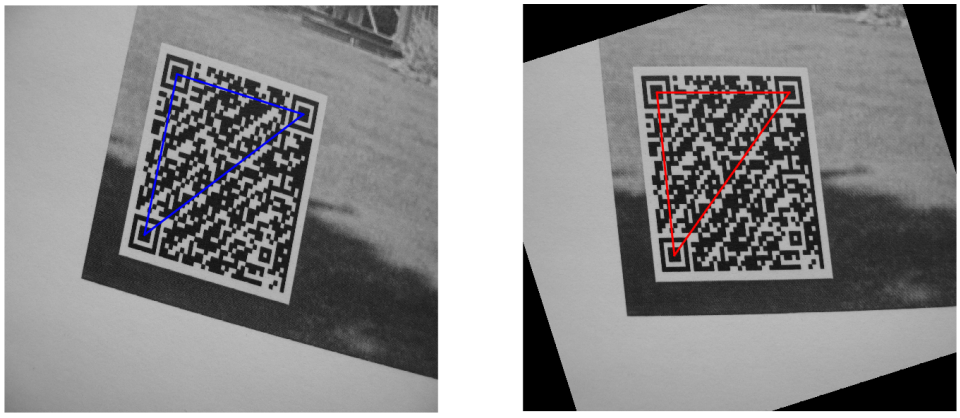

Figure 6: *Before rotation on the left, and after rotation on the right.*

3.4 Adjustment for Perspective Distortion

Even though the QR-code is now rotated properly, it may still have spatial distortion that could possibly make the QR-code unreadable. The most

common spatial distortion among the images in the training set used for this project is perspective distortion. To account for this, projective transformation was used. In projective transformation lines are mapped to other lines. In this implementation the corners of the QR-code together with the center point of the alignment pattern is used as tie points to perform the projective transformation.

3.4.1 Corner Detection

The projective transformation needs a minimum of two 4×2 matrices containing points from the image as input. The two matrices must match in size. One of the two matrices will be called *fixed points* and the other one will be called *moving points*.

The moving points - matrix should contain points in the image that are to be moved or mapped to new points. In this implementation the corners of the QR-code have been used. First, the corners where the fiducial marks were found are determined. This is done by first moving the center point of the fiducial mark by a factor of 7/6 of the number of pixels in the center of the fiducial mark. This is done in either negative or positive x and y direction, depending on the corner. This will return a good estimate for where the corner should be. However, this estimate may not always be good enough depending on the amplitude of the spatial distortion. To make sure an even better corner point is achieved, a small area around the corner is extracted from the image. This small area is then used to more precisely determine where the corner should be.

The fourth corner in the lower right edge of the QR-code is found in a different way. In most versions of QR-codes, or at least the ones used for this study, there is a so called alignment pattern in the lower right area of the QR-code. The alignment pattern consists of white and black pixels with the ratio 1:1:1 in all directions. Searching for this ratio over the whole image would generate too many false positives, making it difficult to determine where the alignment pattern really is. However, it can be found using the normalized 2D cross correlation.

In order to use the normalized 2D cross correlation a template has to be created. The template is created to look like the alignment pattern, and is calculated the following way. As mentioned earlier, the black pixels in the center portion of the fiducial mark have been counted, which means the size of the center portion is known. By dividing the size of the center portion by 3, the approximated size of one bit in the QR-code is achieved. This can now be used to determine the size of the template. The size of the template should be 5 times as big as the size of one bit in the QR-code. Then the

template is filled with black and white pixels in the order white, black and white.

When the template has been created it is used to scan only lower right quarter of the QR-code. The normalized 2D cross correlation function is built in to Matlab and is called *normxcorr2* [4]. This function will return a correlation matrix with values spanning from -1 to 1.

The normalized cross correlation between the template and the image is calculated using Eq.2.

$$\gamma(u,v) = \frac{\sum_{x,y}[f(x,y) - \bar{f_{u,v}}][t(x-u,y-v) - \bar{t}]}{\left\{\sum_{x,y}[f(x,y) - \bar{f_{u,v}}]^2 \sum_{x,y}[t(x-u,y-v) - \bar{t}]^2\right\}^{0.5}} \tag{2}$$

Where $\bar{t}$ is the mean value of the template, $\bar{f}_{u,v}$ is the mean value of the image currently under the template [9].

By extracting the largest value from the correlation matrix the coordinates for the alignment pattern are achieved. These coordinates together with the coordinates for the other corners are then saved in the moving points - matrix.

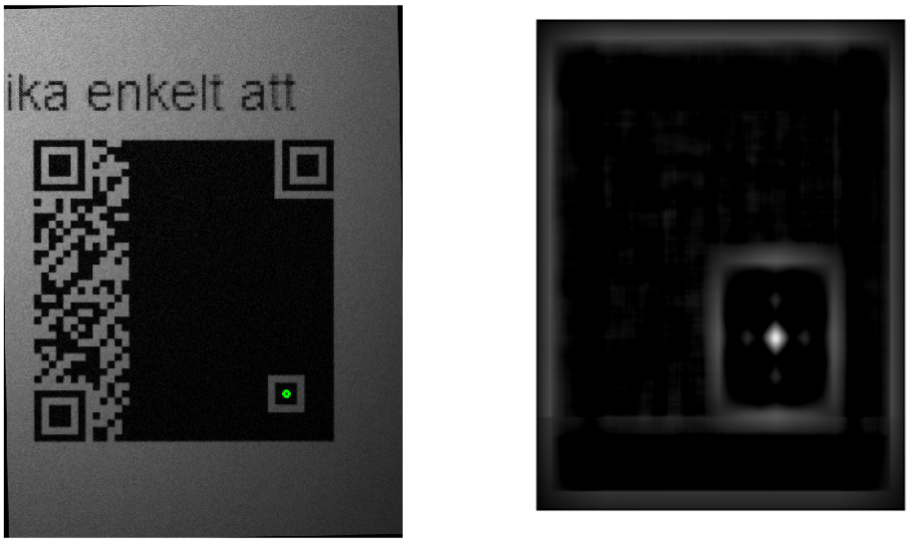

Figure 7: *To the right: the correlation matrix visualized. To the left: The center of the alignment pattern marked with green.*

The fixed points - matrix contains the points we want to move the moving points to. These are our desired, straight points. These points are calculated by using the same size for one bit in the QR-code calculated in the previous step. The estimated height and width are achieved by multiplying the size

of one bit by 41, since it is known that the size of the QR-code is 41x41 bits. Now the coordinates of three desired corners are determined. By calculating three corners, the fourth corner also achieved no extra calculations needed. The fourth corner can be used to calculate where the alignment pattern should be in the straightened up image. The alignment pattern should be seven bits from the fourth corner in both x and y direction. When all these corners have been determined they are passed in to the projection function.

3.4.2 Projective Transformation

The most common distortion that could be fixed with this transform is the perspective distortion where the image was taken from a view with some perspective angle to the QR-code.

$$Tx = x' \tag{3}$$

$$\begin{pmatrix} A & B & C \\ D & E & F \\ G & H & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} x_1{}' & x_2{}' & x_3{}' & x_4{}' \\ y_1{}' & y_2{}' & y_3{}' & y_4{}' \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{4}$$

$$T = x'x^{-1} \tag{5}$$

A linear equation represented by a transformation matrix T applied on the fixed points x that is equal to the corner points x' can be used to extract the transformation matrix in Eq.3 and Eq.5 [8]. Since we have four points with two coordinates there will be total eight linear equations to solve Eq.4.

In Eq.5 the inverse of the fixed points multiplied with the corner points will give the transformation matrix. This inverse could be solved by using Moore-Penrose pseudo inverse [10]. The inverse of the transformation matrix will have to be calculated since the transformation matrix corresponds to the transformation from the fixed points to the distorted corner points. This means that the inverse T^{-1} is the desired transformation matrix.

In matlab the function called *fitgeotrans* [1] was used and it returns a transformation matrix. It uses a built in solver for linear equations which uses a least square solution similar to the Moore-Penrose pseudo inverse [3]. This transformation matrix, together with the original image are sent in to the function *imwarp* [2]. This function applies the transformation matrix to the image and interpolates new sample values in the image with nearest neighbour interpolation.

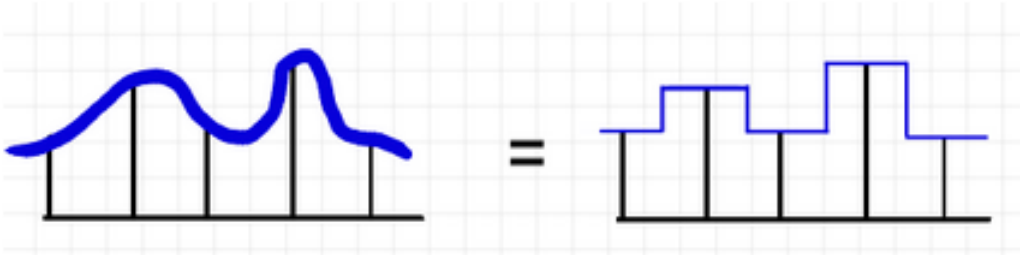

Figure 8: *Neareast neighbour interpolation takes the value of the closest sampled point.*

The projective transformation together with the rotation is the geometric restoration in this implementation. The result before and after the transformation is shown in Fig.9.

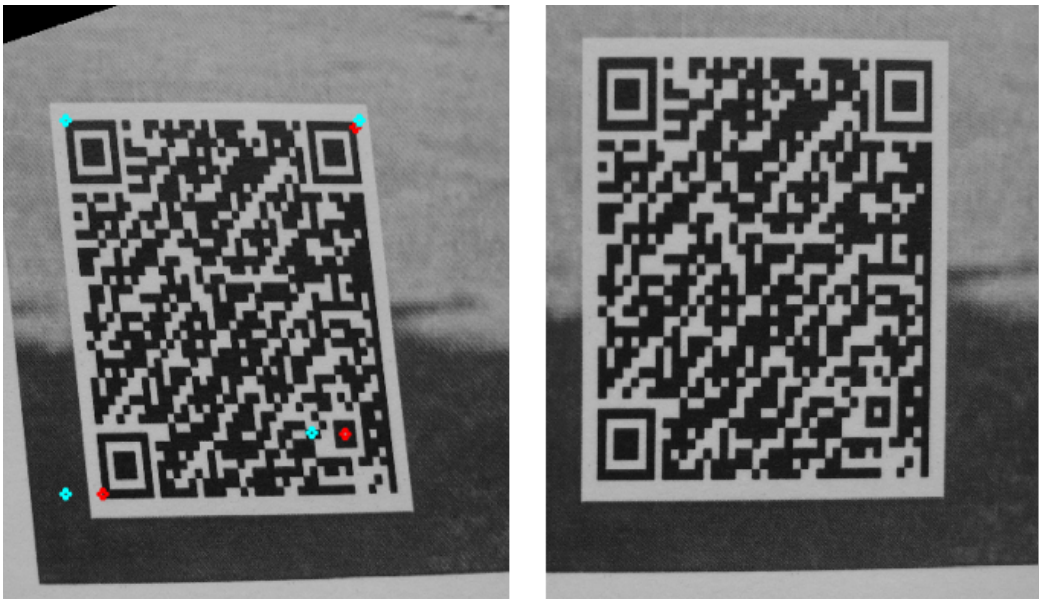

Figure 9: *To the left: Red dots represent moving points. Cyan dots represent fixed points. The point in the upper left corner is the same for both fixed and moving points. To the right is the image after projective transformation.*

3.5 Cropping

When the image has been corrected for perspective distortion, the QR-code is cropped from the image. Since the image now should be rotated and transformed properly, this step can be made without the risk of losing any information. The rectangle which spans the area used for cropping is the same rectangle use in the fixed points - matrix mentioned in the previous section. This is possible since the image is warped and mapped to match

the fixed points. When the QR-code has been cropped from the image, the QR-code is resized so that it becomes a square image.

3.6 Decoding the QR-code

At this stage a binary image of a QR code without the white border is the input to this function. This is where the bits in the QR-code is translated into numbers and then to characters using the ASCII table.

First the dimension of the image is divided into 41x41 since the version 6 QR-codes has these dimensions. The fiducial marks and the alignment pattern are ignored and all bits in sequence of 8 bits(1 byte) are translated into characters and saved in a char array. The image is scanned vertically and when all information in the QR-code has been decoded the char array is transformed into a string and returned as the final result.

Figure 10: *The red dots represent where information is extracted.*

4 Result

The implementation has been tested on 54 images that include three different images with different rotations, distortions and resolution. The algorithm

Name of folder	Images where decoding fails	Successful decoding
Images_Training_1	None	100 % (24/24)
Images_Training_2	None	100 % (6/6)
Images_Training_3	Hus_2e.png	83,3 % (5/6)
Images_Training_4	None	100 % (6/6)
Images_Training_5	None	100 % (8/8)
Images_Training_Illum	None	100 % (4/4)
Total	**Hus_2e.png**	**98,1 % (53/54)**

Table 1: Result of the training set.

returns the correct string for all images except one where a capital A is returned instead of a lower case a. The reason for this error is unknown.

5 Conclusion and Discussion

This implementation works for the majority of the images tested, see Table.1, with a fairly good performance. It can extract QR-codes and decode them in images with quite the amount of distortion and rotation. Given the time spent on this project we are satisfied with the result, but there are many things that could be improved in future work.

5.1 Performance

Matlab is a powerful and fast tool to use when using matrix operations. In a problem like this, where a lot of searching in images has to be made, Matlab is not very fast since traditional loops are not really optimized. There is one place in the implementation where the whole image is scanned. This is the step where the fiducial marks are found, and is by far the heaviest part to calculate in the implementation. Even though optimizations have been made such as only scanning every fourth pixel, preallocating space and using the Matlab function *bwconncomp* the implementation can take up to six seconds to run for large images. The function *bwconncomp* was used instead of counting pixels in the initial step of the implementation.

The average performance in speed is good, with an old laptop[1], an image with the dimensions 3994x2997 which is unusually big, as input take approximately 6 seconds to execute. With a more optimized language like C++ or

[1]Macbook Pro 2011, CPU: 2.3 GHz Intel Core i5 dual core, RAM: 8 GB 1333 MHz DDR3.

similar a an improvement could be made but with Matlab we consider that this is fast enough. See Fig.11.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
main	1	5.716 s	0.003 s	
tnm034	1	5.301 s	0.003 s	
findQR	1	5.271 s	2.951 s	
fixPerspective	1	1.263 s	0.024 s	
binarize	2	0.676 s	0.497 s	

Figure 11: *Benchmark test that shows the time each function takes to execute with an image with the dimensions 3994x2997. The time is shown in real time.*

5.2 Corner detection

The corner detection used in this implementation is sensitive for noise since the algorithm only compares two points in the close neighbourhood of the corner. If there is noise present, there is a risk that the algorithm returns a bad corner. The corners will never be perfectly detected but close enough to achieve an image where it is possible to decode the QR-code. But the corners will lose precision the more geometric distortion there is and at some point the corners will be too inaccurate to use.

A possible improvement could be to implement a well known corner detection like Harris-Stephens algorithm to find more accurate corners and to find the corner in cases where the geometric and distortion is very intense.

5.3 Transformations

The most common distortion in the test images seems to be perspective distortion. Therefore this is the only kind of geometric distortion accounted for. If other geometric distortions like Barrel and Pincushion distortion would be present, the correction may be wrong since these distortions require more than four points in the transformation to return a satisfying result.

5.4 Special cases

As mentioned the implementation works for images with quite much distortion and rotation but if the distortion and rotation would be more extreme, it would probably not work. If the rotation is very big like 90 degrees or more the program would fail since the fiducial marks and alignment patterns

would be at the wrong positions. No check is done for this but could be a good improvement for future work. The algorithm would also fail if the QR-code is reversed.

Also if an image has none or more than one QR-code the implementation would probably fail since it is limited to only be able to find one QR-code. If no QR-code is detected, a future version could possibly give some kind of error message back to the user.

The case where the image has much camera shake distortion [7] is also not considered and could be a possible improvement to add to the pipeline.

References

[1] Mathworks, documentation of matlab function fitgeotrans. `http://se.mathworks.com/help/images/ref/fitgeotrans.html`. Accessed: 2014-12-10.

[2] Mathworks, documentation of matlab function imwarp. `http://se.mathworks.com/help/images/ref/imwarp.html`. Accessed: 2014-12-10.

[3] Mathworks, documentation of matlab function mldivide. `http://se.mathworks.com/help/matlab/ref/mldivide.html`. Accessed: 2014-12-10.

[4] Mathworks, documentation of matlab function normxcorr2. `http://se.mathworks.com/help/images/ref/normxcorr2.html`. Accessed: 2014-12-10.

[5] Luiz FF Belussi and Nina ST Hirata. Fast qr code detection in arbitrarily acquired images. In *Graphics, Patterns and Images (Sibgrapi), 2011 24th SIBGRAPI Conference on*, pages 281–288. IEEE, 2011.

[6] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of graphics, gpu, and game tools*, 12(2):13–21, 2007.

[7] Chung-Hua Chu, De-Nian Yang, Ya-Lan Pan, and Ming-Syan Chen. Stabilization and extraction of 2d barcodes for camera phones. *Multimedia systems*, 17(2):113–133, 2011.

[8] Franz Lemmermeyer. Introduction to algebraic geometry ch.7. `http://www.fen.bilkent.edu.tr/~franz/ag05/ag-07.pdf`, 2005. Accessed 2014-12-10.

[9] JP Lewis. Fast normalized cross-correlation. In *Vision interface*, volume 10, pages 120–123, 1995.

[10] R. Penrose. On best approximate solutions of linear matrix equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52:17–19, 1 1956.