SPEIT SJTU

VS Project

QR Code Detection Report

Tiance WU
117260910057

November 30, 2019

1 Overall

Figure.1 shows the whole process of QR Code detection. In general, the given QR Code is eroded and dilated to gain some basic borders of this QR code. Based on these borders, we could find the edges of the QR code through Hough Transform. We could therefore find out all four corner points of it as well. Afterwards, the holography of this QR Code could be computed according to the coordinates of these four corners, the image could therefore be transformed into an image of bird's-eye-view. Afterwards, The image is tailored and the center and direction of the QR Code could be confirmed by checking its three 'HUI-form' areas. Eventually, the image is rotated into a proper direction and data of QR Code could be extracted. The whole process is done in the file $Transform.cpp$. We entended to encode it with an $main.m$ but our computer has some problem with the MINGW64 compiler of $MatLab$ and we have to finish all process with this C++ code. In order to run the main file $Transform.cpp$, $opencv$ is required. In fact, the whole program is coded with $VisualStudio2019$ with $opencv$ in $C++15$. The final result is called $final.bmp$, it is a 42×42 binary image where white represent 0 and black represent 1. All other necessary files are also stored in the folder. The whole process could also be done by running $QRCodeDetect.exe$.

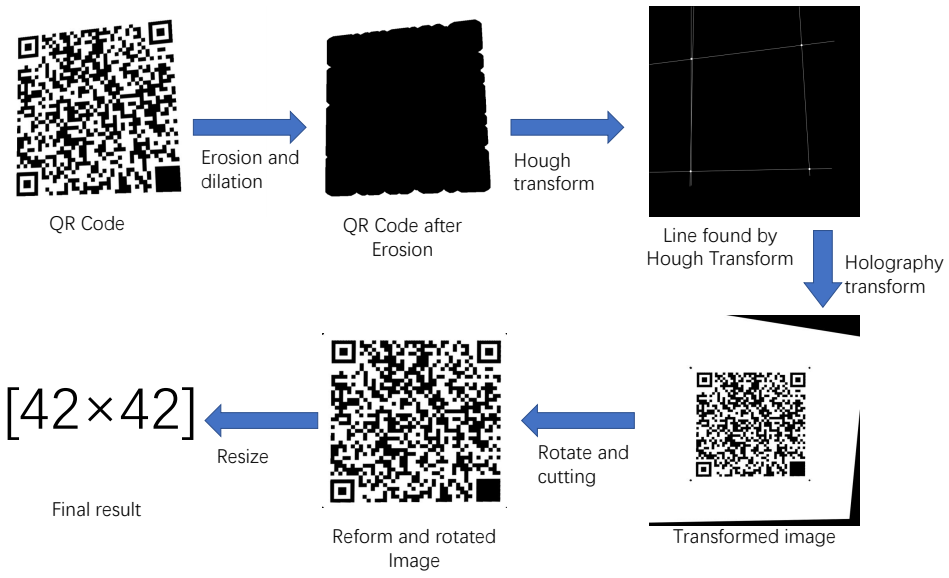

Figure 1: The whole process of QR code detection

2 Main Process

2.1 Erosion and Dilation

In order to get a better result in Hough Transform, dilation is not used in this part. Instead, erostion is used normally between 15 to 20 times. Fig.2 shows an eroded image. It is automatically saved in the program and is named *erosion.bmp*.

After all erosion is done, *erosion.bmp* is formed. Afterwards, we do erosion for another time to it and use *erosion.bmp* itself to subtract the newly-eroded image, which will obtain four borders of the QR Code. The next process is done according to the border.

Figure 2: The whole process of QR code detection

2.2 Hough Transform

After the border is obtained, Hough Transform could be used. The whole space is transformed in $HoughSpace$ to obtain the mose suitable lines. As for only the four edges of QR Code exist in this image, the obtained lines are

for sure these four edges. Figure.3 shows lines obtained by Hough Transform and corner point obtained according to the Line.

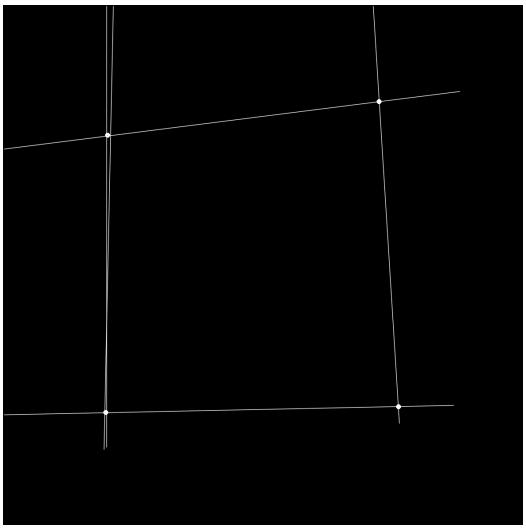

Figure 3: Lines and corner points obtained by Hough Transform

This process causes a problem. As no dilation was done, these four corner points are a little bit more far from the center than expected. However, these hardly affect the process for holography transformation. The main thing requires to be done is tailoring the image after transformation.

2.3 Holography Transformation

By choosing three points (x_1, y_1),(x_2, y_2) and (x_3, y_3) in these four corners, we could create a rectangle with a corner point (x_1, y_1) and the width and height $max(x_2 - x_1, x_3 - x_1)$ and $max(y_2 - y_1, y_3 - y_1)$. By applying the coordinates of the real four corners to this rectangle, we could compute the Holography with $getPerspectiveTransform$ function in *opencv*. Afterwards, with $warpPerspective$ function, the image is transformed. The result is shown in Figure.4.

2.4 Rotate and Cutting

Afterwards, we first cut this image into a rectangle. By drawing the contours of the cutted image, we could fing three 'HUI'-formed areas in the

Figure 4: The transformed image

QR Code. A 'HUI'-formed area has three contours with one surrounding another, while no other part holds the same property. Therefore we could build a $'HUI - detecting - function'$ by using this property.

After finding the three 'HUI-formed' areas, we could compute two things: the center of the QR Code and the fourth corner of the QR Code. We could compute the spatial relationship between these two points immidiately. In order to have a correct position, the fourth corner should at bottom-left of the center. Otherwise, we rotate the image for 90,180 or 270 degrees according to different spatial relationship of the center and the fourth corner. Before doing the rotation, all white areas out of the QR Code in the image should be tailored(these white areas exist because of the former problem of corner points detection). As for the image is now a rectangle, it is easy to erase these white areas.

After all processes are done, the image is resized into 42×42 and binarized. Figure.6 shows the final result. This is a 42×42 binarized image where white represent 0 and black represent 1. The image is called $final.bmp$ in the program and will be automatically formed after running the $Transform.cpp$ or $QRCodeDetect.exe$(which is compiled from $Transform.cpp$). If not considering the problem of $MINGW$ in our computer with $MatLab$, we could also use $MatLab$ commands to form a matrix according to this binary image.

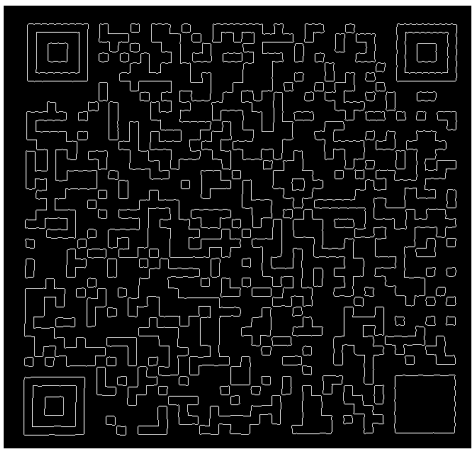

Figure 5: Contours of the cutted image

Figure 6: The final result