OSPP 2020

开源软件供应链点亮计划—暑期2020

Project Proposal

OpenCV China

Project ID：2020098

Implementation of QR-codes decoder and encoder

Applicant：Hollow Man

Hollow Man

Last Modified Time: 2020.6.8

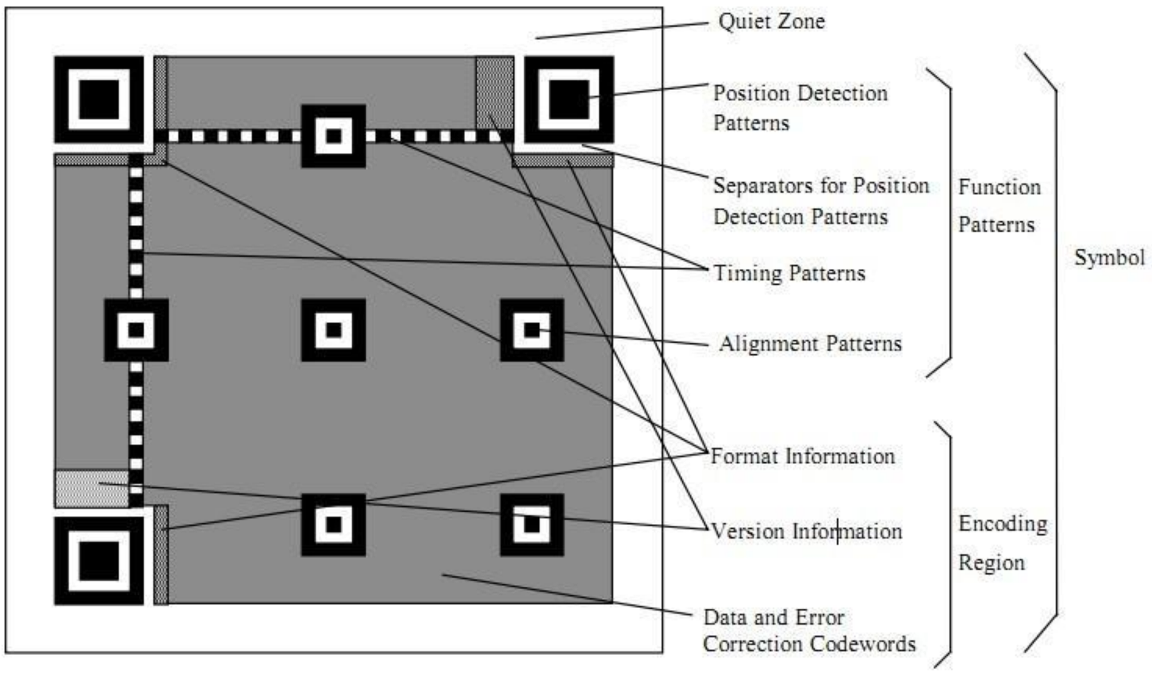
Content

1. Introduction

A. About QR Code

Compared with bar code, the amount of data stored in two-dimensional code is larger, which can contain mixed content such as numbers, characters, and Chinese text; it has certain fault tolerance (can be read normally after partial damage); and it has high space utilization. QR code is a kind of two-dimensional code.

Basic structure of QR Code:



Position Detection Pattern, Timing Patterns, Alignment Patterns: used for the location of the position of QR code. For each QR code, the location is fixed, but the size and specification will be different;

Correction pattern: when the specification is determined, the number and position of correction pattern are also determined;

Format information: indicates the error correction level of the modified QR code, which includes L, M, Q and H;

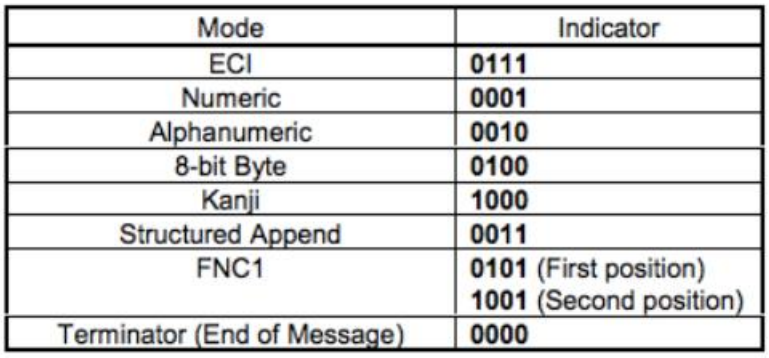
Version information: that is, QR code specifications. QR code symbols have 40 specifications of matrix (generally black and white), from 21 × 21 (version 1) to 177 × 177 (version 40). Each version of symbols adds 4 modules to each side of the previous version.

Data and error correction codewords: the actual stored two-dimensional code information and error correction codeword (used to correct errors caused by two-dimensional code damage).

The QR Code coding process is divided into the following seven steps:

1. **Data analysis**: determine the character type of the code, and convert it into symbol character according to the corresponding character set; select the error correction level, and the higher the error correction level is, the smaller the capacity of the real data is.

2. **Data encoding**: convert data characters into bit stream, one code word for each 8 bits, and form a code word sequence of data as a whole. In fact, if you know the data code sequence, you will know the data content of the QR code.



Mode	Indicator
ECI	**0111**
Numeric	**0001**
Alphanumeric	**0010**
8-bit Byte	**0100**
Kanji	**1000**
Structured Append	**0011**
FNC1	**0101** (First position) **1001** (Second position)
Terminator (End of Message)	**0000**

Take data “01234567” (number) coding as an example:

1) **Group**: 012 345 67

2) **Convert to binary**: 012 → 000000 1100 345 → 0101011001 67 → 1000011

3) **Conversion sequence**: 000000 1100 0101011001 1000011

4) **Character number to binary**: 8 → 000000000

5) **Add mode indicator (refer to figure above)** ***0001***: 00010000001000 0000001100 0101011001 1000011

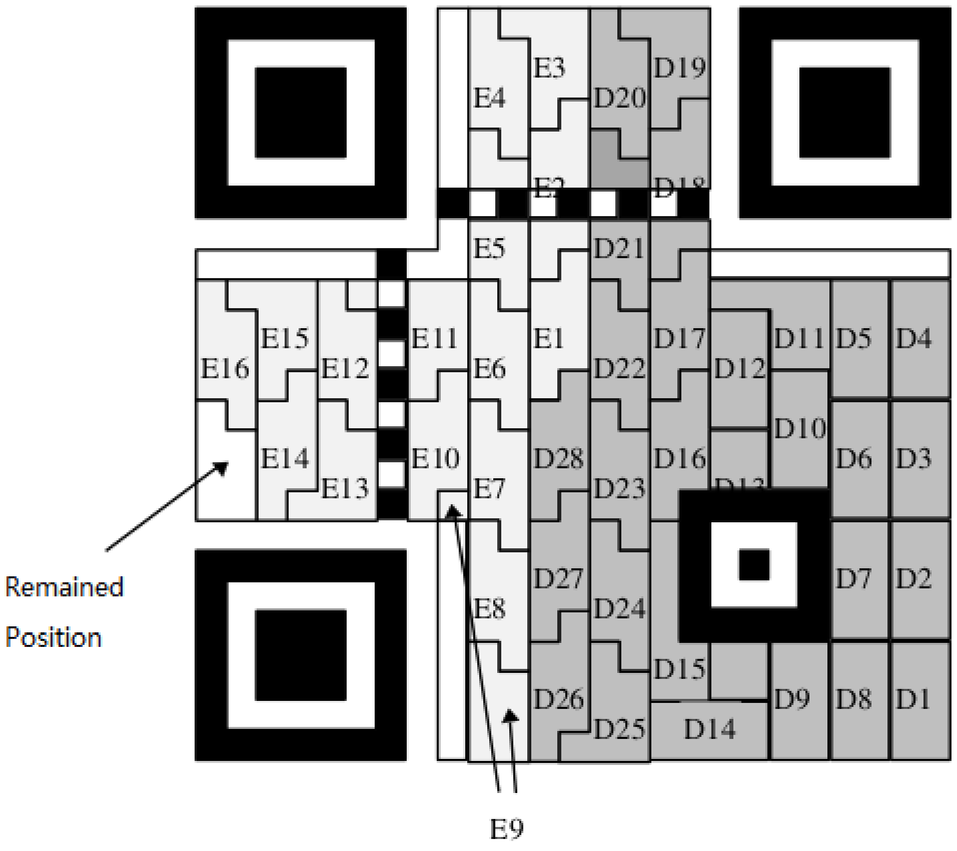
For letters, Chinese, Japanese, etc., there are only differences in the way and mode of grouping, and the basic methods are the same.

3. **Error correction code**: block the above code word sequence as required, generate error correction code word according to the error correction level and block code word, and add the error correction code word to the data code word sequence to become a new sequence. When the two-dimensional code specification and error correction level are determined, the total number of code words and error correction code words it can accommodate are also determined. For example, when the error correction level is h in version 10, the total number of code words can accommodate 346, including 224 error correction code words. That is to say, about one third of the codewords in the two-dimensional code area are redundant. For the 224 error correction codewords, it can correct 112 substitution errors (such as black and white reversed) or 224 sentence reading errors (unreadable or unable to decode), so the error correction capacity is: 112 / 346 = 32.4%

4. **Construct the final data information**: under the condition that the specification is determined, put the sequence generated above into blocks in order, divide the data into blocks according to the regulations, and then calculate each block to get the corresponding error

correction code block, form a sequence of error correction code block in order, and add it to the back of the original data code block. For example: D1, D12, D23, D35, D2, D13, D24, D36 D11, D22, D33, D45, D34, D46, E1, E23,E45, E67, E2, E24, E46, E68，…

5. **Construction matrix**: put the detection graph, separator, positioning graph, correction graph and code word module into the matrix, and fill the above complete sequence into the area of the corresponding two-dimensional code matrix.



6. **Mask**: the mask pattern is used in the coding area of the symbol, so that the dark and light colour (black and white) areas in the two-dimensional code pattern can be optimally distributed.

7. **Format and version information**: generate format and version information and put it into the corresponding area. Version 7-40 contains version information, and all versions without version information are 0. Two positions on the QR code contain version information, which are redundant. The version information consists of 18 bits, 6x3 matrix, in which the data is, for example, version No. 8 when it is 6 bits, 001000 when it is data bit, and the following 12 bits are error correction bits.

B. About the implementation of QR code detection in OpenCV

After OpenCV version 4.0, the QR code location and decoding method has been added based on QUIRC. Its core code is located at "modules/objdetect/src/qrcode.cpp" of OpenCV code base.

The QR code detection class is QRCodeDetector, and its main member methods are:

1. Constructor

2. Destructor

3. void setEpsX(double epsX) and void setEpsY(double epsY): the coefficients of the location codes are detected in the X and Y directions of these two settings. By default, epsx = 0.2 and epsy = 0.1;

4. bool detect(cv::InputArray img, cv::OutputArray points) const: locate QR code;

5. string decode(cv::InputArray img, cv::InputArray points, cv::OutputArray straight_qrcode = cv::noArray()): decode;

6. std::string detectAndDecode(cv::InputArray img, cv::OutputArray points=cv::noArray(), cv::OutputArray straight_qrcode = cv::noArray()): locate and decode.

The principles are as follows:

Fixed scale: 1:1:3:1:1 according to the black-and-white interval of the positioning pattern, that is, five lines with approximate scale obtained by horizontal or vertical scanning are regarded as part of the positioning pattern.

1. After scanning horizontally and vertically, the central points of three patterns are obtained;

2. By using K-means clustering, three sets can be iterated into three centre points, thus the centre of the location pattern can be obtained;

3. Then the order of locating points is determined by the relationship between angle and area.

4. Flood fill gets the outer frame of the location pattern. According to the longest distance of the diagonal, the lower left and the upper right feature points are determined, and then the upper left feature points are determined according to the distance relationship;

5. The distance is used to determine the two farthest points from the upper left corner in the lower left and upper right patterns. The intersection of the previous feature points is the feature point at the lower right corner;

6. Carry out perspective transformation;

7. Decoding.

Details:

The detect method first converts the image to a grayscale image, and then calls the QRDetect class to detect.

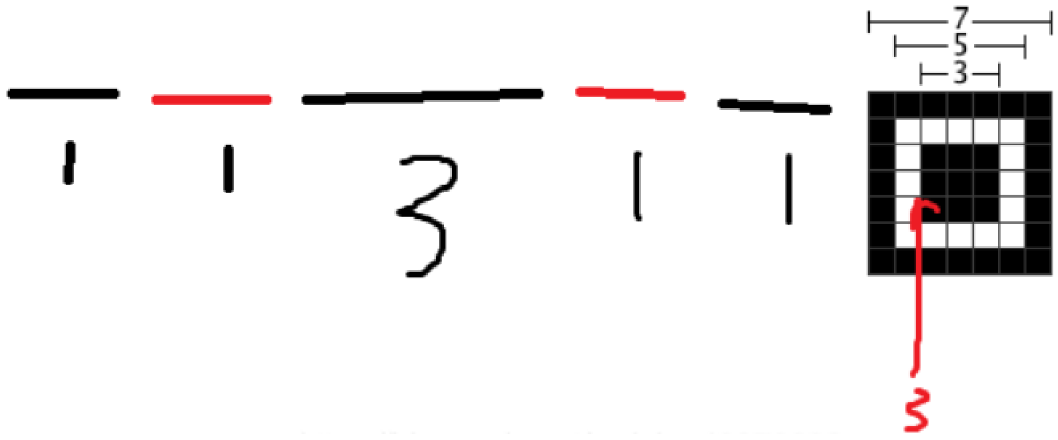
The QRDetect qrdect object in the QRCodeDetector detect method performs init once. In init, first resize the picture with width and height less than 512 to 512, and then perform an adaptive threshold operation, where the blocksize is set to 83. Then localization, in which search horizontal lines and separate vertical lines are used to scan points on rows and columns.

Horizontal scan method**: QRDetect::searchHorizontalLines()**

Since the image has been converted to black-and-white binary image, the location of the centre point can be obtained by analysing the pixel distribution characteristics of each line of black edge.

pixels_position is used to store the position of black and white boundary pixels, test_line stores the length of five consecutive black-and-white line segments before and after a pixel point, and checks whether they are consistent according to the proportion, coefficient eps_vertical limits the degree of inclusion. By this method, the point obtained is always the third boundary point, and subtraction two is the leftmost boundary point.

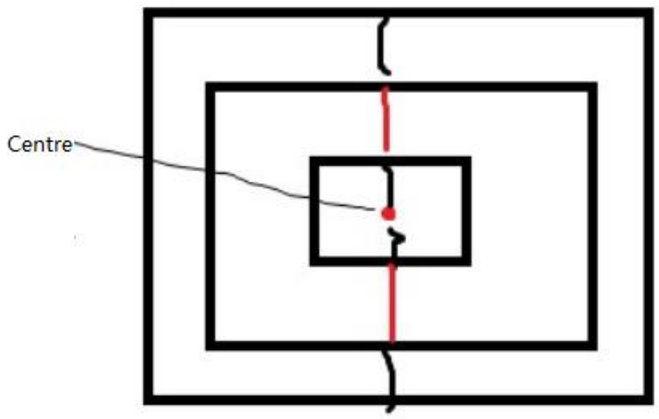
Finally, the Vec3d vector is returned, which respectively stores the number of columns (the first junction point), the number of rows, and the horizontal pixel length of the scanned pattern (calculated by scale).



Vertical scan method: QRDetect::separateVerticalLines(const vector &list_lines)

Continue to process the point set obtained by vertical scanning. The principle is similar to that of horizontal scanning. First scan downward, then scan upward. Then we get six lines instead of five, x = cvRound(list_lines[pnt][0] + list_lines[pnt][2] * 0.5) here is the approximate number of columns to get the center of the pattern. Theoretically, it is located in the middle of the small black square, and three line segments will be obtained up and down. In proportion detection, weight += fabs((test_lines[i] / length) - 3.0/14.0)this part means that only a small part of pixels in the vertical direction of the central block are taken. 3.0/14.0 measures the degree of proximity to the test_[line0] is the first segment to scan down, test_line[3] is the first segment of upward scanning, the closer to the center, the smaller the weight value.

Finally, it returns a set of pixels that meet the horizontal and vertical requirements.



Then, K-means is used to cluster the points on the column into 3 points, among which the number of iterations is 3.

QRDetect::localization(): Centre point acquisition is only part of it. In this method, K-means clustering method is used, and finally three centre points are obtained by exchange iteration. K-means the general principle of clustering

a. Initialize k different centre points;

b. Each training sample is assigned to the cluster represented by the nearest centre;

c. Update the centre point to the mean of all training samples in the cluster;

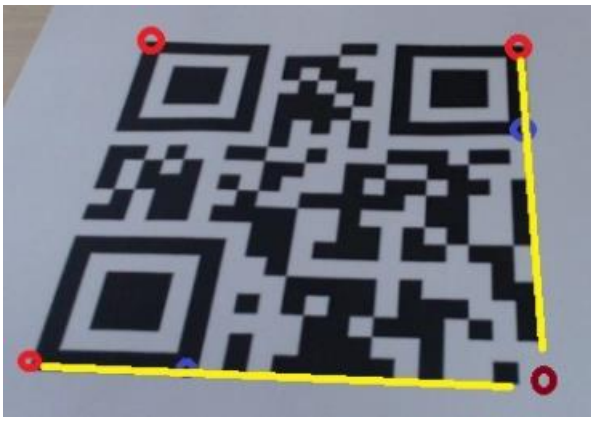
d. Repeat B, C;

Because the horizontal and vertical scanning always get a series of approximate point sets of three locating pattern centre points, using K-means clustering will finally get the more accurate position of the three centre points. If the cluster does not get three points, it will return false.

After clustering, three points are corrected by fixationPoints. Firstly, cosine theorem is used to filter out the case where the cosine value of any angle is less than 0.85, and then a series of corrections are made. Finally, the case where the distance between adjacent points is less than 10pixel is filtered out.

There are two main operations in the computeTransformationPoints() function. One is to find the alignment point:

At the beginning, we will find three red feature points in the following figure, which is easy to understand, because the distance between the points in the upper left corner and the other two points is the same.



Then on the back words in the lower left corner and the upper right corner, find the farthest point from the red point in the upper left corner, then you can get two blue colour feature points. According to the intersection of red and blue dots, we can get the fourth vertex, so we can find all the points needed for alignment.

The second is perspective transformation. In OpenCV, 5 groups of point pairs are selected. In addition to the 4 vertices found above, the centre point is calculated by calculating the diagonal intersection point. Using these 5 points to calculate the Homograph, and then perspective transformation back, we can get an ideal two-dimensional code that is directly opposite to us.

Then **QRDecode::decodingProcess()** calls QUIRC to decode.

C. About this project

This project is expected to create a native QR code decoder and encoder for OpenCV. The decoding and encoding specifications will be in accordance with ISO IEC 18004 2015 standard, and Reed Solomon error correction code will be used. The principle is shown in the introduction of "about QR code".

D. About me

I'm Hollow Man. Now I'm an exchange student at University of Leeds, UK funded by ChinaScholarship Council International Exchange Program for Excellent Undergraduates during2019/20 semester 2. It’s the first time for me to participate in the OSPP activity.

I love the spirit of open source, and that’s why I joined and took an active part in the activities of Lanzhou University Open Source Society at my University. I think the most intriguing thing about open source is that people from various backgrounds work across different timelines without borders to achieve one common goal, the success of their projects without expecting anything in return.

I’ve always been passionate about coding. I have set up a lot of my own small-scale project repositories using various coding languages on Github. However, one thing I haven’t experienced yet is working with an open source organization. Although most students would choose private companies for their summer internship, it would be just a regular monotonous internship because once the duration of the internship is over, my association with the company would end and I would never know how/if my code was being used. Therefore, I decided to contribute to an open source community, work with developers from around the world, and that's why I choose OSPP as my summer internship.

I chose to implement the native QR code decoder and encoder for OpenCV, not only because I think I have enough knowledge of C++ language and Object-Oriented Programming, but also because I have enough interest, passion and experience in using OpenCV. Last year, I participated in the face recognition and access control system competition of our University Lanck Society. I was responsible for the construction of the back-end and realized the function of face recognition using OpenCV. Finally, our project won the third prize and 1000 RMB prize: https://github.com/HollowMan6/Lanck-Face-Recognition-Lock-Competition-Backend-Code . At the same time, because I participated in the excellent undergraduate international exchange programme of the Chinese Scholarship Concil this semester, and I am currently in the UK, it is also doomed that my English ability is completely capable of communicating with upstream developers without obstacles.

I will feel proud of myself if my proposal and hard work are accepted and finally a success to be merged into the upstream.

2. Project implementation

This project is intended to write an OpenCV native QR code decoder to replace the part of QR code decoding currently called QUIRC API, and then write a QR code generator for OpenCV. The decoding and encoding specifications will be in accordance with ISO IEC 18004 2015.

During the project period, I will first refer to QUIRC: https://github.com/dlbeer/quirc for the QR code decoding method implementation. I can refer to the relevant QR code decoding algorithm, error correction rules, data character coding method data and realize a QR code decoder.

Later on, I will implement a QR code generator. The generator act like that if you want to call the generator, you can set the error correction level and generation specification of the QR code in the call parameters. (because the QR code has strong error correction ability, I can also consider supporting self-defined icon in the generated QR code picture centre). The returned image object will be an OpenCV picture object.

3. Timeline

Because I have no other arrangements for summer vacation, and due to the epidemic this year, the school opening time is likely to be postponed, so I can work full-time in the three months from July to September (unlimited time, at least 30 hours per week).

I know a mentor isn't a replacement for Google or stack overflow, so I will only discuss with my tutor the difficulties I cannot overcome. I will send a summary to my tutor by email every two weeks to inform him of the progress of my work so far and the progress of these two weeks.

Here is the schedule:

A. Week 1 (7.1 - 5, 2020)

● Learn more about OpenCV's code specifications.

B. Week 2-3 (7.6 - 19, 2020)

● With the implementation of QUIRC as a reference, I will learn the QR code decoding algorithm, error correction rules, data character coding methods, and started to write the general framework of my QR code decoder.

C. Week 4-5 (7.20 – 8.2, 2020)

●Start to write the core algorithm of QR code decoder, and carry out relevant tests.

D. Week 6-7 (8.3 - 15, 2020)

● Start to prepare interim report.

● Period for any unexpected delay.

E. Interim (8.15, 2020)

Deliverables

● QR code decoding function completed.

F. Week 8-9 (8.17 - 30, 2020)

● Understand the implementation principle of QR code encoder that already existed on GitHub, and start to write the general framework of my QR code encoder.

G. Week 10-11 (8.31 – 9.13, 2020)

●Start to write the core algorithm of QR code encoder, and carry out relevant tests.

H. Week 12-14 (9.14 - 30, 2020)

● Proceed with the closing report.

● Period for any unexpected delay.

I. Closing(9.30, 2020)

Deliverables

● Full QR code decoding and encoding functions for OpenCV completed.

J. Future (9.31, 2020 -)

I will continue contributing and do my best to maintain the OpenCV project after I successfully complete the OSPP project and if possible, I am happy to be a mentor of OpenCV project for OpenCV in the future OSPP.

4. Project Reference (Some in Chinese)

1. ISO IEC 18004 2015 QR code Standard: https://github.com/yansikeim/QR-Code/blob/master/ISO%20IEC%2018004%202015%20Standard.pdf
2. Details and principles of QR code generation: https://blog.csdn.net/hk_5788/article/details/50839790
3. Simple QR code generation practice: https://zhuanlan.zhihu.com/p/21463650
4. Explanation of Reed-Solomon error correction code: https://www.jianshu.com/p/8208aad537bb
5. Implementation of a QR code encoder: https://github.com/gil9red/QRCodeGenerator
6. Analysis of OpenCV 4.0.0 QR code identification code: http://www.p-chao.com/2018-11-23/opencv4-0-0%E4%BA%8C%E7%BB%B4%E7%A0%81%E8%AF%86%E5%88%AB%E4%BB%A3%E7%A0%81%E7%AE%80%E6%9E%90/
7. OpenCV's core code about QR Code: https://github.com/opencv/opencv/blob/master/modules/objdetect/src/qrcode.cpp