UM0300

AmebaPro QR Code Detection user guide

Abstract

The user guide introduces QR Code detection with ZBAR algorithm and the QR Code scanner example on AmebaPro.

Table of Contents

1 QR Code Introduction

QR Code (Quick Response Code), first designed in 1994 for the automotive industry in Japan, has become common in consumer advertising. Typically, a smartphone is used as a QR Code scanner, displaying the Code and converting it to some useful form.

1.1 QR Code Version

QR Code consists of black squares arranged in a square grid on a white background. Here are some sample QR Code symbols:

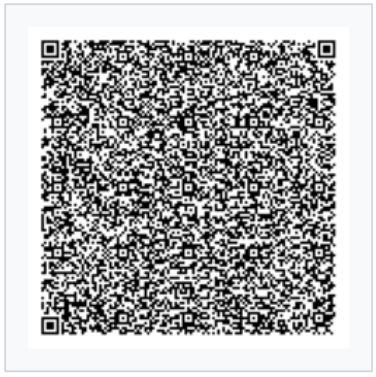
Version 1 (21×21). Content: "Ver1"

Version 2 (25×25). Content: "Version 2"

Version 3 (29×29). Content: "Version 3 QR Code"

Version 4 (33×33). Content: "Version 4 QR Code, up to 50 char"

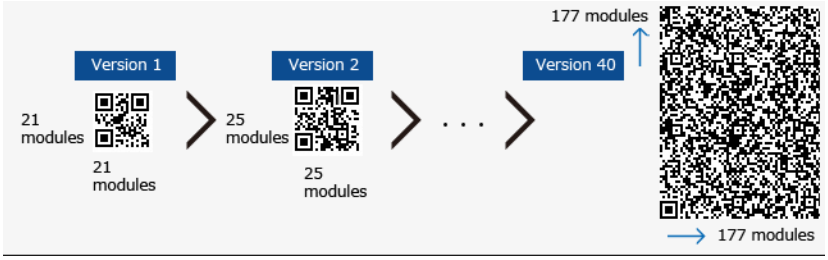
Version 10 (57×57). Content: "VERSION 10 QR CODE, UP TO 174 CHAR AT H LEVEL, WITH 57X57 MODULES AND PLENTY OF ERROR-CORRECTION TO GO AROUND. NOTE THAT THERE ARE ADDITIONAL TRACKING BOXES"



Version 25 (117×117 enlarged to 640x640)

The symbol versions of QR Code range from Version 1 to Version 40. Each version has a different module configuration or number of modules. "Module configuration" refers to the number of modules contained in a symbol, commencing with Version 1 (21 × 21 modules) up to

Version 40 (177 × 177 modules). Each higher version number comprises 4 additional modules per side.



Each QR Code symbol version has the maximum data capacity according to the amount of data, character type and error correction level. In other words, as the amount of data increases, more modules are required to comprise QR Code, resulting in larger QR Code symbols.

1.2 Error Correction Feature

QR Code has error correction capability to restore data if the Code is dirty or damaged. Four error correction levels are available for users to choose according to the operating environment. Raising this level improves error correction capability but also increases the amount of data QR Code size.

To select error correction level, various factors such as the operating environment and QR Code size need to be considered. Level Q or H may be selected for factory environment where QR Code gets dirty, whereas Level L may be selected for clean environment with the large amount of data. Typically, Level M (15%) is most frequently selected.

QR Code Error Correction Capability*	
Level L	Approx 7%
Level M	Approx 15%
Level Q	Approx 25%
Level H	Approx 30%

*Data restoration rate for total codewords (codeword is a unit that constructs the data area. One codeword of QR Code is equal to 8 bits.)

1.3 Determine the Version of QR Code

Suppose the data to be input consists of 100-digit numerals. This can be achieved by following the steps described below:

1. Choose "numeral" as the type of input data.

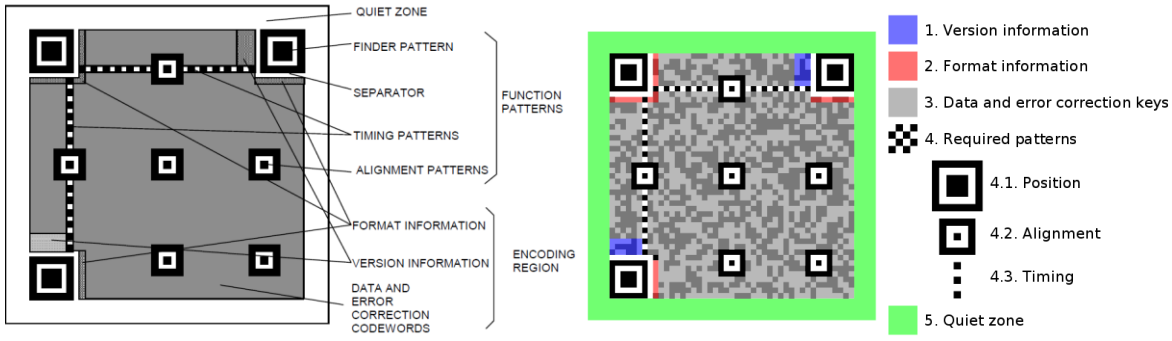
2. Choose a data correction level from the alternatives of L, M, Q and H. (M is assumed here.)

3. Find a figure in the table, 100 or over and the closest to 100 that is at the intersection with a correction level M row. The number of the version row that contains this figure is the most appropriate version number.

Version	Modules	ECC Level	Data bits (mixed)	Numeric	Alphanu-meric	Binary	Kanji
1	21×21	L	152	41	25	17	10
		M	128	34	20	14	8
		Q	104	27	16	11	7
		H	72	17	10	7	4
2	25×25	L	272	77	47	32	20
		M	224	63	38	26	16
		Q	176	48	29	20	12
		H	128	34	20	14	8
3	29×29	L	440	127	77	53	32
		M	352	101	61	42	26
		Q	272	77	47	32	20
		H	208	58	35	24	15

1.4 Symbol Structure

Each QR Code symbol shall be constructed of nominally square modules set out in a regular square array and shall consist of an encoding region and function patterns, namely finder, separator, timing patterns, and alignment patterns. Function patterns do not encode data. The symbol shall be surrounded on all four sides by a quiet zone border.

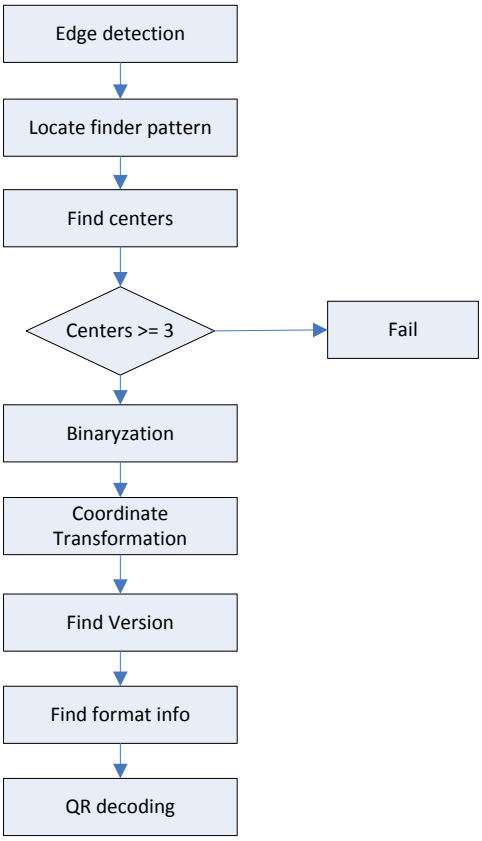


2 Basic Algorithm of ZBAR

QR Code system consists of 2 parts, software and printer to generate QR Codes and a scanner and application program to read them. This will mainly introduce the latter, which is how to detect and decode a QR Code.

ZBAR and ZXING are commonly used methods for QR Code detection at present. ZBAR is based on C language which is suitable for embedded use, and its speed is better than ZXING, so we use ZBAR for the QR Code detection and decoding.

The algorithm locates the three distinctive squares at the corners of the QR Code image, using a smaller square (or multiple squares) near the fourth corner to normalize the image for size, orientation, and angle of viewing. The small dots throughout the QR Code are then converted to binary numbers and validated with an error-correcting algorithm. The basic algorithm flow is as follows:



2.1 Edge Detection

Scan the image pixel by pixel and pixel accuracy can be controlled according to the scan density setting. First scan horizontally, then vertically, so image scanning is done in line with Z font mode.

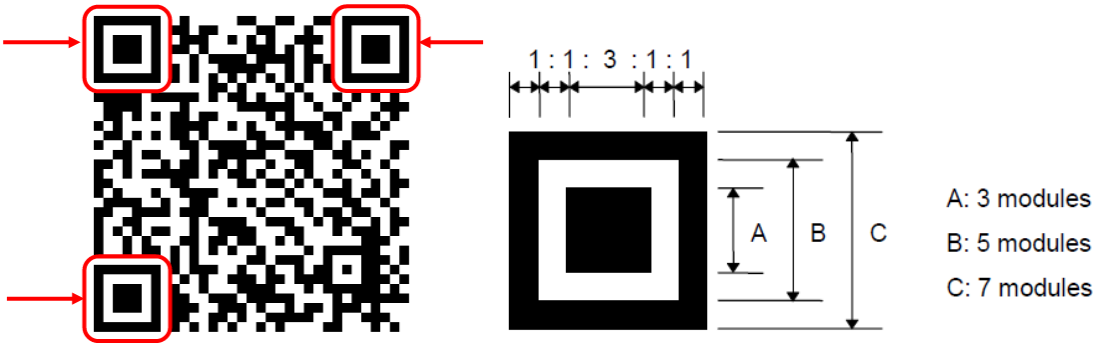
Weighted moving average smoothing is used and the window size is N=4.

Then Detect edge using two order differential. Edge Decision Rule: The position where the second derivative is zero is the maximum or minimum value of the first order, so it is

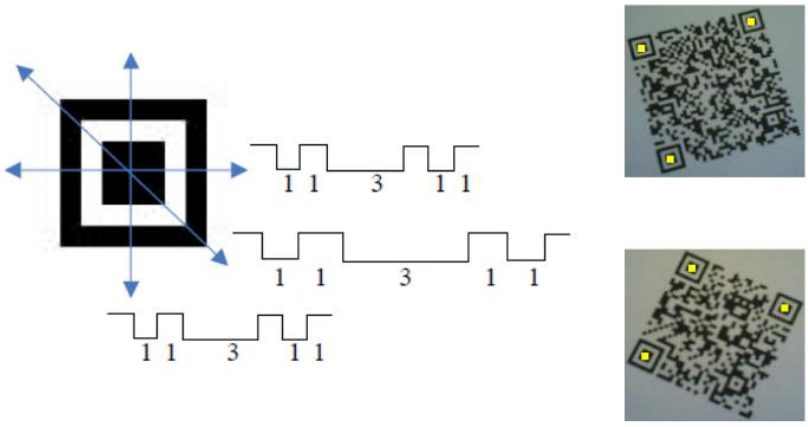
considered as the edge point; there must be edge point in the place where the sign of the second derivative changes.

2.2 Locate Finder Pattern

The finder pattern in QR Code consists of three identical finder patterns located at three of the four corners of the symbol. Module widths in each finder pattern form a dark-light-dark-light-dark sequence the relative widths of each element of which are in the ratios 1 : 1 : 3 : 1 : 1.



According to the related properties of the triangle, the QR Code rotates at any angle, and the size ratio between the black and white modules remains unchanged.



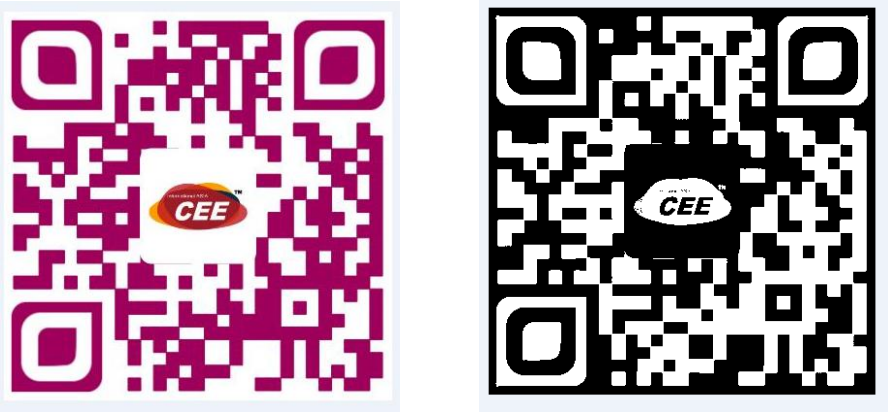
2.3 Find Centers

Locate a set of putative finder centers in the image. First we search for horizontal and vertical lines that have (dark:light:dark:light:dark) runs with size ratios of roughly (1:1:3:1:1). Then we cluster them into groups such that each subsequent pair of endpoints is close to the line before it in the cluster. This will locate many line clusters that don't cross a finder pattern,

but it will filter most of them out. Where horizontal and vertical clusters cross, a prospective finder center is returned.

2.4 Binaryzation

Transform grayscale images into binary image by adaptive threshold. This compares the current pixel value to the mean value of a window surrounding it. The threshold is T = m / N - D, where D = 3, m is the sum of pixels in the (x, y) coordinate w * h domain, and if T > F (x0, y0), the result is zero, otherwise it is 1. To avoid division, T > F (x0, y0) is converted to f (x0, y0) * w * H + 3 > m, and the multiplication is converted to shift operation. For m, the sum of H columns is initialized first, then the first row of the window is subtracted from the next row and the next row of the window is added.



(a) Before binaryzation (b) After binaryzation

2.5 Coordinate Transformation

The original QR Code image may be oblique, and the coordinate transformation is to correct the oblique picture into a positive square.

Determine whether the three points are on the same straight line first and return error if they are on the same line.

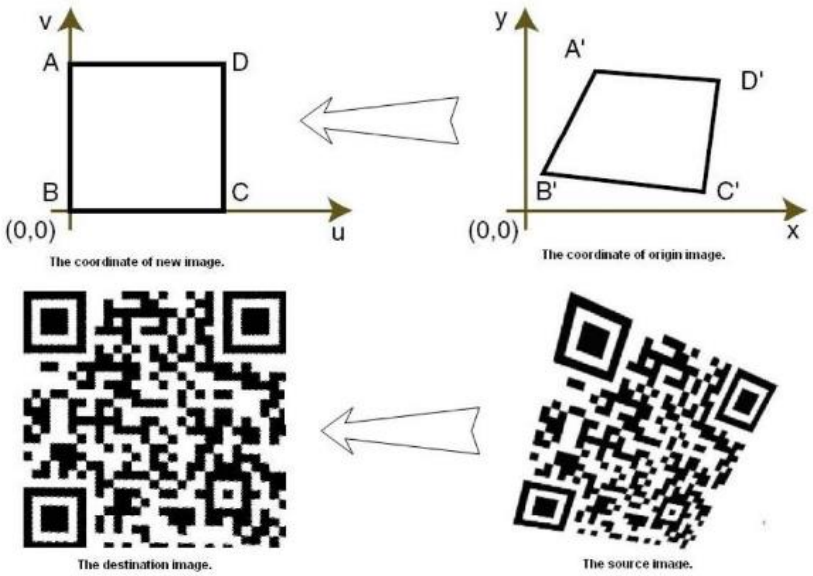
Find out the ordinal number corresponding to the maximum distance between two points in three points, consider the two points corresponding to the maximum distance as two diagonal symbols, and then find the upper left corner, the point corresponding to the maximum distance is the upper left corner of the symbol.

Estimate the module size and version number from the two opposite corners. The module size is not constant in the image, so we compute an affine projection from the three

points we have to a square domain, and estimate it there. Although it should be the same along both axes, we keep separate estimates to account for any remaining projective distortion.

Upgrade the affine homography to a full homography to accurately estimate module size and version information, read format information, and version information decoding.

If the estimated versions are significantly different, reject the configuration. Otherwise we try to read the actual version data from the image. If the real version is not sufficiently close to our estimated version, then we assume there was an unrecoverable decoding error (so many bit errors we were within 3 errors of another valid Code), and throw that value away. Reads the format info bits near the finder modules and decodes them.



2.6 QR Decoding

After the version and format information is obtained, the function area of QR Code has been identified and decoded, and then the data area of QR Code is parsed.

First, the image is removed by mask processing, and the location pattern in the image is identified. The QR Code is then removed from the functional area to the bit stream of 0 and 1. Reed-Solomon error correction algorithm is used to check and correct the extracted bit stream, and the final identified bit stream is output. The function analyzes and judges the resulting bit stream, determines what encoding mode the current QR Code belongs to, and decodes the bit stream after finding the corresponding encoding mode, and finally obtains the decoding result of QR Code.

3 QR Code Scanner Example Introduction

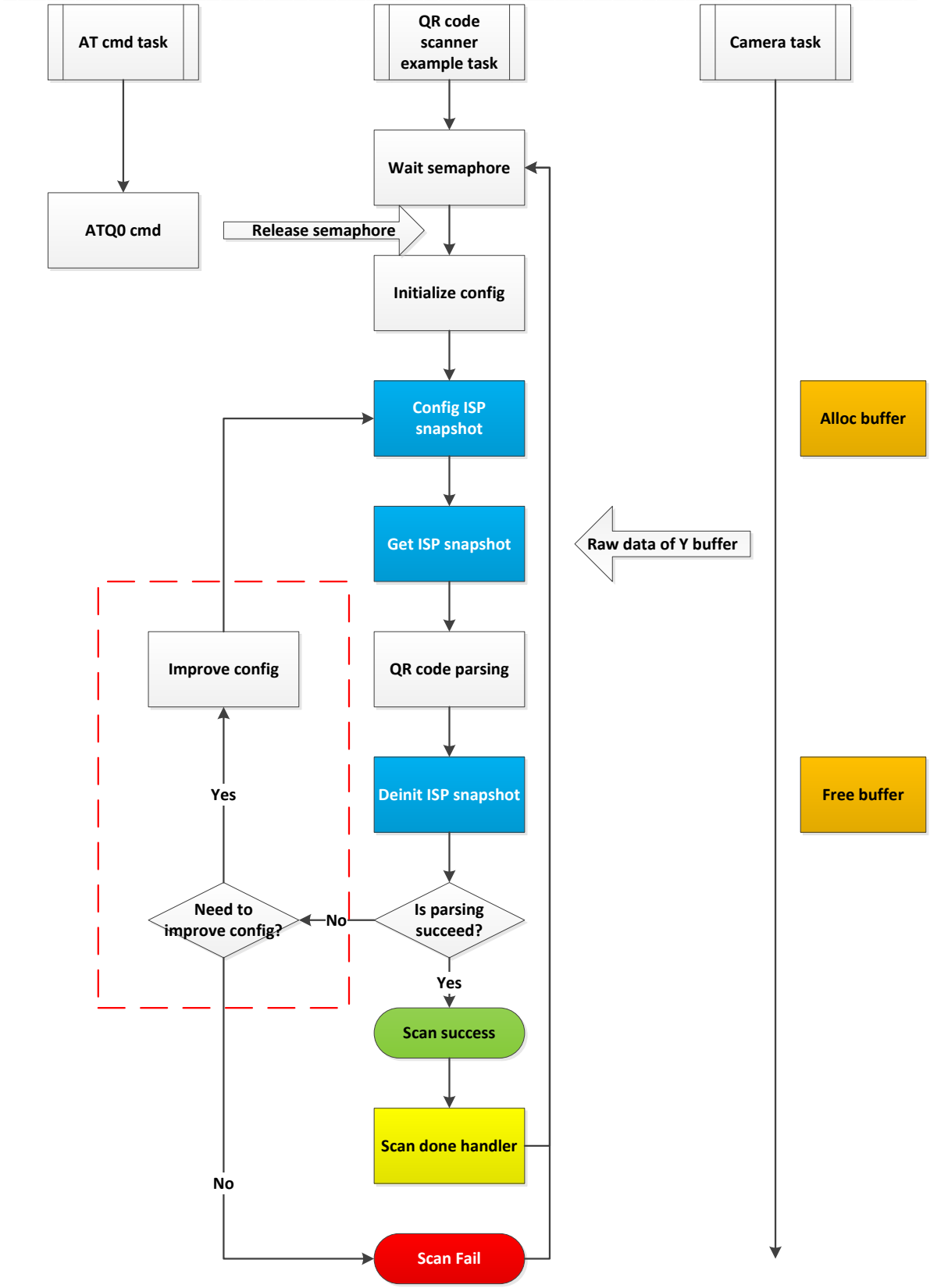
3.1 Overview of Scan Flow

In the example, QR Code scanner example task *example_qr_code_scanner_thread* is created and waits for the semaphore *g_qr_code_scanner_sema*. When ATQ0 command releases the semaphore, QR Code scanner example task starts parsing QR Code.

In order to save memory, QR Code scanner example task shares memory with camera task. QR Code scanner example task uses API *yuv_snapshot_isp_config()* to init ISP snapshot and allocate memory, uses API *yuv_snapshot_isp()* to get raw data of Y buffer of the snapshot from camera task, uses API *yuv_snapshot_isp_deinit()* to deinit ISP snapshot and free memory.

If QR Code parsing success, scan done handler *example_qr_code_scanner_done_event* will be executed, otherwise the image resolution or scan density may be improved and try again. If a WIFI QR Code is parsed, Ameba Pro will get the SSID, password, security type and hidden type to connect to the AP.

The flow chart of QR Code scanner example is shown as follows.


AT cmd task
QR code scanner example task
Camera task
Wait semaphore
ATQ0 cmd
Release semaphore
Initialize config
Config ISP snapshot
Alloc buffer
Get ISP snapshot
Raw data of Y buffer
Improve config
QR code parsing
Deinit ISP snapshot
Free buffer
Yes
Need to improve config?
No
Is parsing succeed?
Yes
Scan success
Scan done handler
No
Scan Fail

3.2 ATQ0 Command

ATQ0 command is used to start parsing QR Code. When ATQ0 command executes, the semaphore *g_qr_code_scanner_sema* is released. QR Code scanner example task obtains the semaphore and starts parsing QR Code.

3.3 QR Code Parsing API

QR Code parsing API is opened to customers. Introduction of the API is shown as follows.

```
typedef enum {
        QR_CODE_SUCCESS = 0,
        QR_CODE_FAIL_UNSPECIFIC_ERROR = 1,
        QR_CODE_FAIL_NO_FINDER_CENTER = 2,
        QR_CODE_FAIL_DECODE_ERROR = 3
} qr_code_scanner_result;

/*
 * @brief                This function parses the image which contains a QR code and returns the result string if parsing success.
 * @param [raw_data]:    The pointer to the raw data of the luminance value of the image to be parsed.
 * @param [width]:       The width of the image to be parsed.
 * @param [height]:      The height of the image to be parsed.
 * @param [x_density]:   The vertical scan density.
 * @param [y_density]:   The horizontal scan density.
 * @param [string]:      The pointer to the result string.
 * @param [len]:         The pointer to the length of the result string.
 * @return               The parsing result of the image.
 */
qr_code_scanner_result qr_code_parsing(unsigned char *raw_data, unsigned int width, unsigned int height, int x_density, int y_density, unsigned char *string, unsigned int *len);
```

3.4 QR Code Scanner Configuration

QR Code scanner configuration mainly contains two parts, image resolution and scan density. Image resolution is used to init and configure ISP snapshot, which supports 480P (640 * 480), 720P (1280 * 720) and 1080P (1920 * 1080). Scan density is used to configure QR Code scanner, which supports two value, 1 and 2.

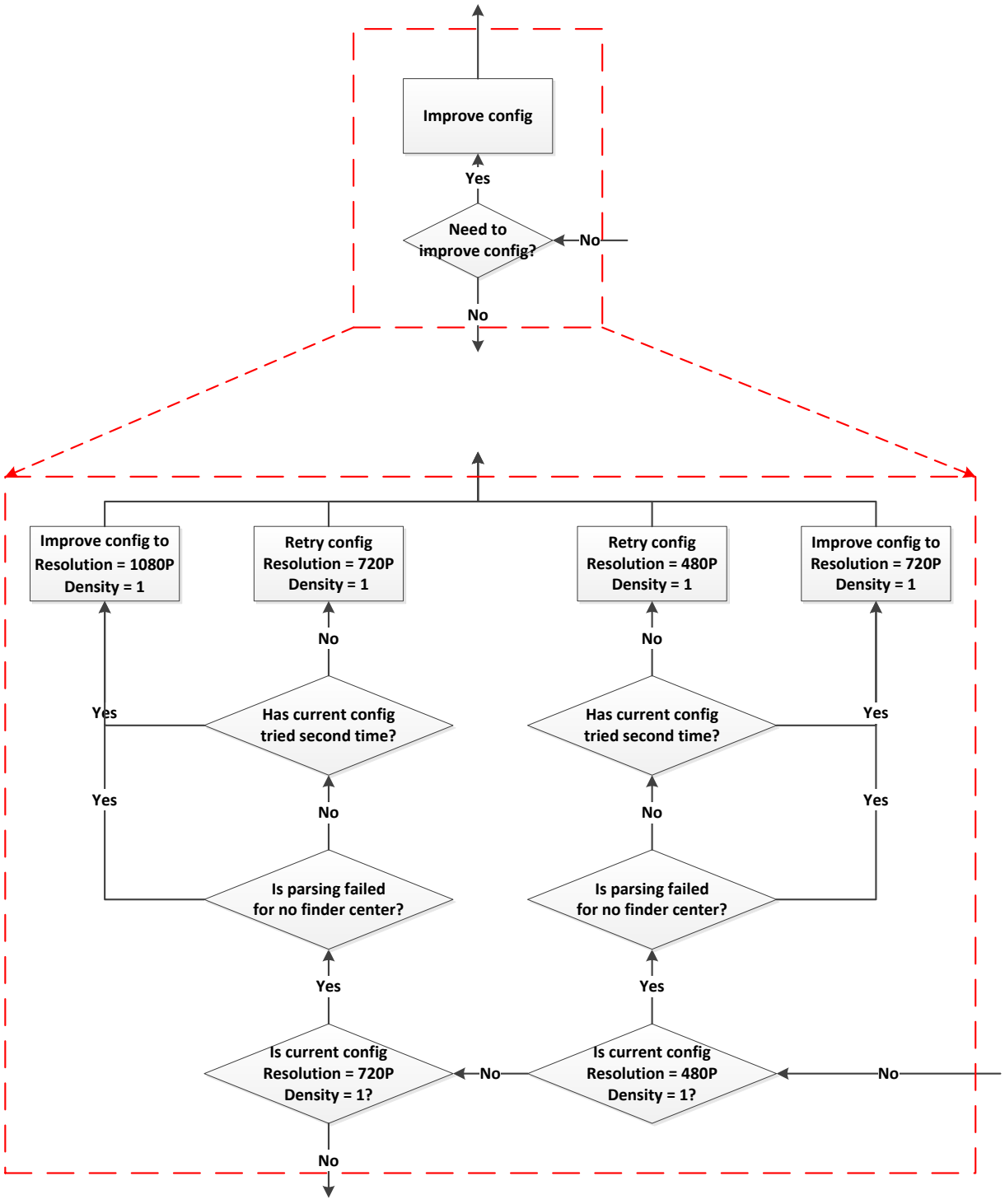
Combination of different image resolution and scan density can form six different QR Code scanner configurations. In the example, the six configurations are named *QR_CODE_480P_DENSITY_2*, *QR_CODE_480P_DENSITY_1*, *QR_CODE_720P_DENSITY_2*, *QR_CODE_720P_DENSITY_1*, *QR_CODE_1080P_DENSITY_1* and *QR_CODE_1080P_DENSITY_1*.

Each configuration costs different time to parse QR Code and also has different suitable distance as follows.

Image Resolution	Scan Density	Test Item	Version 3	Version 4	Version 5	Version 6
480P	2	Longest suitable distance (cm)	30	20	20	20
		Average used time (ms)	287	296	297	305
	1	Longest suitable distance (cm)	40	40	30	20
		Average used time (ms)	374	385	394	404
720P	2	Longest suitable distance (cm)	40	40	30	30
		Average used time (ms)	536	532	566	573
	1	Longest suitable distance (cm)	50	50	40	40
		Average used time (ms)	950	947	977	986
1080P	2	Longest suitable distance (cm)	60	50	50	50
		Average used time (ms)	980	1009	1009	1019
	1	Longest suitable distance (cm)	70	60	60	50
		Average used time (ms)	1890	1866	1887	1910

Considering the longest suitable distance and average used time of each configuration, we choose configuration *QR_CODE_480P_DENSITY_1*, *QR_CODE_720P_DENSITY_1* and *QR_CODE_1080P_DENSITY_1* to complete the QR Code scanner example. The configuration choice rule of the QR Code scanner example is shown as following flow chart.

Of course, customers may choose their own configurations if they care more about time cost or suitable distance.


Improve config
Yes
Need to improve config?
No
No
Improve config to Resolution = 1080P Density = 1
Retry config Resolution = 720P Density = 1
Retry config Resolution = 480P Density = 1
Improve config to Resolution = 720P Density = 1
No
No
Yes
Has current config tried second time?
Has current config tried second time?
Yes
Yes
No
No
Yes
Is parsing failed for no finder center?
Is parsing failed for no finder center?
Yes
Yes
Is current config Resolution = 720P Density = 1?
No
Is current config Resolution = 480P Density = 1?
No
No

3.5 WIFI QR Code Format

The format of a WIFI QR Code is "WIFI:S:SSID;T:<WPA|WEP|nopass>;P:<password>;H:<true|false>;;". These characters shall be encoded with Byte Mode when generated a QR Code.

Where

S represents SSID.

T represents security type, can be "WEP", "WPA", "nopass" or omitted. If T is omitted, also means no password.

P represents password, can be omitted. It will be ignored if T is "nopass" or omitted, in which case P shall be omitted.

H represents hidden type, can be "true", "false" or omitted. If H is omitted, also means SSID is not hidden.

Order of parameters does not matter. Special characters ":" ";" and "\" should be escaped with character "\" before in SSID and password.

3.6 Used Heap Size and Coding Size

Max used heap size during QR Code parsing is approximately 100KB.

The QR Code scanner example relevant coding size is shown as follows.

Module	*ro code (Byte)*	*ro data (Byte)*	*rw data (Byte)*
application_is: [1]			
example_qr_code_scanner.o	*1748*		*64*
Total:	*1748*		*64*
lib_qr_code.a: [12]			
bch15_5.o	*588*		
binarize.o	*920*		
decoder.o	*404*		
image.o	*230*		
img_scanner.o	*2078*		
isaac.o	*856*		
qr_code_scanner.o	*552*		
qr_finder.o	*274*		
qrdec.o	*22484*		
qrdectxt.o	*1084*		

refcnt.o	*2*
rs.o	*2112*
scanner.o	*618*
symbol.o	*372*
util.o	*302*
---	*---*
Total:	*32876*

4 QR Code Scanner Example Test Report

4.1 How to Test QR Code Scanner Example

Define *CONFIG_EXAMPLE_QR_CODE_SCANNER* to 1 in platform_opts.h.

4.2 Test WIFI QR Code

A typical WIFI QR Code “WIFI:S:ABCDEFGH;T:WPA;P:12345678;;” (8 bytes SSID and 8 bytes password) has 34 bytes at least, it shall be a Version 3 QR Code with error correction level L. If a WIFI QR Code has the longest 32 bytes SSID and the longest 64 bytes password (ignored escape character), it shall be a Version 6 QR Code with error correction level L.

Therefore, in our test, Version 3 ~ 6 WIFI QR Codes with error correction level L are tested in different distance and different angle.

Where

Version 3 WIFI QR Code:
“WIFI:S:ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef;T:WPA;P:ghi;;”

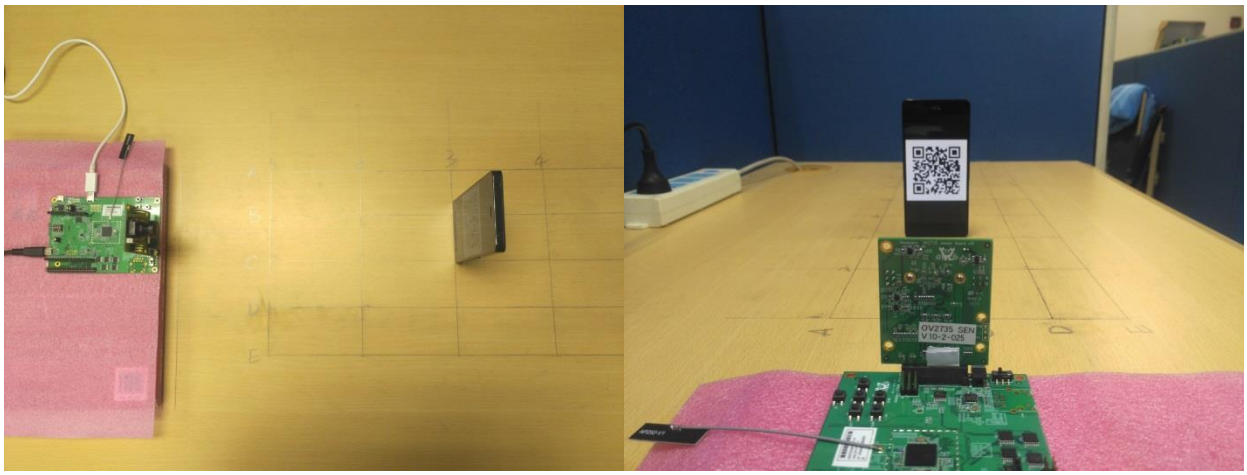
Version 4 WIFI QR Code:
“WIFI:S:ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef;T:WPA;P:ghijklmnopqrstuvwxyz;H:false;;”

Version 5 WIFI QR Code:
“WIFI:S:ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef;T:WPA;P:ghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQR;H:false;;”

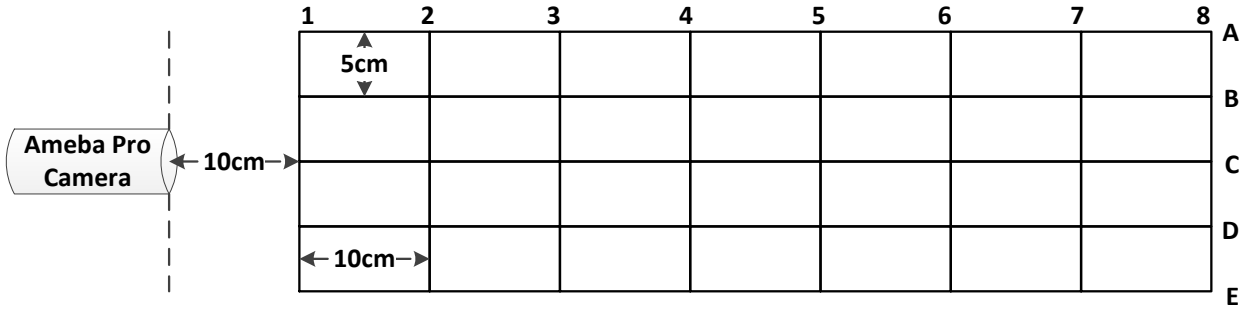
Version 6 WIFI QR Code:
“WIFI:S:\:\;\\\:\;\\\:\;\\\:\;\\MNOPQRSTUVWXYZabcdef;T:WPA;P:ghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefgh;H:false;;”

4.3 Test Environment

Test environment is shown as follows. The center of QR Code is aligned with the center of camera on Ameba Pro horizontally.



In order to test on different distance and different angle, the QR Code is placed on different point as following sketch from the top view. Distance between two horizontal points (1 ~ 8) is 10cm, which between two vertical points (A ~ E) is 5cm.



Each point will be tested 100 times. Success times, fail times and average used time shall be recorded.

4.4 Test Result

Test result of Version 3 QR Code is shown as following table.

Vertical Point \ Horizontal Point		1	2	3	4	5	6	7	8
A	Success times	72	100	100	100	100	100	54	0
	Fail times	28	0	0	0	0	0	46	100
	Average used time (ms)	1427	415	382	384	1628	1747	3079	
B	Success times	100	100	100	100	100	100	73	0
	Fail times	0	0	0	0	0	0	27	100
	Average used time (ms)	440	396	387	462	1943	2333	3341	
C	Success times	100	100	100	100	100	100	71	0
	Fail times	0	0	0	0	0	0	29	100
	Average used time (ms)	522	381	412	393	1711	1967	3036	
D	Success times	100	100	100	100	100	100	100	0
	Fail times	0	0	0	0	0	0	0	100
	Average used time (ms)	398	363	378	452	1643	1577	2906	
E	Success times	77	100	100	100	100	100	86	0
	Fail times	23	0	0	0	0	0	14	100
	Average used time (ms)	1374	363	357	543	1670	1930	3012	

Test result of Version 4 QR Code is shown as following table.

Vertical Point \ Horizontal Point		1	2	3	4	5	6	7	8
A	Success times	100	100	100	100	100	100	0	0
	Fail times	0	0	0	0	0	0	100	100
	Average used time (ms)	1428	382	382	818	1178	3886		
B	Success times	100	100	100	100	100	100	0	0
	Fail times	0	0	0	0	0	0	100	100
	Average used time (ms)	433	374	373	493	1195	2920		
C	Success times	100	100	100	100	100	100	0	0
	Fail times	0	0	0	0	0	0	100	100
	Average used time (ms)	508	364	363	394	1294	3835		
D	Success times	100	100	100	100	100	100	0	0
	Fail times	0	0	0	0	0	0	100	100
	Average used time (ms)	407	377	362	396	1142	3904		
E	Success times	100	100	100	100	100	100	0	0
	Fail times	0	0	0	0	0	0	100	100
	Average used time (ms)	1482	364	352	479	1209	3907		

Test result of Version 5 QR Code is shown as following table.

Vertical Point \ Horizontal Point		1	2	3	4	5	6	7	8
A	Success times	69	100	100	100	100	26	0	0
	Fail times	31	0	0	0	0	74	100	100
	Average used time (ms)	2358	393	379	1968	4072	3080		
B	Success times	100	100	100	100	100	62	0	0
	Fail times	0	0	0	0	0	38	100	100
	Average used time (ms)	439	367	357	1680	3425	2931		
C	Success times	100	100	100	100	100	67	0	0
	Fail times	0	0	0	0	0	3	100	100
	Average used time (ms)	627	317	371	1652	3852	3270		
D	Success times	100	100	100	100	100	47	0	0
	Fail times	0	0	0	0	0	53	100	100
	Average used time (ms)	410	363	401	1686	3637	3294		
E	Success times	100	100	100	100	100	95	0	0
	Fail times	0	0	0	0	0	5	100	100
	Average used time (ms)	1724	370	467	1670	3592	2892		

Test result of Version 6 QR Code is shown as following table.

Vertical Point \ Horizontal Point		1	2	3	4	5	6	7	8
A	Success times	0	100	100	100	48	0	0	0
	Fail times	100	0	0	0	52	100	100	100
	Average used time (ms)		413	1435	1263	3102			
B	Success times	100	100	100	100	64	0	0	0
	Fail times	0	0	0	0	36	100	100	100
	Average used time (ms)	1449	397	1583	2053	3868			
C	Success times	100	100	100	100	100	0	0	0
	Fail times	0	0	0	0	0	100	100	100
	Average used time (ms)	581	385	1614	1382	3744			
D	Success times	100	100	100	100	100	0	0	0
	Fail times	0	0	0	0	0	100	100	100
	Average used time (ms)	445	366	390	1621	3713			
E	Success times	100	100	100	100	97	0	0	0
	Fail times	0	0	0	0	3	100	100	100
	Average used time (ms)	1269	376	875	2043	3019			

In order to obtain a better success rate, based on the above test result, we advise that:

(1) Distance between QR Code and camera should be less than 40cm;

(2) Angle α in the following sketch should be less than $arctan\left(\frac{5cm}{10cm}\right) \approx 26°$.