Analisis decodificación QR Code

Autores:
Lázaro Pereira
Javier Cardozo
Santiago Báez

Docentes:
Andrés Aguirre
Nicolás Furquez

Contenido

Alcance del proyecto

El alcance del proyecto es implementar un plugin que reconozca códigos QR a partir de una camera. Dicho plugin debe ser implementado en su totalidad en código Python.

Gestión de riesgos

Los riesgos identificados para el proyecto se apoyará en el *Documento de riesgos del proyecto.*

Proceso técnico

Sistema operativo	Linux (Sugar)
Lenguaje de programación	Python 2.7

Análisis del problema

La solución al problema podemos dividirlo en dos fases totalmente independientes:

1- Capturar la imagen desde la cámara web y luego transformarla a una matriz de 0's y 1's (binarizar la imagen).
2- Analizar algoritmo de decodificación QR Code

1-Análisis del problema - Captura y binarización de imagen

La captura y binarización de la imagen del código QR se puede resumir a los siguientes pasos:

1- Realizar la captura de la imagen a través de una cámara.
2- Identificar las dimensiones de la matriz a binarizar.
3- Procesar la imagen y crear la matriz binaria a ser utilizada en la parte de decodificación.

Para todos los pasos de este problema se utilizará la librería PyCV, la interfaz proporcionada para las librerías de OpenCV en Python, en mayor o menor medida.

Captura de imagen

La captura de la imagen se deberá llevar a cabo en un equipo con el sistema operativo ya indicado. Además deberá poseer las librerías necesarias para garantizar el funcionamiento de OpenCV.
Para asegurar el correcto funcionamiento se realizará una implementación que tome la imagen de forma perpendicular con el fin de facilitar el procesamiento. En la sección de Trabajos a futuro se describen posibles mejoras del algoritmo que no son contempladas en este alcance pero pueden llegar a ser incluidas de ser suficiente el tiempo.

Obs.: Por más información sobre la interface PyCV:
https://pypi.python.org/pypi/pycv/0.2.1

Identificación de dimensiones

La dimensión del código puede variar de acuerdo a la versión en la que se basa. Esta dimensión es importante ya que es la que permite la separación de varios módulos contiguos del mismo valor.

Para identificar la dimensión es necesario determinar la distancia entre los patrones localizadores ubicados en tres esquinas de la imagen del código y la cantidad de módulos entre ellos. Una vez se haya obtenido esta información se podrá construir la matriz binaria necesaria para la parte de decodificación.

Procesamiento de imagen

Una vez determinada la dimensión, será necesario realizar un procesamiento progresivo mediante un fraccionamiento de la imagen donde se procese fila por fila el código y se construya la matriz binaria necesaria para el algoritmo de decodificación.

Trabajos a futuro

Con los procesos actuales la captura de la imagen se llevará a cabo perpendicularmente y donde el código sea lo único en la imagen. Posteriormente se podría realizar un procesamiento en el cual se filtre de la imagen objetos que no sean parte del código, así como realizar un acondicionamiento que permita interpretar el código como si se encontrase perpendicular a la cámara.

2-Análisis del problema – Analizar algoritmo de decodificación QR Code

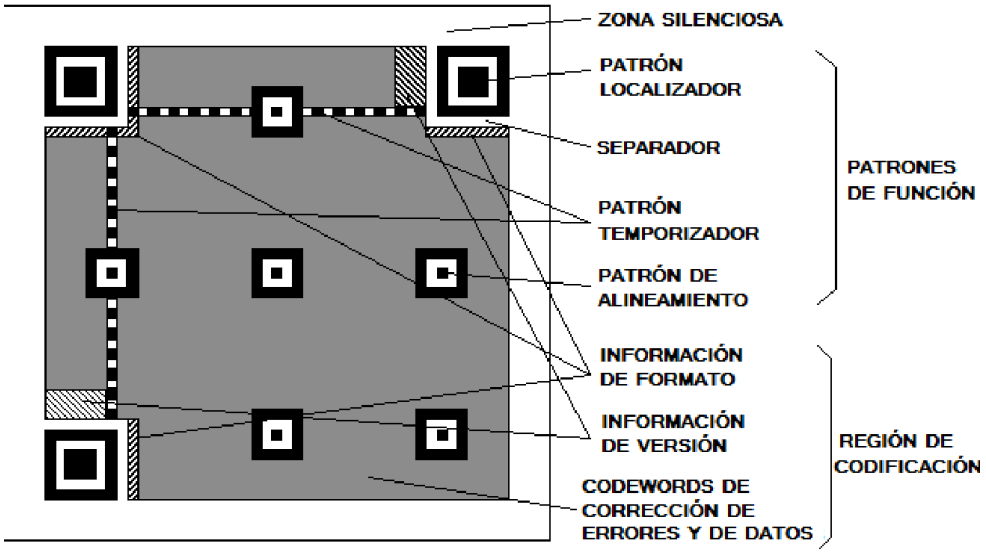
Podemos separar el problema de la decodificación en 5 pasos:

4- A partir de la matriz de 0's y 1's leer la información de formato, detectar y corregir posibles errores. Obtener el nivel de corrección de errores y el tipo de patrón de mascara de datos usado. Luego leer la orientación, patrones de temporizador, patrón de alineamiento.
5- Leer la región de codificación, en él podremos detectar la información de versión si está, la información del formato y los datos.
6- Pasar la máscara de datos a la matriz de unos y ceros, en la región de codificación, mediante una operación XOR, esto deshará la máscara.
7- Obtener los codewords de datos y de error de la región de codificación teniendo en cuenta su orden de colocación.
8- Detectar los posibles errores en los codewords de datos, utilizando los codewords de error y corregirlos.
9- Dividir los codewords corregidos en segmentos según los indicadores de modo y contadores de caracteres encontrados.

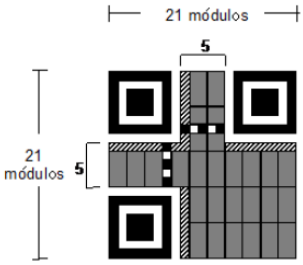
En primera instancia se explicara la estructura del código.

Hay 40 versiones de QR, la versión 1 tiene 21x21 módulos y la versión 40 tiene 177x177.

Aquí se indican como se estructura el QR



A modo de ejemplos se muestra la versión 1

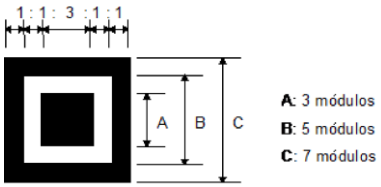


Versión 1

Patrones de localización

Para detectar la orientación se debe buscar los 3 patrones de localización, que están formados por cuadrados rellenados por 3x3 módulos negros, rodeado por un cuadrado de 5x5 módulos blancos que a su vez está rodeado por otro cuadrado de 7x7 con módulos negros.

Al momento de encontrar los 3 patrones de localización se puede ubicar la orientación que tiene el QR. La imagen a continuación indica cómo están formados los patrones de localización.



Una vez localizados los tres patrones, hay que determinar la orientación del QR. Para ello se analizan las posiciones de los centros de cada patrón y se detecta cuál de ellos es el patrón superior izquierda. El hecho de haber detectado los patrones, nos da una idea de cuál es el ancho de módulo en pixeles de la imagen.

Patrones temporizadores

Patrón de función que permite re sincronizar las coordenadas de mapeo del símbolo ante posibles distorsiones moderadas.
Los patrones temporizador son dos, uno vertical y otro horizontal. Están formados por una línea o columna de módulos blancos y negros alternados, comenzando y terminando en un módulo negro. El temporizador horizontal cruza la fila número 6 entre los separadores superiores y el vertical igual pero cruzando la columna 6.

Patrones de alineamiento

Los patrones de alineamiento están formados por un módulo negro, rodeado de un cuadrado de 3x3 módulos blancos que a su vez está rodeado por otro cuadrado de 5x5 módulos negros. Su número en el símbolo varía según la versión.

Versión

Una manera de identificar la versión es contando la cantidad de módulos que tiene a lo largo el QR.

Sabiendo la versión del símbolo, ya sabemos cuántos patrones de alineamiento posee el símbolo y su posición, en estas posiciones debemos buscar los patrones 11111, 10101 y el 10001.

Obs.: Las versiones anteriores a la 7 no tienen información de versión, la versión 1 no tiene ningún patrón de alineamiento. Todas tienen tres patrones localizadores, dos patrones temporizadores, tres separadores y la información de formato por duplicado.

Corrección de errores

Los códigos QR emplean codificación de errores basada en algoritmos de Reed-Solomon, generando un conjunto de codewords de corrección de errores (ECC, Error Correction Codewords) que se añaden a los de datos aportando redundancia.

Existen 4 niveles de corrección de errores en los símbolos QR:
-L (Low). Puede corregir hasta el 7% de los codewords de datos del símbolo.
-M (Medium). Puede corregir hasta el 15% de los codewords de datos del símbolo.
-Q (Quality). Puede corregir hasta el 25% de los codewords de datos del símbolo.
-H (High). Puede corregir hasta el 30% de los codewords de datos del símbolo.

Obs.: Por más información sobre algoritmo de Reed-Solomon
http://es.wikipedia.org/wiki/Reed-Solomon y http://en.wikipedia.org/wiki/BCH_code

Información de formato

La información de formato es una secuencia de 15 módulos (bits), de los que 5 contienen datos y los otros 10 se emplean para corrección de errores en los 5 primeros, mediante un código BCH (15,5). De los 5 bits de datos, los dos primeros indican el nivel de corrección de error usado (L = 01; M = 00; Q = 11; H = 10) y los otros tres indican el patrón de la máscara de datos usada. Una vez calculados los 15 bits se les debe aplicar mediante la operación lógica XOR la máscara “101010000010010”, para evitar una información de formato compuesta solo por bits a 0. La información de formato se ubica por duplicado (alrededor de los patrones de localización) en el símbolo QR, ya que su decodificación es esencial para la correcta decodificación del símbolo.

Obs.: Por más información sobre código BCH http://en.wikipedia.org/wiki/BCH_code

Datos

Para definir bien la región de codificación es necesario encontrar los patrones temporizadores. Estos patrones nacen y terminan en un borde de los patrones localizadores, con una separación de un módulo blanco; este módulo blanco pertenece a un patrón separador.

Los datos codificados, por su parte, se agrupan en conjuntos de 8, denominados codewords, que adoptan diversas formas según su ubicación en la estructura. La región de codificación es la región del símbolo no ocupada por patrones de función y sí por codewords de datos y de corrección de errores, así como por la información de formato y versión.

Se denomina modo a la forma de representar un conjunto de datos como una cadena de caracteres.

Modo	Indicador de modo
ECI	0111
Numérico	0001
Alfanumérico	0010
Byte (Binario)	0100
Kanji	1000
Estructura apilada	0011
FNC1	0101 (Primera posición)
	1001 (Segunda posición)
Terminador (fin del mensaje)	0000

Versión	Modo			
	Numérico	Alfanumérico	Byte	Kanji
1 -9	10	9	8	8
10 -26	12	11	16	10
27 -40	14	13	16	12

En los códigos QR hay diferentes modos diferentes de codificar la información:

1- Modo numérico. Dígitos (0-9). Densidad media de 10 bits para cada 3 caracteres.

2- Modo alfanumérico. 45 caracteres: 0-9, A-Z y otros 9 caracteres: espacio, $, %, :, ., *, +, -, /).
Densidad media de 11 bits para cada 2 caracteres.

3- Modo byte. Código binario según se define en JIS X0208. 8 bits por carácter.

4- Modo Kanji. Caracteres del alfabeto japonés según se define en Shift JIS. Densidad media de 13 bits para cada 2 caracteres.

5- Modo de estructuras apiladas. Para dividir la información en varios códigos QR relacionados.

6- Modo FNC1. Codificación de UCC/EAN (códigos de barras unidimensionales) o de cualquier otro estándar específico de la industria que esté aprobado por AIMI.

7-ECI (Extended Channel Interpretation, interpretación de canal extendido). Permite a los

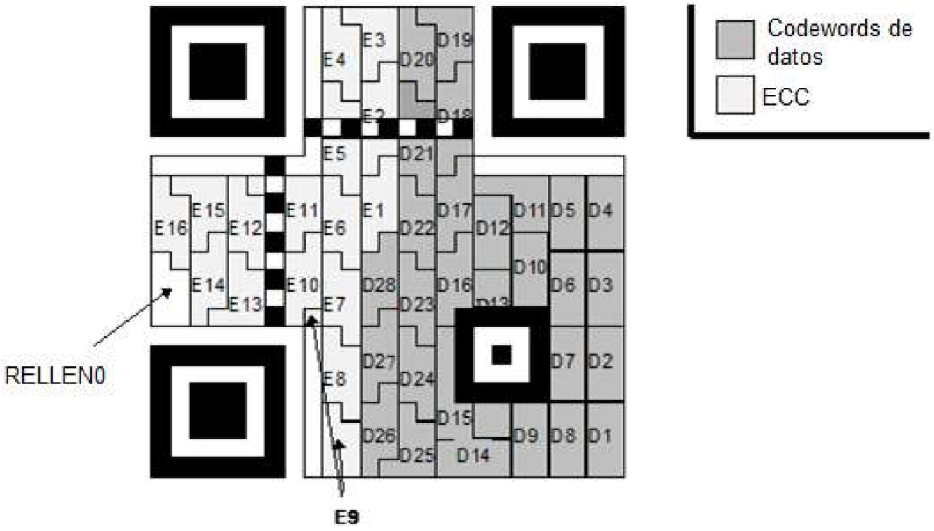
Versión	Módulos por lado	Módulos de patrones de función	Módulos de Información de formato y versión	Módulos en región de codificación	*Codewords* en región de codificación (datos y corrección de errores)	Bits restantes
1	21	202	31	208	26	0
2	25	235	31	359	44	7
3	29	243	31	567	70	7
4	33	251	31	807	100	7
5	37	259	31	1079	134	7
6	41	267	31	1383	172	7
7	45	390	67	1568	196	0
8	49	398	67	1936	242	0
9	53	406	67	2336	292	0
10	57	414	67	2768	346	0
11	61	422	67	3232	404	0
12	65	430	67	3728	466	0
13	69	438	67	4256	532	0
14	73	611	67	4651	581	3
15	77	619	67	5243	655	3
16	81	627	67	5867	733	3
17	85	635	67	6523	815	3
18	89	643	67	7211	901	3
19	93	651	67	7931	991	3
20	97	659	67	8683	1085	3
21	101	882	67	9252	1156	4
22	105	890	67	10068	1258	4
23	109	898	67	10916	1364	4
24	113	906	67	11796	1474	4
25	117	914	67	12708	1588	4
26	121	922	67	13652	1706	4
27	125	930	67	14628	1828	4
28	129	1203	67	15371	1921	3
29	133	1211	67	16411	2051	3
30	137	1219	67	17483	2185	3
31	141	1227	67	18587	2323	3
32	145	1235	67	19723	2465	3
33	149	1243	67	20891	2611	3
34	153	1251	67	22091	2761	3
35	157	1574	67	23008	2876	0
36	161	1582	67	24272	3034	0
37	165	1590	67	25568	3196	0
38	169	1598	67	26896	3362	0
39	173	1606	67	28256	3532	0
40	177	1614	67	29648	3706	0

flujos de datos de salida ser interpretados de forma diferente a los conjuntos de caracteres por defecto.

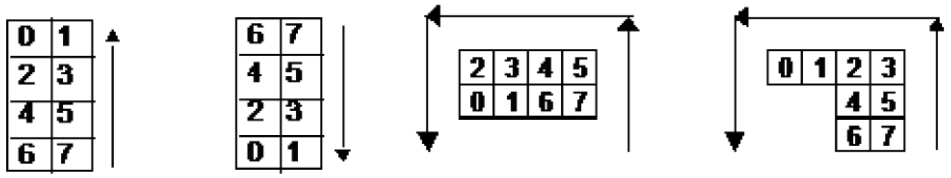
El número total de codewords en el mensaje viene determinado por el tamaño de la región de codificación, que es el tamaño del símbolo menos el de los patrones de función. Este número total se dividirá en codewords de datos y codewords de corrección de errores, más los codewords restantes si son necesarios para completar la estructura del símbolo. Para construir la secuencia de codewords en el símbolo, los codewords de datos deben dividirse para poderles aplicar el algoritmo de Reed-Solomon en un cierto número de bloques, determinado por la versión y el nivel de corrección de errores seleccionado. Para cada uno de estos bloques se calculan los correspondientes codewords de corrección de errores de acuerdo a los coeficientes de los polinomios establecidos por el algoritmo.

Colocación de Codewords en la matriz

Los codewords normalmente tendrán la forma de 2x4 módulos, empezando por la parte inferior derecha del símbolo, subiendo primero hacia arriba hasta llegar al patrón localizador, luego pasamos a la columna adyacente izquierda y bajamos hasta abajo, para luego volver a subir. Los codewords se van colocando en zigzag. En la siguiente figura se muestra como sería la colocación de codewords en un símbolo versión 2.



El orden de los bits de cada codeword varía cuando el codeword se coloca hacia arriba o hacia abajo:



Si la capacidad total del símbolo no se llena con los codewords, se rellena con 3,4 o 7 bits de relleno.

Máscaras de datos

La máscara se le aplica tanto a los codewords de datos como también a los de corrección de errores, mediante la operación XOR. Existen 8 patrones de mascara y se aplican todos y nos quedamos con el mejor resultado.

Los patrones posibles y su código para la información de formato son los siguientes:

Código de patrón de máscara de Datos	Fórmula
000	$(i + j) \bmod 2 = 0$
001	$i \bmod 2 = 0$
010	$j \bmod 3 = 0$
011	$(i + j) \bmod 3 = 0$
100	$((i \text{ div } 2) + (j \text{ div } 3)) \bmod 2 = 0$
101	$(i\,j) \bmod 2 + (i\,j) \bmod 3 = 0$
110	$((i\,j) \bmod 2 + (i\,j) \bmod 3) \bmod 2 = 0$
111	$((i+j) \bmod 2 + (i\,j) \bmod 3) \bmod 2 = 0$

Conclusiones

1- El presente algoritmo de decodificación si bien tiene algunas complejidades, no tiene limitaciones computacionales respecto al desarrollo exclusivo en Python.
2- Convertir una imagen a una matriz de bits 0's y 1's a priori tampoco se visualiza limitaciones computaciones para ser desarrollado en Python.