

Assignment 2 – Colour 2D barcodes

2D barcodes are becoming very popular. A well-known one is the QR code, which is common in mobile apps. There is a new type of barcode that uses colours to increase the depth of the encoding. Your task is to write a program in C or C++ using OpenCV that **reads the 2D barcodes below**. The 2D barcodes may be rotated, scaled and slightly distorted.

This new 2D barcode uses a simple encoding with 8 colours. The colours are black, blue, green, cyan, red, magenta, yellow, and white. These colours can represent three bit words. Notice that there is a relationship with the RGB colour space (e.g., black is 0 (0,0,0), blue is 1 (0,0,255), and green is 2 (0,255,0) etc).

There are 64 characters that can be encoded: the digits 0~9, the lower case a~z, the upper case A~Z, a full stop '.' and space. The encoding table can be found on Stream. The encoding of text is done through the 2D bar using the order Top-left towards Right-bottom (just like scanning an image). Every **two** squares of colours represents one of the 64 characters (3bits + 3bits). The maximum number of characters in one barcode is **1050**. The last square (bottom right) is left unused.

An example of encoding: char 'a' is 1. Therefore, its bit representation is 000001. Dividing this 6 bit group into two groups, we have 000 and 001. The first square should be black (0,0,0) for R=0, G=0 and B=0 and the second square should be blue (0,0,1) for R=0, G=0 and B=255. Each square block of colour represents 3 bits, and a pair of squares represent a single character.

An example of decoding: two squares red and yellow. Red is represented by (255,0,0) and yellow is (255,255,0) in RGB. So red is 100 and yellow is 110. Putting the two together, 100110 (38 in decimal) is 'L' in the table. Therefore, red/yellow is decoded as 'L'.

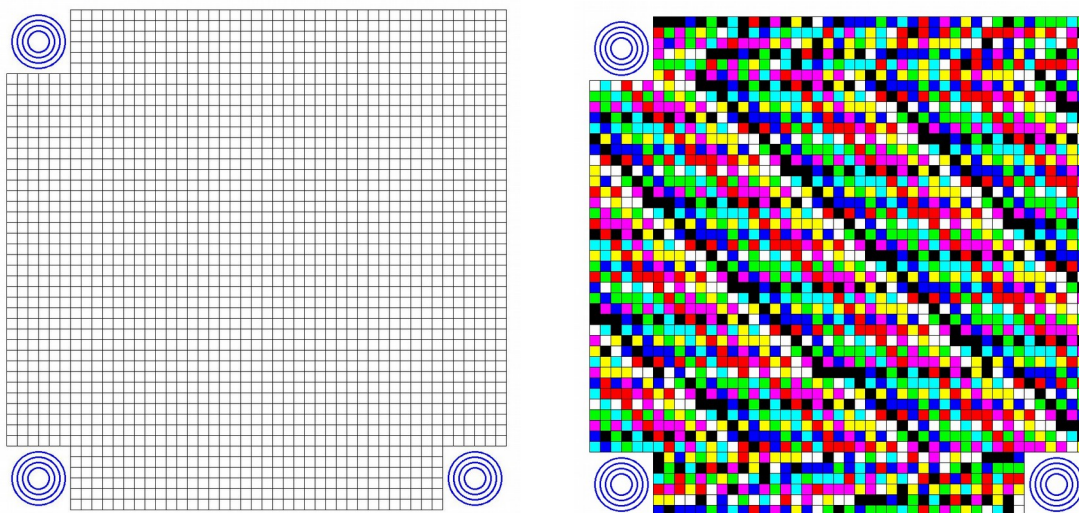


































































































































Figure 1. a) an empty 2D bar showing the concentric circles. b) an example of encoding all possible characters in order (see table 1 on the next page).

In figure 1 two examples of the 2D bar are shown. The concentric circles can be used for aligning the 2D bar when the image is rotated. The centres of the concentric circles can also be used to locate the squares properly when the image is scaled and rotated. The lines between the squares can also be used to rotate the 2D bar to an horizontal position. Both the lines and the concentric circles can be used concurrently to get a better accuracy, so the centre of each squares where the colours will be determined can be scanned properly.

Notes:

- 1) Initially, work with an upright static version of the barcodes. It is easier to debug decoding problems.
- 2) Use a combination of Hough transforms for straight lines and circles to re-orient the image in case the image is rotated or scaled.
- 3) Once you have the image properly positioned, start reading the colours from upper-left to down-right, reading two squares at a time. For example, A is encoded as 27 (011011) and G as 33 (100001). Therefore A is encoded with 011 (cyan) 011 (cyan) and G is encoded as 100 (red) 001 (blue).
- 4) Figure 1-b shows the following pattern: <space>abcd...zABCD...Z12..9<stop>. This is the same order in which the characters are encoded ('space' is 000000, '.' is 111111, 'a' is 000001 etc).
- 5) Remember that the **last square (bottom-right of the 2D bar) does not encode anything**, as it cannot pair up with any other square.
- 6) The encoding/decoding table is: (use the array available on Stream)

char	value	bits	colours	char	value	bits	colours	char	value	bits	colours	char	value	bits	colours
space	0	000000	 	p	16	010000	 	F	32	100000	 	V	48	110000	 
a	1	000001	 	q	17	010001	 	G	33	100001	 	X	49	110001	 
b	2	000010	 	r	18	010010	 	H	34	100010	 	Y	50	110010	 
c	3	000011	 	s	19	010011	 	I	35	100011	 	W	51	110011	 
d	4	000100	 	t	20	010100	 	J	36	100100	 	Z	52	110100	 
e	5	000101	 	u	21	010101	 	K	37	100101	 	0	53	110101	 
f	6	000110	 	v	22	010110	 	L	38	100110	 	1	54	110110	 
g	7	000111	 	x	23	010111	 	M	39	100111	 	2	55	110111	 
h	8	001000	 	y	24	011000	 	N	40	101000	 	3	56	111000	 
i	9	001001	 	w	25	011001	 	O	41	101001	 	4	57	111001	 
j	10	001010	 	z	26	011010	 	P	42	101010	 	5	58	111010	 
k	11	001011	 	A	27	011011	 	Q	43	101011	 	6	59	111011	 
l	12	001100	 	B	28	011100	 	R	44	101100	 	7	60	111100	 
m	13	001101	 	C	29	011101	 	S	45	101101	 	8	61	111101	 
n	14	001110	 	D	30	011110	 	T	46	101110	 	9	62	111110	 
o	15	001111	 	E	31	011111	 	U	47	101111	 	.	63	111111	 

7) the colours present in the barcode are:

black (0,0,0), red (255,0,0), green (0,255,0), blue (0,0,255), white (255,255,255), cyan (0,255,255), magenta (255,0,255) and yellow (255,255,0). Notice that the eight colours give all the possible states for a 3 bit word.

8) There are many ways to re-orient, scale and align the image, so each colour can be extracted. For example, one could start with the Hough for lines and just rotate the image, so the lines are horizontal/vertical. However, this could leave the image rotated to 90 degrees, or even upside-down. One could then use the concentric circles to rotate the image again, this time by a multiple of 90 degrees. The image could be out of centre and in an arbitrary scale, so limits of the 2D bar can be used to determine a standard scale and centralise the image. The concentric circles can also be used for the same purpose.

The assignment is worth **10 marks**.

5 marks for correctly decoding static *upright* test images.

5 marks for correctly decoding *rotated, noisy, scaled* and slightly *distorted* images.

3 bonus marks: for *dynamic detection and interpretation* (i.e., using the camera)

Submit your assignment on Stream by the due date. Late assignments lose 10% per day after the due date (unless you justify the delay with the lecturer). **DUE DATE: 12/June/2020**