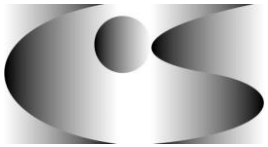
May 15, 2012

COMPUTER SCIENCE DEPARTMENT
Technion - Israel Institute of Technology



AR on QR

Final Presentation

Augmented Reality above QR Code

Students: Aviya Levy, Shany Shmuely
Supervisor: Dan Vardi, Elster Constantine

Agenda

- Project overview
- Application Demonstration (site)
- Software state machine
- Algorithms overview
  - QR Code detection
    - Previous Attempts
    - Final Algorithm
  - QR Code 2D transformation
    - Corners detection
    - Homography
- Future work (site)

Project overview

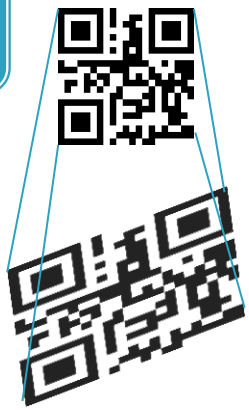
- Implement an application which allows smartphone users to receive advertisements from android smartphone's camera in augmented reality environment.
- The method works by recognizing QR codes and display relevant advertisement.

Algorithm state machine

Frame+AR model rendering

Finder Patterns detection

Corners Detection



Extract the QR Code transformation

Decoding By Zxing

Selecting a relevant model

Algorithms Overview

Finder Patterns Detection

Finder Patterns detection

Corners Detection

Extract the QR Code transformation

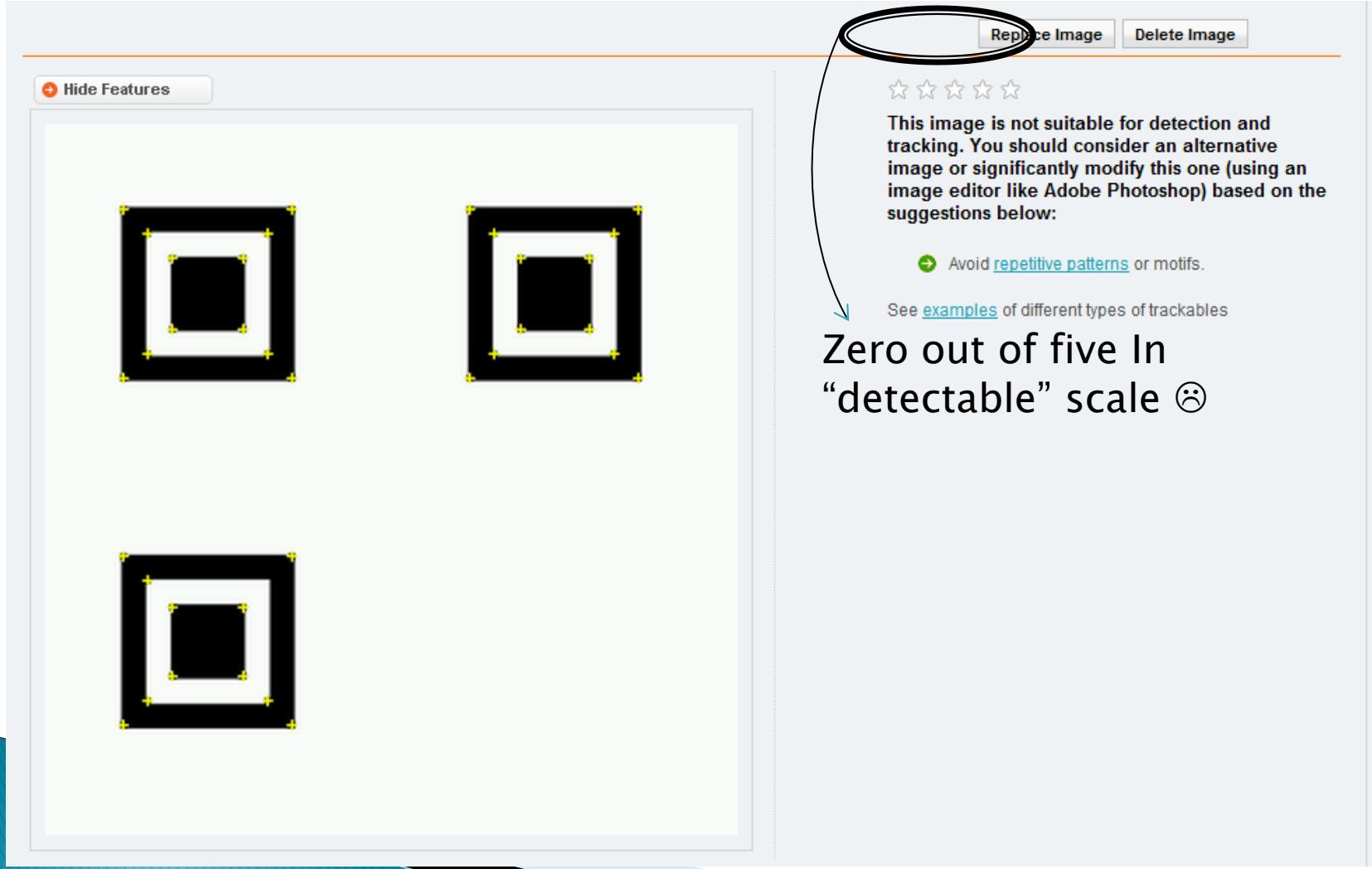
Decoding By Zxing

Selecting a relevant model

Frame+AR model rendering

Trial #0- Vuforia

Vuforia supplies a Tool that check if an Image is suitable for detection, the result:



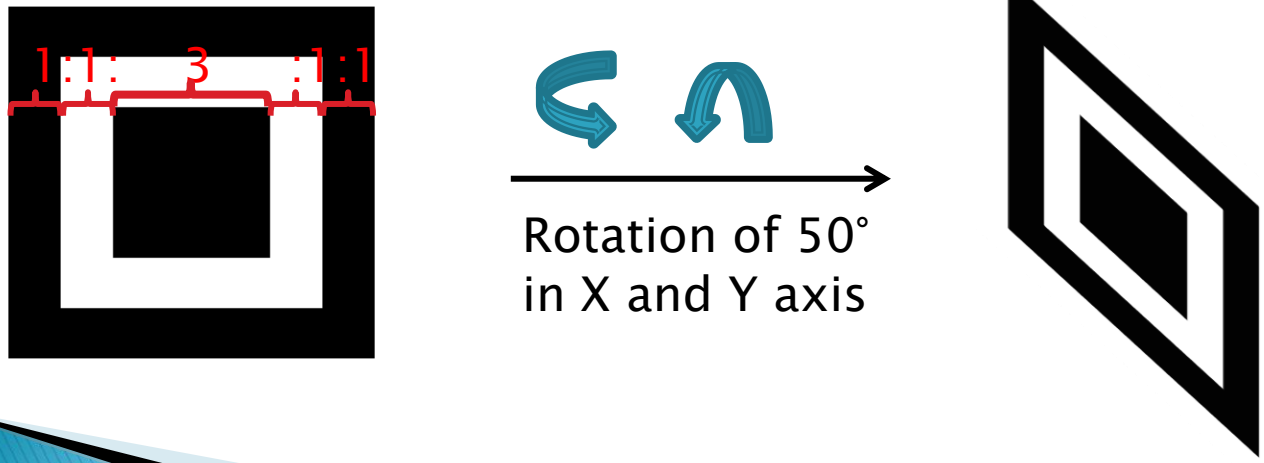
Zero out of five In "detectable" scale ☹

Trial #1 – ZXing detection

ZXing library contain a Detector class that detect a QR Code in an image.

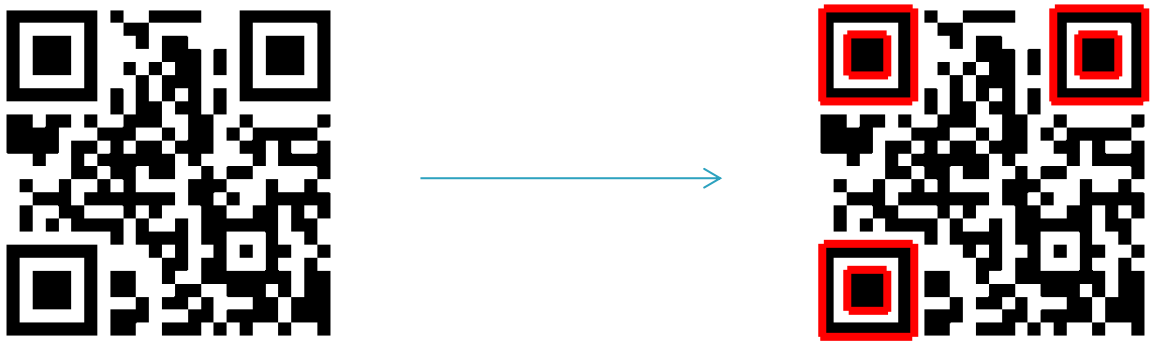
Detect the QR Code finder pattern by searching for black/white/black/white/black modules in 1:1:3:1:1 ratio in a row, then search for similar patterns in the follows rows.

☹

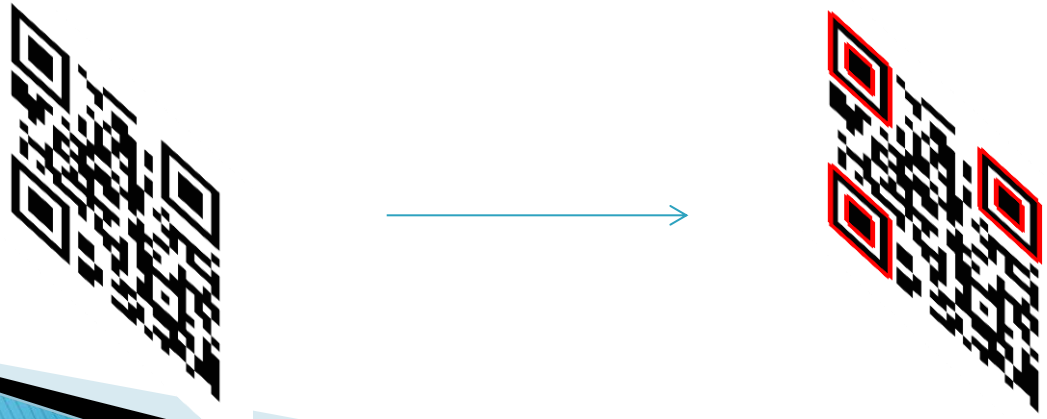


Trial #2- Hough Lines algorithm

Hough Lines algorithm which detects straight lines in a frame, then it will be easy searching the three finder patterns.

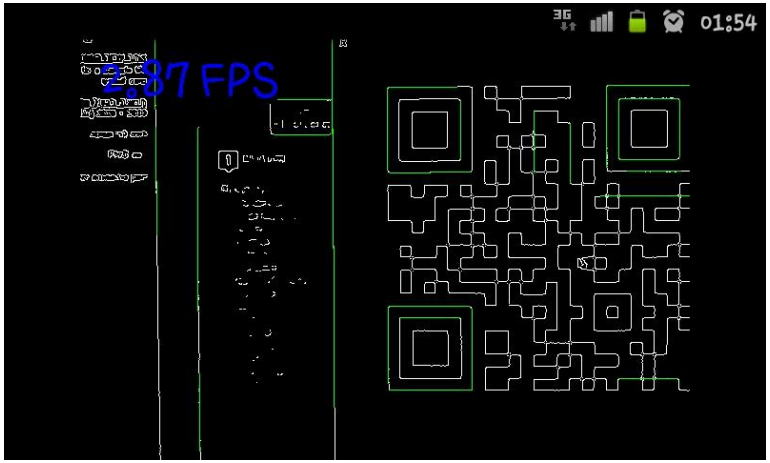


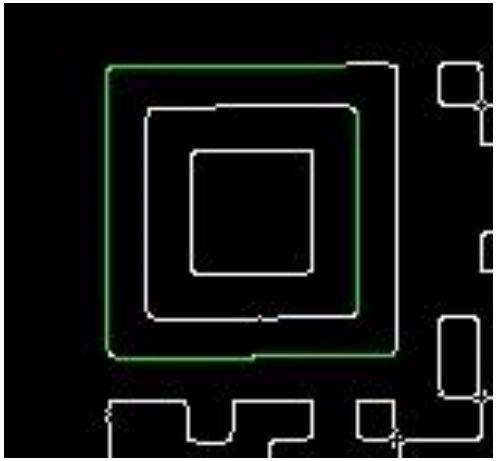
Hough Lines algorithm works also for rotated QR Code ☺



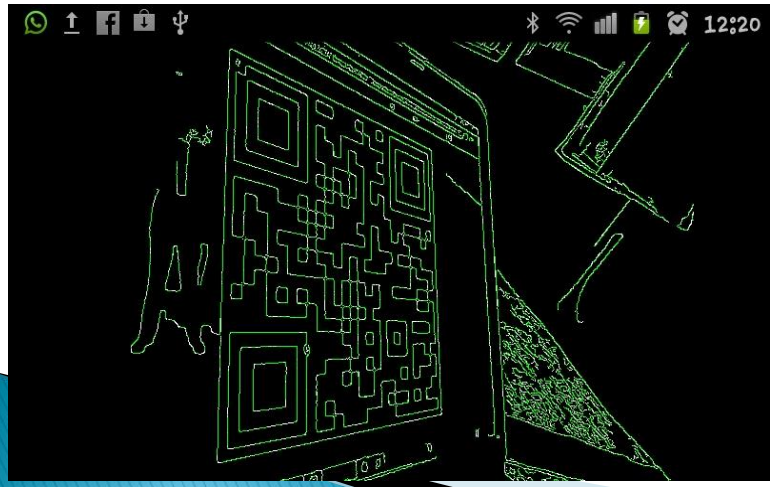
The algorithm extract edges from the Image and processes them to detects lines features.

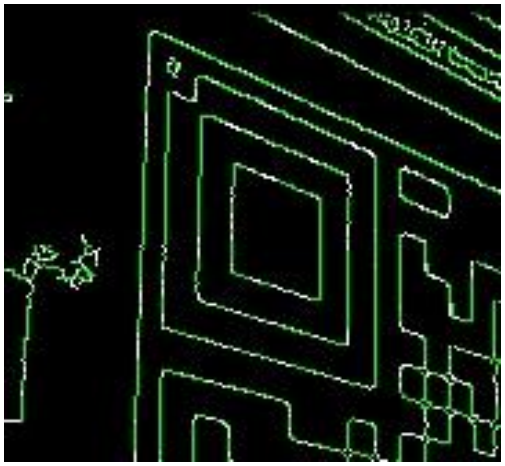
The results:





After lowering the threshold

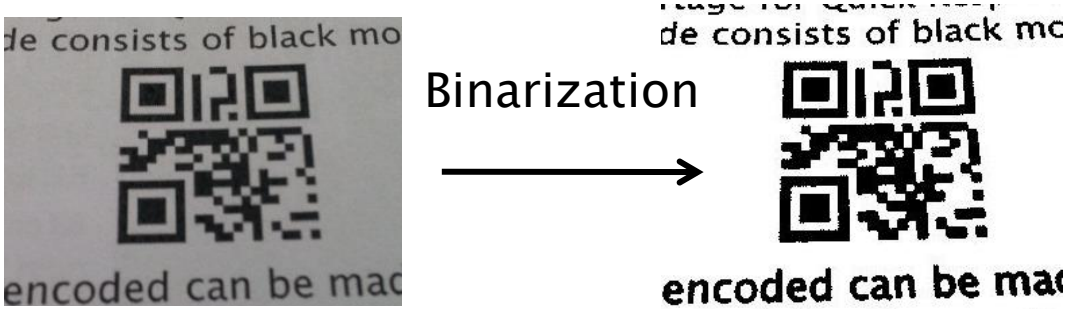




After lowering the threshold the result is detection of too much lines, and also not the desired.
the outcome can be bad performance and in the worst cause also wrong detection. ☹

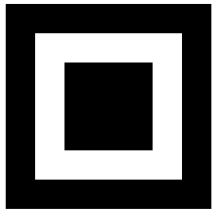
Trial #3- Connected components processing algorithm

- Connected component (CC) is a region in a binary digital image.



- Every CC has many kinds of properties, Such as area, shape, center of mass, moment
  etc.

- Finding 2 (or 3) CC, locating one inside the other, with same CM, or same moment will lead to location of one finder pattern.

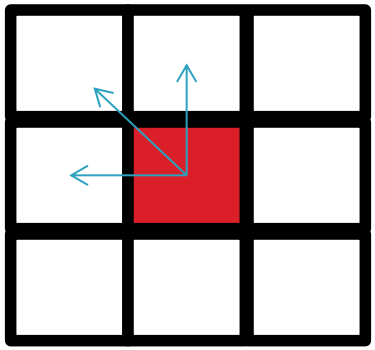


Connected Components Detection Algorithm Implementation

Based On Two pass algorithm.

We use a raster scan on the binary image, and check connectivity for each pixel.

Connectivity checks are carried out by checking the labels of pixels that are East ,North-East, North of the current pixel (assuming 8-connectivity).



First Pass Data Structure:

Bitmap – 2D bit array representing the binary frame.

Label – 2D array in size of Bitmap.size(), save the label of every pixel.

UF-union find data structure.

```
Bool connectedToLeft   = (Bitmap.getLeft(i,j) == Bitmap.get(i,j));
Bool connectedToLeftUp = (Bitmap.getUpLeft(i,j) == Bitmap.get(i,j));
Bool connectedToUp     = (Bitmap.getUp(i,j) == Bitmap.get(i,j));
```

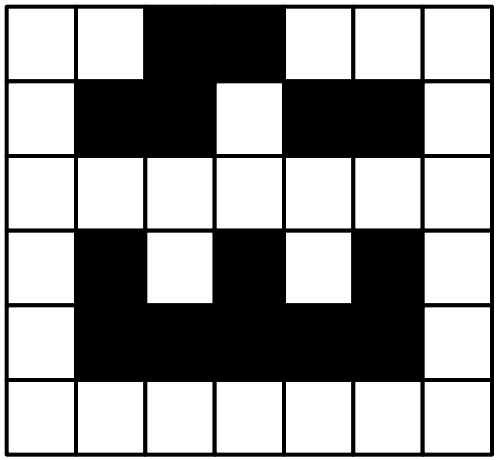
```
for( int i=0; i<Bitmap. Width, i++)
for( int j=0; j<Bitmap. Height, j++)
{
    if( connectedToLeft )
    {
            Label.Set(i, j, Label.getLeft(i,j));
            if( connectedToUp &&  Label.getLeft(i,j)!=Label.getUp(i,j))
            {
                    UF.Union( Label.getLeft(i,j) , Label.getUp(i,j) );
            }

            if( connectedToLeftUp &&  Label.getLeft(i,j)!=Label.getLeftUp(i,j))
            {
                    UF.Union( Label.getLeft(i,j) , Label.getUpLeft(i,j) );
            }

    }else if( connectedToLeftUp )
    {
            Label.Set(i,j,Label.getUpLeft(i,j));
            if( connectedToUp &&  Label.getUpLeft(i,j)!=Label.getUp(i,j))
            {
                    UF.Union( Label.getUpLeft(i,j) , Label.getUp(i,j) );
            }

    } else if( connectedToUp )
    {
            Label.Set(i, j, Label.getUpLeft(i,j) );

    } else
    {
            Label.Set(i, j , Label.NewLabel());
    }
}
```

Second Pass:

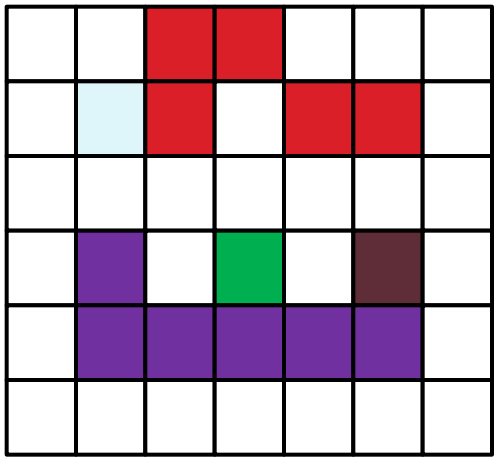
```
for( int i=0; i<Bitmap. Width, i++)
for( int j=0; j<Bitmap. Height, j++)
{
        Label.Set(i, j , UF.Find(Label.get(i,j)));
}
```

This is the bottle neck of runtime – takes 40%–50% of total runtime.

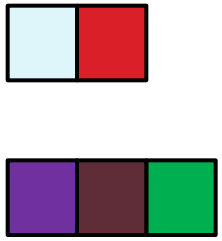
Binary Image:



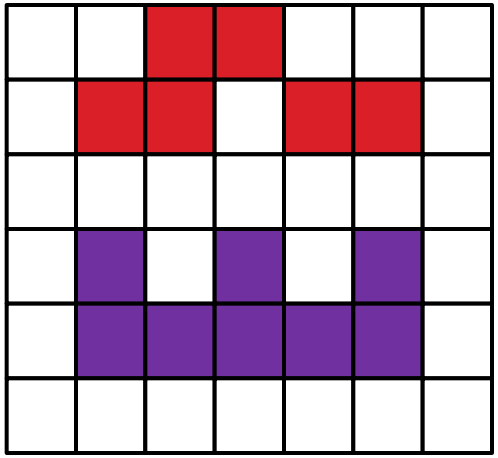
First Pass:



UF:



Second Pass:



QR Code Corners Detection

Frame+AR model rendering

Finder Patterns detection

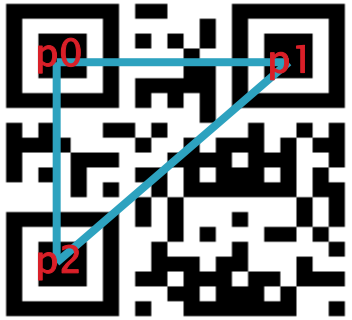
Corners Detection

Extract the QR Code transformation

Decoding By Zxing

Selecting a relevant model

Extracting QR Code corners
Step 1 – Ordering the 3 points



- The algorithm
  - Calculate 3 angles of the Triangle.
  - Select the vertex with the largest angle to be p0
  - Set p1 and p2 arbitrarily.
  - If cross(p1–p0,p2–p0).getZ() > 0, swap p1 and p2

- The angle of P0 mostly the largest one but not always:

Extracting QR Code corners

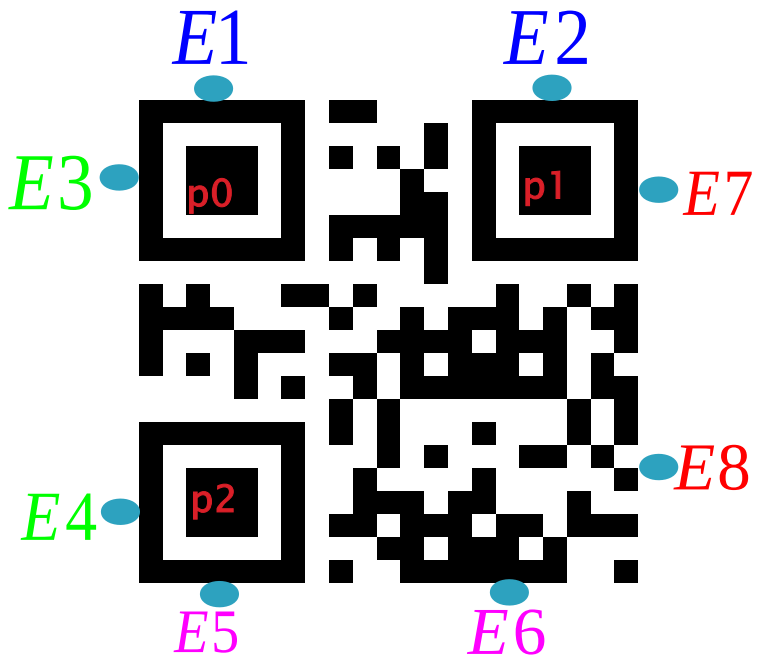
Step 2 – find 8 points on the QR Code frame:

- Find the blue and the green lines

$$\left.\begin{array}{l} blueLine: \; a_1 x + b_1 y + c_1 = 0 \\ (a_1, b_1, c_1) \cdot (E1_x, E1_y, 1) = 0 \\ (a_1, b_1, c_1) \cdot (E2_x, E2_y, 1) = 0 \end{array}\right\} \Rightarrow$$

$$(E1_x, E1_y, 1) \times (E2_x, E2_y, 1) = (a_1, b_1, c_1)$$

$$greenLine: \; (E3_x, E3_y, 1) \times (E4_x, E4_y, 1) = (a_2, b_2, c_2)$$



- Let C0 the upper left corner, lies on blue line and also on the green line

$$(C0_x, C0_y, 1) \cdot (a_1, b_1, c_1) = 0$$

$$(C0_x, C0_y, 1) \cdot (a_2, b_2, c_2) = 0$$

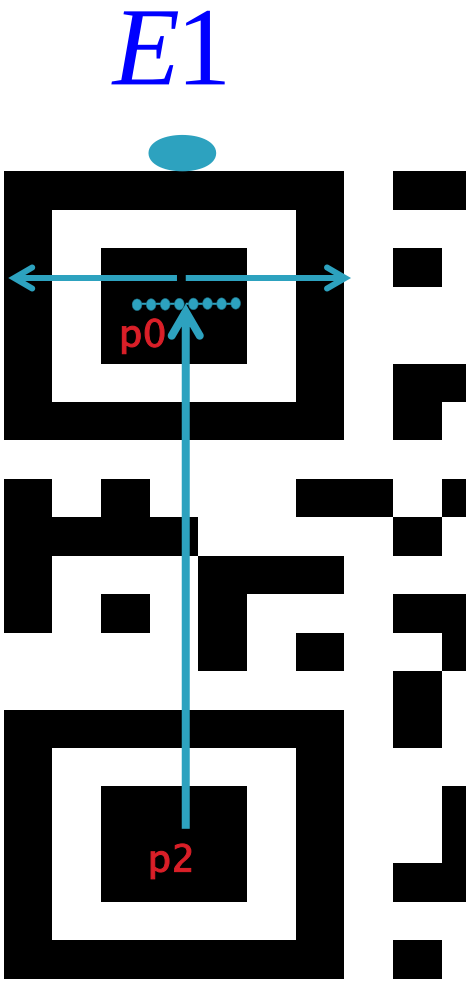
$$(a_1, b_1, c_1) \times (a_2, b_2, c_2) = (x, y, z)$$

$$(x, y, z) / z = (C0_x, C0_y, 1)$$

Extracting QR Code corners

Step 2.1 – find 6 points on the QR Code frame:

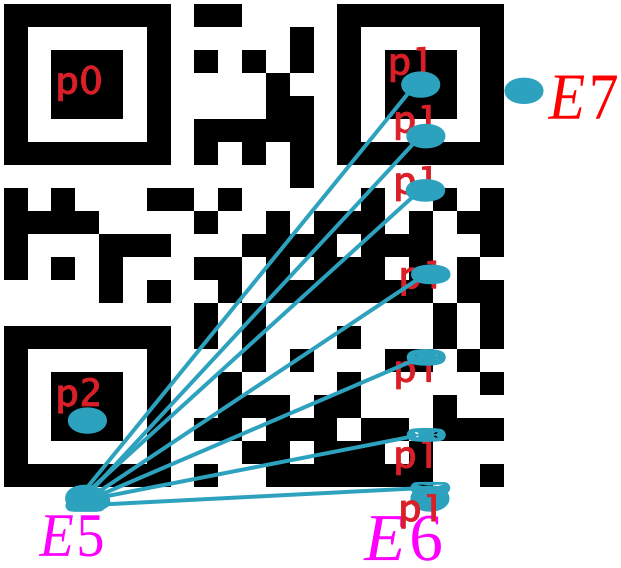
- The algorithm
  - Let N be the vector perpendicular to V02
  - Sample pixels in directions N and –N starting from P0
  - While the sampled line average color is bigger then 0.2:
    - Shift the sampled line in direction V02
  - While the sampled line average color is lower then 0.8:
    - Shift the sampled line in direction V02
  - While the sampled line average color is bigger then 0.2:
    - Shift the sampled line in direction V02
  - Return the middle point on last sampled line



Extracting QR Code corners

Step 2.2 – find 2 additional points on the QR Code frame:

- The algorithm
  - Sample The line between P1and E5
  - While the sampled line average color is bigger then 0.2:
    - Shift p1 in direction V20
    - Sample The line between P1 and P2
  - Set E6 to the last point on the sampled line
  - Do the same operation to find E8



Extract the transformation

Finder Patterns detection

Corners Detection

Extract the transformation

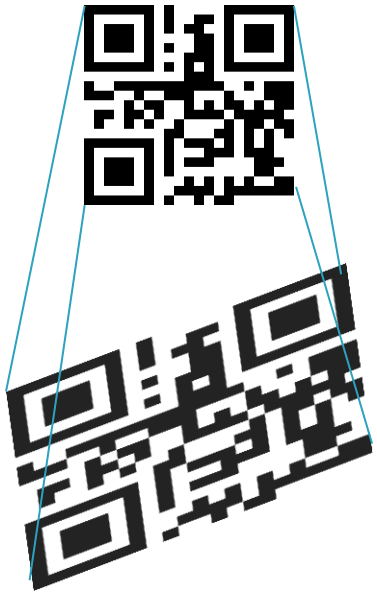
Decoding By Zxing

Selecting a relevant model

Frame+AR model rendering

Homography

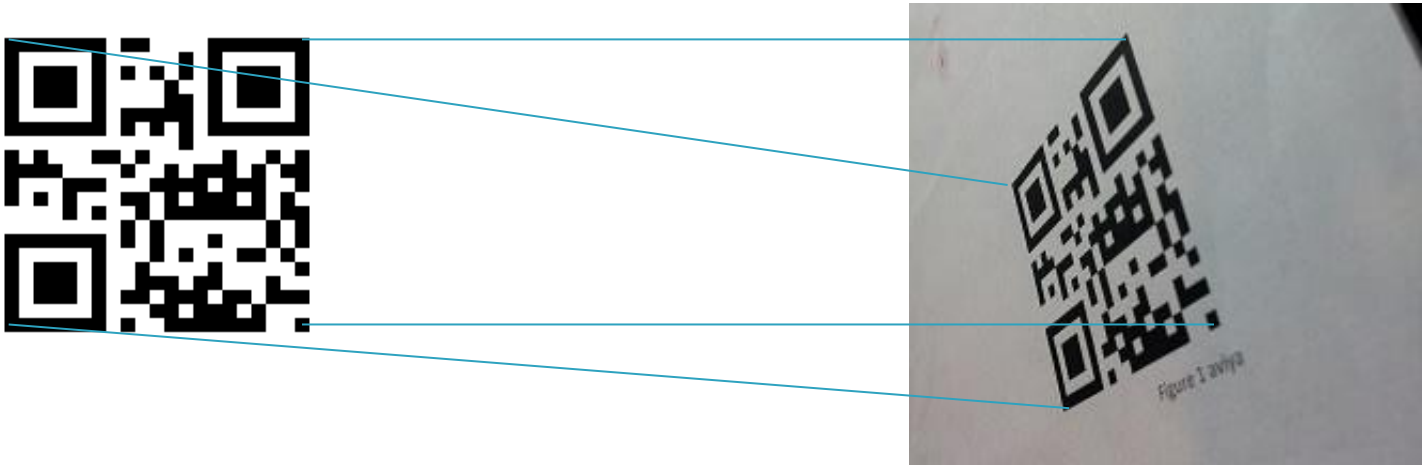
- It describes what happens to the perceived positions of observed planar objects when the point of view of the observer changes
- Let H be the homographic Matrix between QR1, QR2 and (x,y,1) a vector such that (x,y) is a pixels in a QR1 and let(x',y',z')=(x,y,1)*H so (x'/z', y'/z') is the position of the same pixel in QR2.
- With this information we can achieve two goals
  - Reconstruction of the original QR code.
  - Transformation of the planar object need to be displayed on the QR Code.



Extracting Homographic matrix

- We calculate the homography using OpenCv Library.
- The method requires at least 4 corresponded pairs of points.

Extracting Homographic matrix- Result:



QR Code Decoding (Zxing)

Finder Patterns detection

Corners Detection

Extract the QR Code transformation

Decoding By Zxing

Selecting a relevant model

Frame+AR model rendering

Select a relevant model

Frame+AR model rendering

Finder Patterns detection

Corners Detection

Extract the QR Code transformation

Decoding By Zxing

Select a relevant model

Frame Rendering

Finder Patterns detection

Corners Detection

Extract the QR Code transformation

Decoding By Zxing

Select a relevant model

Frame+AR model rendering

Restrains

(a) 10 alphanumeric characters (b) 100 alphanumeric characters

(c) 500 alphanumeric characters (d) 1000 alphanumeric characters

Fig. 5. Different versions of QR code symbols.

Restrains

- In order to get the binary image, Threshold algorithm is applied on the original image.
- The threshold is constant value [0,255].
- Problem- different scenes have different luminance level, one constant threshold can cause very dark/bright binary frame.
- The solution – adaptive threshold
  Coloring pixels according to its neighborhoods luminance level.
- This algorithm is expensive in terms of runtime.

Future work (site)