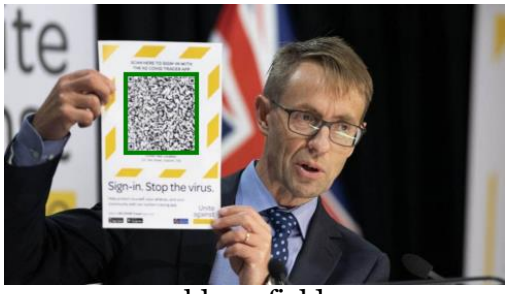
COMPSCI 373 Image Processing Assignment Report

Below are the result of executing my initial code which has successfully extracted the QR code in poster1small:

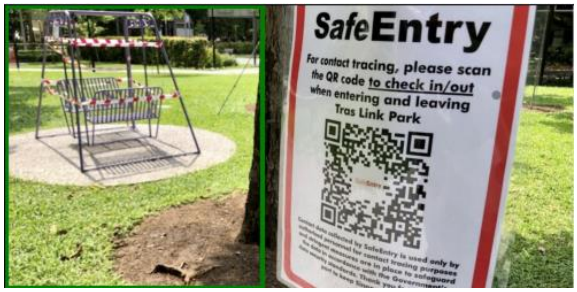
poster1small

bch



bloomfield

connecticut



playground

poster1smallrotated

shanghai

As seen from the output images above, regardless of the rotated QR codes, the drawn green box for image “bch” and “playground” are very far off. I speculate that this was largely caused by the fact that after multiple morphological closing operations, the QR code is no longer the largest connected component. I originally used the below code for the morphological closing operations, i.e. three dilations followed by three erosions.

```
# step 7
dilated = threshold
for i in range(3):
    dilated = computeDilation8Nbh3x3FlatSE(dilated, image_width, image_height)
eroded = dilated
for i in range(3):
    eroded = computeErosion8Nbh3x3FlatSE(eroded, image_width, image_height)
```

After adjusting the code so that the operation only perform one dilation and one erosion:

```
# step 7
dilated = computeDilation8Nbh3x3FlatSE(threshold, image_width, image_height)
eroded = computeErosion8Nbh3x3FlatSE(dilated, image_width, image_height)
```

I successfully extracted the QR code in “bch”.

poster1smallrotated, shanghai

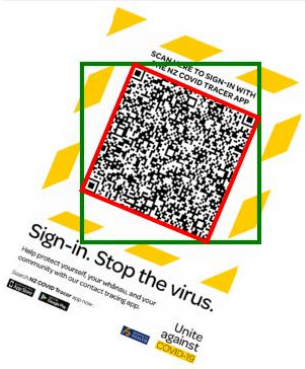
As seen above, my original algorithm draws a upright square around the rotated QR code. I notice that the rectangle correctly "bounds" the rotated QR code, which means that I already know half of the vertex coordinate for each of the 4 end points of the rotated QR code. I wrote the following method to compute the other vertex coordinate of the end points:

```
def computeEdgePointsForRotated(largestComponent, minw, maxw, minh, maxh, image_width, image_height):

    for i in range(0, image_height):
        for j in range(0, image_width):
            if largestComponent[i][j] == 255:
                if j == minw:
                    pointA = (j, i)
                if i == minh:
                    pointB = (j, i)
                if j == maxw:
                    pointC = (j, i)
                if i == maxh:
                    pointD = (j, i)

    return (pointA, pointB, pointC, pointD)
```

By passing the vertex coordinates of the four end points into a polygon,

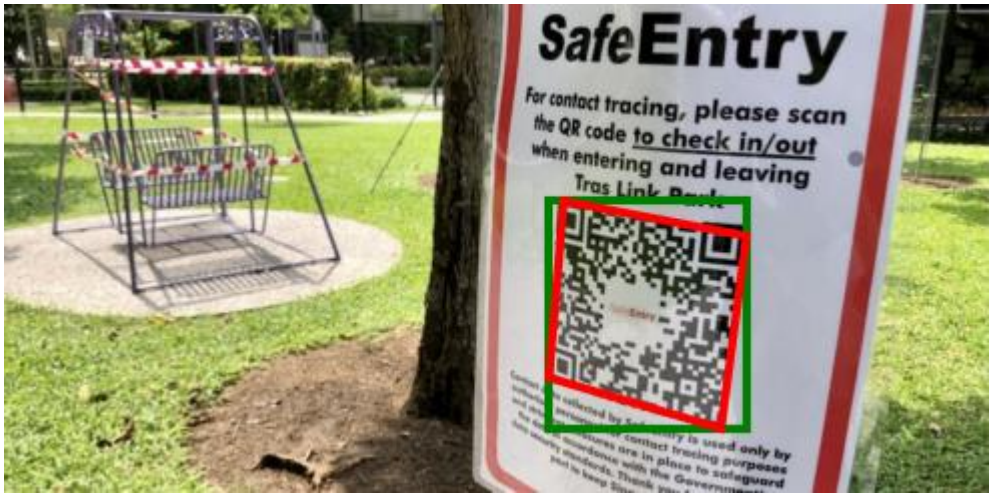
```
polygon = Polygon((pointA, pointB, pointC, pointD), linewidth=3, edgecolor='red', facecolor='none')
axes.add_patch(polygon)
```

I was able to draw the polygon around rotated QR code successfully.



However, the algorithm I used for rotated QR code and upright QR code are different. Ideally there should be a single algorithm that can extract QR code for any sort of image, which is an area I can improve my code on.

For "playground", as there is a lot of colours in the image such that the grey value of the background swing was very similar to that of the QR code, I could not initially extract the QR code. By adjusting the number of time for smoothing and threshold value, I was able to extract the QR code for "playground".


SafeEntry
For contact tracing, please scan
the QR code to check in/out
when entering and leaving