Image Processing Project Report

Introduction

The aim of the project was to implement a function using MATLAB to read the colour pattern from sets of real photos and simulated images. The problem involves noise removal from the image, crop and resize the image and/or fixing the rotation and projection of the image in order to read the colour pattern in the right order, correction of the white balance and increase of the colour saturation in order for the program to accurately differentiate colours, and implementation of a function that will take colour values as input and output the names of the colour.

Approach

In the image, there are four circles indicating the corners of the image, and two rectangles showing where the top left corner is.

The general idea of the approach is to find the circles and transform them back to the four corners of the image. Then by finding the circle that is closest to the two rectangles, the program can work out which corner is meant to be the top left corner, thus the degree it needs to rotate in order to recover the image.

The program is separated into different sections:

1. Section one reads the target image and one of the original images. The original image is needed to provide information about the size of the original images, so that in later section the program knows the size of the output view while recovering the image.

2. In section two, white balance [1] is performed to fix some of the real photos with unrealistic colour casts especially IMAG0042.

3. Section three will set any pixel with RGB value greater than certain value to white. This is to fix the problem where some of the images' white background become less white after white balance especially org_1.

4. Section four’s aim is to create an instance of the image that is binary, so that only the four circles, two rectangles, and the 4x4 grid will be left behind. Soon realized that if colours within the grid is dark enough, the binary function will convert them to black instead of white. To overcome this problem, section four reduces the colour saturation of the image then apply the binary function.

5. Section five will reduce the effect of salt and pepper noise by applying averaging filter and then Wiener filter [2]. But then it is too blurry for further action, so the binary function is applied again to sharpen the image.

6. Section six dilates the image using linear structuring elements, both horizontally and vertically. This aims at removing bits of the 4x4 grid, so that the lines of the grid aren’t touching the circles or rectangles.

7. Section seven dilates the image using disk-shaped structuring element aiming to remove some of the remaining noise and to make sure nothing is touching the circles or rectangles. This is important because if a circle or rectangle is connected to some other regions, then it won’t be consider as a circle or rectangle in later section of the program.

8. Section eight erodes the image using disk-shaped structuring element aiming to reconstruct any circles that are broken due to dilation. This is especially useful on noisy images because their circles are usually shattered after dilation.

9. Section nine computes the complement of the image, reversing the black and white. This is required because only white pixels are considered as part of a region, and if the program needs to find the regions for circles and rectangles, then the regions must be white.

10. Section ten remove regions that are too small or too big to be circles or rectangles. Note that from section four to ten, all actions are applied to a separate instance of the image only to find the correct transformation to recover the target image.

11. Section eleven crops the image so that all black pixels will be removed from the sides. This will improve accuracy when finding the closest circle for each corner in later section. The crop is applied to the target image as well to ensure the transformation can be applied correctly in later section.

12. Section twelve detects the circles and rectangles by firstly gets all regions' properties [3], and then gets the ten biggest regions assuming these have the highest chance of being a circle or a rectangle. Then out of these ten regions, gets the six regions that are furthest away from the average of regions' centroids. The assumption is that the circles and the rectangles are always at the edge of the image, so the remaining six regions must be the four circles and the two rectangles. The program then sort the six regions by eccentricity, the two with highest eccentricity will be considered as rectangles and the rest will be circles. In addition, the program finds the circle that is closest to the two rectangles, this information will be used in the next section to determine the amount of rotation needed to recover the image.

13. Section thirteen recovers the image by finding the closest circle for each corner, then map the circles to the corresponding corner. This will stretch the grid back to original shape and the circles will be at one of the four corners. However, this does not guarantee the image is correctly rotated. While finding the closest circle for each corners, the program also compare the centroid of the closest circle and the circle that is closest to the two rectangles. If there is a match of the two centroids, the program knows they are in fact the same circle, this also indicates which corner the two rectangles are closest to. This will give enough information to determine the amount of rotation needed to recover the image.

14. After the target image is recovered, in section fourteen the program blurs the image using Gaussian filter [4], and then increases the colour saturation [5], and then blurs the image again using Gaussian filter. It is necessary to remove some of the noises by blurring the image before increasing the saturation, otherwise the noises will be amplified. It is proven by experiment that increasing the saturation will help identifying the colours; for example before the saturation increase, some lighter red that is almost orange will be classified as yellow, and some lighter blue will be classified as white. The second Gaussian blur is applied to ensure the effect of noise is reduced to minimal.

15. In section fifteen, the program reads the middle pixel of each cell in the 4x4 grid and gets their RGB values. Compare each cell's RGB value to red, green, blue, yellow, and white individually, and find out which colour it is closest to. The comparison is done by calculating the Euclidean distance between the RGB value and the five colours in 3D space as shown in Figure 1.

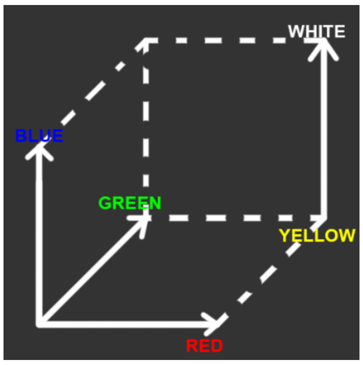


Figure 1. The Euclidean distances between the cell's RGB value and the five colours is calculated to determine the closest colour.

Result

As shown in Appendix 1, 27 out of 40 images were solved in which two are real photos.

The reasons for not solving are listed below:

- proj_1, proj2_1, proj2_4 are ridiculously projected and really hard to solve.
- In proj_4, the two circles at the top are too small and got removed after dilation.
- proj1_3 is not solved because one of the circles got stretched and has greater eccentricity than one of the rectangles. Therefore in section twelve, the program fails to distinguish the circles and rectangles and the rest of the program fails.
- Most of the real photos are not solved because of the noises around the target colour pattern and the assumption of the algorithm that the circles and rectangles are always at the edge of the image. The assumption of bigger region has higher chance of being circle and rectangle is also proven to be false in most of the real photos. Since most of them failed to find the correct circles and rectangles, the rest of the program fails.
- Some of the real photos have poor lighter and the shadows make them hard to be solved.
- IMAG0041 is successfully recovered, but it became so bright after white balance and section three, causing all the red cells being classified as yellow.

- On top of all the difficulties, IMAG0044 also has motion blur applied which makes it even harder to solve.

Evaluation and improvement suggestion

The program is good as it successfully solved 67.5% of the given images. However, in section 3, 4, 5, 6, 7, 8, 10, 12, 14, there are variables that are hard-coded to control different functions meaning that the program is doing well simply because it is fine-tuned and maybe over fitted. If this program is applied to a set of new images, there is no guarantee that it will perform well.

In terms of future improvement, one idea is to use the region's curvature to distinguish between circle and rectangle instead of rely on region's eccentricity.

Rather than focusing on circle detection, an alternative will be to detect lines using the Hough transform function [6]. By finding the line intercepts, ideally it will be able to locate the grid pattern in any given image, possibly bypass lots of difficulties regarding regions and noises around the target matrix.

The white balance function can be improved so that it won't brighten up the image too much.

Further research is needed to solve the problems of shadow and motion blur [7]. And since these problems may not appear on every images, it is hard to have one linear function that can solve all kind of images with different kinds of problem.

References

[1] K. Taylor, “Color-balance demo with GPU computing,” MathWorks, 2013. [Online]. Available: http://uk.mathworks.com/matlabcentral/fileexchange/41089-color-balance-demo-with-gpu-computing/content/GPUDemoCode/whitebalance.m. [Accessed 21 4 2016].

[2] MathWorks, “Noise Removal,” MathWorks, 2016. [Online]. Available: http://uk.mathworks.com/help/images/noise-removal.html. [Accessed 21 4 2016].

[3] MathWorks, “regionprops,” MathWorks, 2016. [Online]. Available: http://uk.mathworks.com/help/images/ref/regionprops.html. [Accessed 21 4 2016].

[4] MathWorks, “imgaussfilt,” MathWorks, 2016. [Online]. Available: http://uk.mathworks.com/help/images/ref/imgaussfilt.html. [Accessed 21 4 2016].

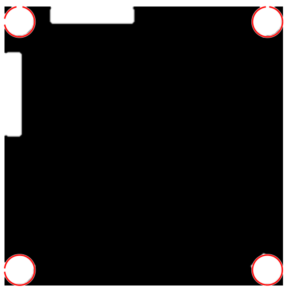
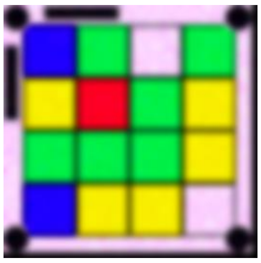
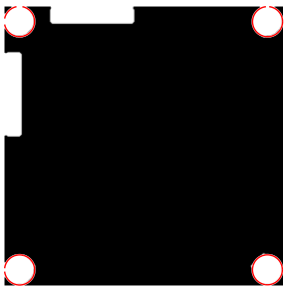
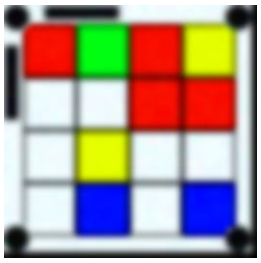
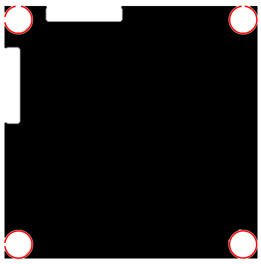
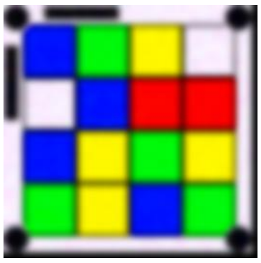
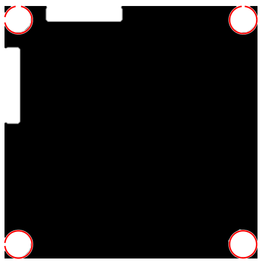
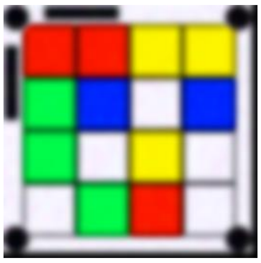
[5] J. Simon, “Increase RGB image saturation.,” MathWorks, 2011. [Online]. Available: http://uk.mathworks.com/matlabcentral/answers/10608-increase-rgb-image-saturation. [Accessed 21 4 2016].

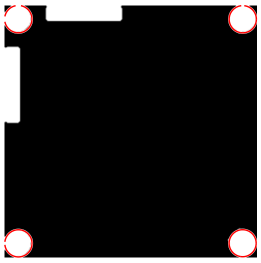
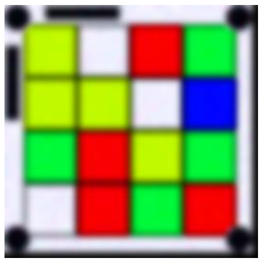
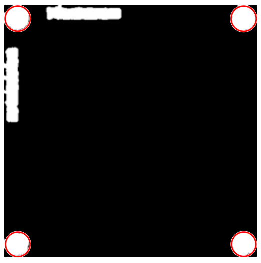
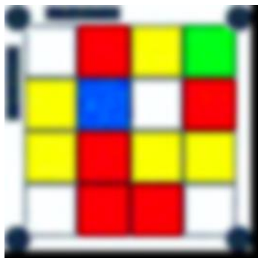
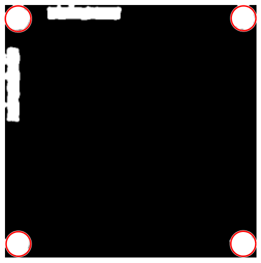
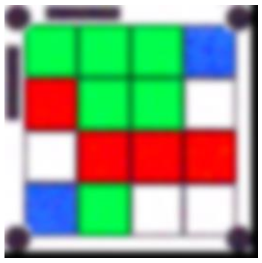
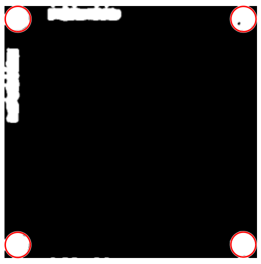
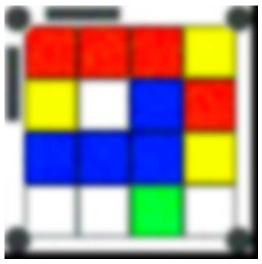
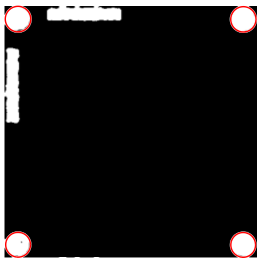
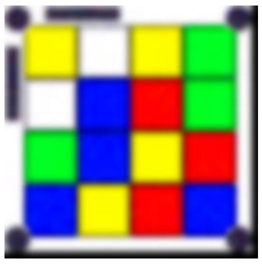
[6] MathWorks, “Hough Transform,” MathWorks, 2016. [Online]. Available: http://uk.mathworks.com/help/images/hough-transform.html. [Accessed 21 4 2016].

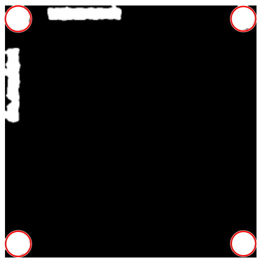
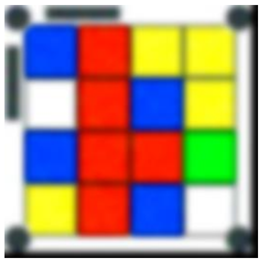
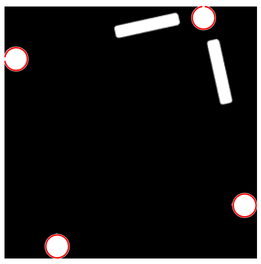
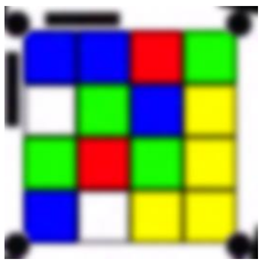
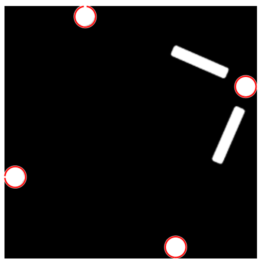
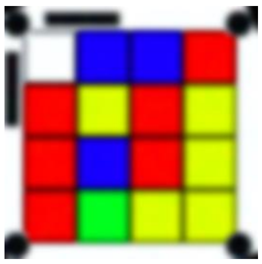
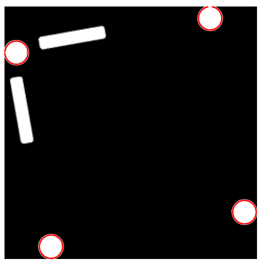
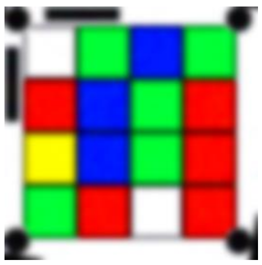
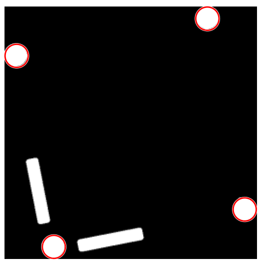
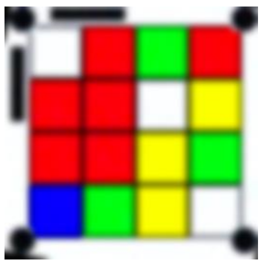
[7] MathWorks, “deconvwnr,” MathWorks, 2016. [Online]. Available: http://uk.mathworks.com/help/images/ref/deconvwnr.html. [Accessed 21 4 2016].

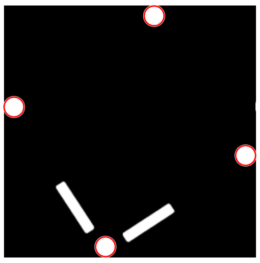
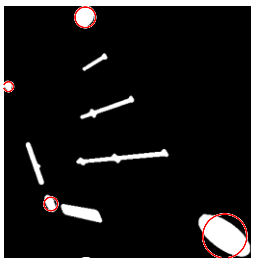
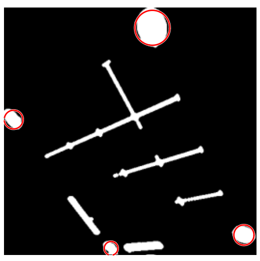
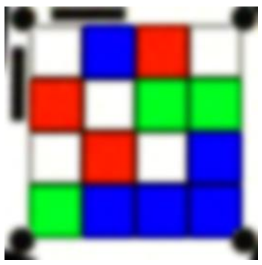
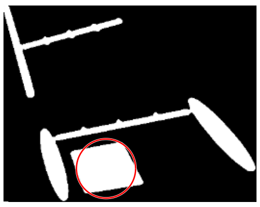
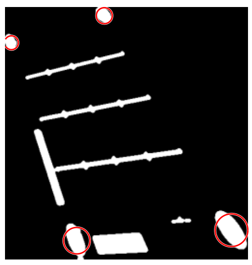
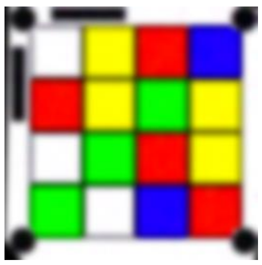
Appendix

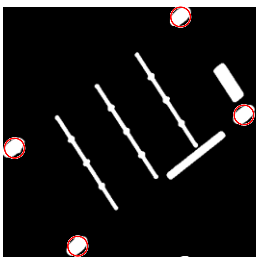
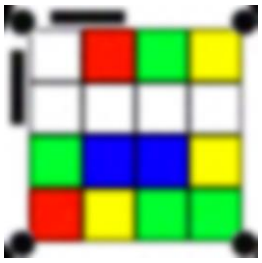
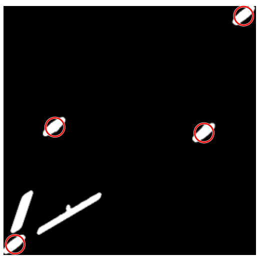
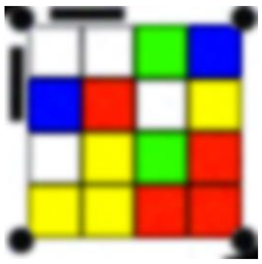
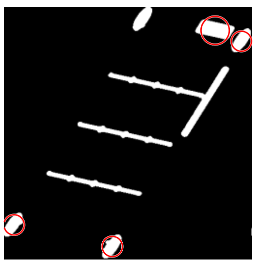
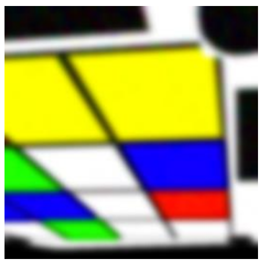
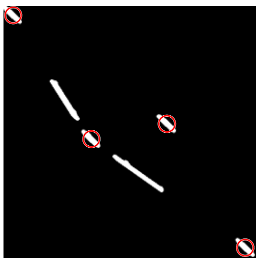
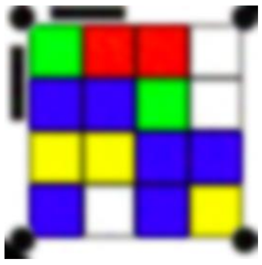
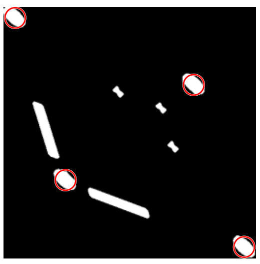
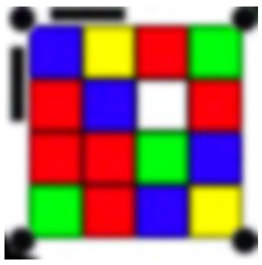
Appendix 1, result table

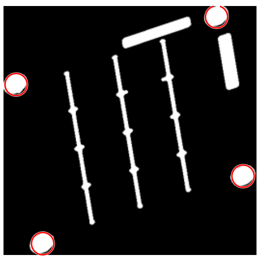
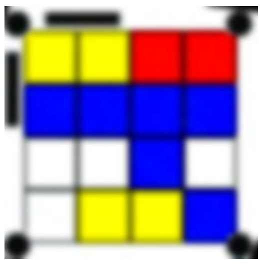
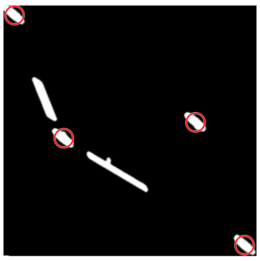
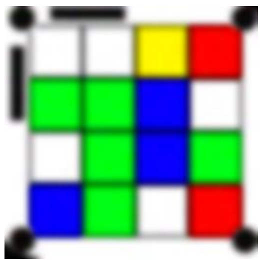
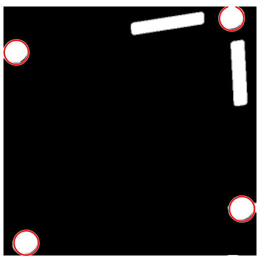
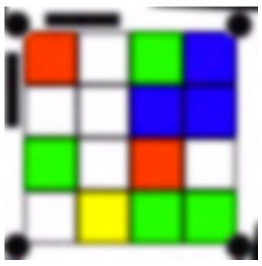
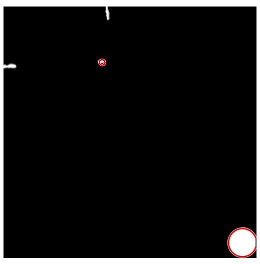
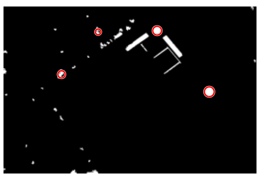
Name	Found circles	Processed image	Result
org_1			BGWG YRGY GGGY BYYW
org_2			RGRY WWRR WYWW WBWB
org_3			BGYW WBRR BYGY GYBG
org_4			RRYY GBWB GWYW WGRW

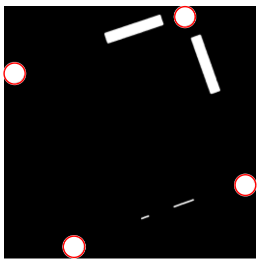
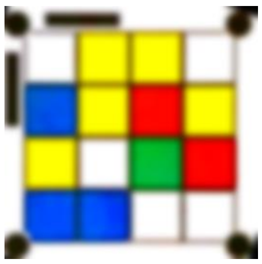
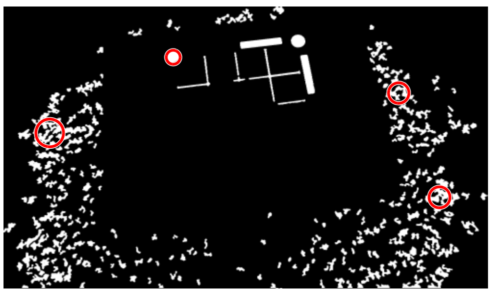
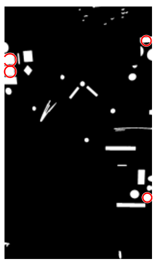
org_5			YWRG YYWB GRYG WRGR
noise_1			WRYG YBWR YRYY WRRW
noise_2			GGGB RGGW WRRR BGWW
noise_3			RRRY YWBR BBBY WWGW
noise_4			YWYG WBRG GBYR BYRB

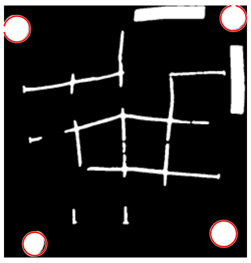
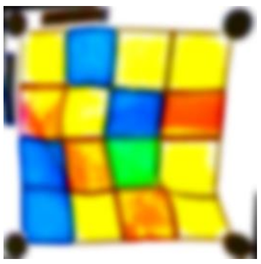
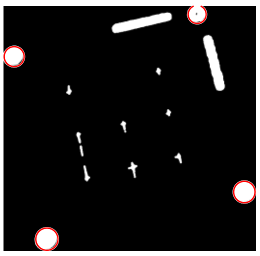
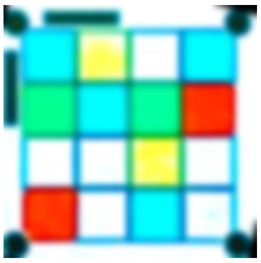
noise_5			BRYY WRBY BRRG YRBW
rot_1			BBRG WGBY GRGY BWYY
rot_2			WBBR RYRY RBRY RGYY
rot_3			WGBG RBGR YBGR GRWR
rot_4			WRGR RRWY RRYG BGYW

rot_5			YBWR GGBB GYBW BYGW
proj_1	ERROR	ERROR	ERROR
proj_2			GYGB GBWB BRRB YYBR
proj_3			WBRW RWGG WRWB GBBB
proj_4		ERROR	ERROR
proj_5			WYRB RYGY WGRY GWBR

proj1_1			WRGY WWWW GBBY RYGG
proj1_2			WWGB BRWY WYGR YYRR
proj1_3			WRONG RESULT
proj1_4			GRRW BBGW YYBB BWBY
proj1_5			BYRG RBWR RRGB GRBY
proj2_1	ERROR	ERROR	ERROR

proj2_2			YYRR BBBB WWBW WYYB
proj2_3			WWYR GGBW WGBG BGWR
proj2_4	ERROR	ERROR	ERROR
proj2_5			RWGB WWBB GWRW WYGG
IMAG0032		ERROR	ERROR
IMAG0033		ERROR	ERROR

IMAG0034		ERROR	ERROR
IMAG0035		ERROR	ERROR
IMAG0036			WYYW BYRY YWGR BBWW
IMAG0037		ERROR	ERROR
IMAG0038			WRONG RESULT

IMAG0041			WRONG RESULT
IMAG0042			BYWB GBGR WWYW RWBW
IMAG0044		ERROR	ERROR

Appendix 2, project.m full code

```
function result = colourMatrix(filename)
    %% S1 read image
    im = imread(filename);
    imOri = imread('Archive/org_1.png');

    %% S2 correction of white balance

%http://uk.mathworks.com/matlabcentral/fileexchange/41089-color-balan
ce-demo-with-gpu-computing/content/GPUDemoCode/whitebalance.m
    % WHITEBALANCE forces the average image color to be gray
    % Copyright 2013 The MathWorks, Inc.
    % Find the average values for each channel
    pageSize = size(im,1) * size(im,2);
    avg_rgb = mean( reshape(im, [pageSize,3]) );
```

```
% Find the average gray value and compute the scaling array
avg_all = mean(avg_rgb);
scaleArray = max(avg_all, 128)./avg_rgb;
scaleArray = reshape(scaleArray,1,1,3);
% Adjust the image to the new gray value
im = uint8(bsxfun(@times,double(im),scaleArray));

%% S3 set any pixel with RGB value greater than certain value to white
im(im>200)=255;

%% S4 reduce saturation and turn image into binary
HSV1 = rgb2hsv(im);
HSV1(:,:,2) = HSV1(:,:,2)/2;
HSV1(HSV1 < 0) = 0;
imLowSat = hsv2rgb(HSV1);
bi = im2bw(imLowSat,0.3);

%% S5 remove salt and pepper noise
biNoSalt = filter2(fspecial('average',8),bi);
biNoSalt = wiener2(biNoSalt,[6 6]);
biNoSalt = im2bw(biNoSalt,0.6);

%% S6 line dilate
se = strel('line',2,0);
biDi = imdilate(biNoSalt,se);
se = strel('line',2,90);
biDi = imdilate(biDi,se);

%% S7 disk dilate
se = strel('disk',2);
biDi = imdilate(biDi,se);

%% S8 disk imerode
se = strel('disk',4);
biEr = imerode(biDi,se);

%% S9 complement the binary image
biCo = imcomplement(biEr);
```

```
    %% S10 Extract objects from binary image by size, region size must
be between certain range
    biAf = bwareafilt(biCo, [200 13000]);

    %% S11 crop image
    stats = regionprops('table',biAf,'BoundingBox');
    box = stats.BoundingBox;
    %add fifth column storing x + width
    box = [box box(:,1)+box(:,3)];
    %add sixth column storing y + height
    box = [box box(:,2)+box(:,4)];
    box = sortrows(box, 1);
    minx = box(1,1);
    box = sortrows(box, 2);
    miny = box(1,2);
    box = sortrows(box, 5);
    maxx = box(end,5);
    box = sortrows(box, 6);
    maxy = box(end,6);
    newWidth = maxx-minx;
    newHeight = maxy-miny;
    biCrop = imcrop(biAf,[minx miny newWidth newHeight]);
    im = imcrop(im,[minx miny newWidth newHeight]);

    %% S12 Circle and rectangle detection
    stats =
regionprops('table',biCrop,'Centroid','Eccentricity','Area','MajorAxi
sLength','MinorAxisLength');
    %sort by area and get the ten biggest regions.
    stats = sortrows(stats,'Area');
    stats = stats(max(end-9,1):end,:);
    %find the six furthest regions away from the average of regions’
centroids
    centre = mean(stats.Centroid);
    temp = [];
    C = stats.Centroid;
    for i = 1:height(stats)
```

```
        X = [C(i,1),C(i,2);centre(1),centre(2)];
        temp(end+1) = pdist(X,'euclidean');
    end
    temp = temp(:);
    stats.DistFromCentre = temp;
    stats = sortrows(stats,'DistFromCentre');
    stats = stats(max(end-5,1):end,:);
    %sort by Eccentricity and get the 2 regions that have the largest value,
which have the highest chance of being the rectangles. The rest will be
circles.
    stats = sortrows(stats,'Eccentricity');
    statsForRect = stats(end-1:end,:);
    statsForCir = stats(1:end-2,:);
    %find the circle that is closest to the two rectangles. This circle
should be the one at the top left corner.
    temp2 = [];
    C = statsForCir.Centroid;
    R = statsForRect.Centroid;
    for i = 1:height(statsForCir)
        D = 0;
        for j = 1:height(statsForRect)
            X = [C(i,1),C(i,2);R(j,1),R(j,2)];
            D = D + pdist(X,'euclidean');
        end
        temp2(end+1) = D;
    end
    temp2 = temp2(:);
    statsForCir.DistFromRect = temp2;
    statsForCir = sortrows(statsForCir,'DistFromRect');
    statsForCirClosestToRect = statsForCir(1,:);

    %show image with found circles
    figure(1);
    imshow(biCrop);
    diameters = mean([statsForCir.MajorAxisLength
statsForCir.MinorAxisLength],2);
    radii = diameters/2;
    hold on
    viscircles(statsForCir.Centroid,radii);
    hold off

    %% S13 Recover the image
    %for each corner, find the closest circle and map them to each other.
    Corners = [1 1;1 newWidth; newHeight newWidth; newHeight 1];
    badPoints = [];
    degreeNeedToRotate = 0;
    for i = 1:4
        closest = C(1,:);
        D = pdist([C(1,:);Corners(i,:)], 'euclidean');
        for j = 2:4
            if pdist([C(j,:);Corners(i,:)], 'euclidean') < D
                closest = C(j,:);
                D = pdist([C(j,:);Corners(i,:)], 'euclidean');
            end
        end
        badPoints = [badPoints;closest];
        if closest == statsForCirClosestToRect.Centroid
            degreeNeedToRotate = (i-1) * -90;
        end
    end
    %correct positions of the four circles
    correctC1 = [24.5 24.5];
    correctC2 = [24.5 444.75];
    correctC3 = [444.75 444.75];
    correctC4 = [444.75 24.5];
    CorrectPoints = [correctC1; correctC2; correctC3; correctC4];
    tform = estimateGeometricTransform(badPoints, CorrectPoints,
'projective');
    outputView = imref2d(size(imOri));
    output  = imwarp(im,tform,'OutputView',outputView);
    %once the circles are at the right place, rotate the image accordingly.
    im = imrotate(output, degreeNeedToRotate);

    %% S14 increase colour saturation and blur the image using gaussian
filter
    im = imgaussfilt(im,5);
    im = im2double(im);
    HSV = rgb2hsv(im);
    HSV(:,:,2) = HSV(:,:,2)*3;
    HSV(HSV > 1) = 1;
    im = hsv2rgb(HSV);
    im = imgaussfilt(im,5);
    figure(2);
    imshow(im);

    %% S15 Generate result
    result = char(4,4);
    %distance from the edge of the image to the first cell
    offset_x = 30.5;
    offset_y = 30.5;
    square_size = 105;

    for x = 1:4
      for y = 1:4
        px = offset_x + ((x-0.5)*square_size);
        py = offset_y + ((y-0.5)*square_size);
        R = [im(px,py,1),im(px,py,2),im(px,py,3);1,0,0];
        redDist = pdist(R, 'euclidean');
        G = [im(px,py,1),im(px,py,2),im(px,py,3);0,1,0];
        greenDist = pdist(G, 'euclidean');
        B = [im(px,py,1),im(px,py,2),im(px,py,3);0,0,1];
        blueDist = pdist(B, 'euclidean');
        Y = [im(px,py,1),im(px,py,2),im(px,py,3);1,1,0];
        yellowDist = pdist(Y, 'euclidean');
        W = [im(px,py,1),im(px,py,2),im(px,py,3);1,1,1];
        whiteDist = pdist(W, 'euclidean');
        dists = [redDist,greenDist,blueDist,yellowDist,whiteDist];
        colours = {'R','G','B','Y','W'};
        mapObj = containers.Map(dists,colours);
        closestDist = min(dists);
        result(x,y) = mapObj(closestDist);
      end
    end
end
```