Candidate Number: 253251
10 May 2022

Image Processing Report

Candidate No: 253251

1 Introduction

The aim of this project is to create a function that can accept an image as input, which represents the colour matrix of each different projection and direction. All images need to be correctly identified the individual colour of each square within a group of 16 coloured squares. The results are displayed in a 4x4 matrix, where different numbers represent each of the 5 colours. The set of images I used is the "Images2" set, which contains images with noise and images in various directions and projections. These images must be corrected before colour processing can be performed.

2 Strategy

In general, my thinking on this image processing project is as follows. First, read the image in the computer that needs to be recognised, and make a preliminary image addition to it. Then, since there were a lot of rotated or projected images in the images2, I had to correct the target image and correct it by finding the four fixed points of the reference image. Finally, in order to get the output colour matrix required by the topic, I iteratively identified the previously corrected image

3 Main part

This section I'll expand through the function code features I wrote.

Image Pre-Process

Start by reading the image from the computer that needs to be processed. Then, In order to better handle colour pictures, in a single tone background, the part with obvious colour differences is cut with channels, and the use of LAB colour space can be completed quickly. So I converted the RGB image to a LAB colour space image. For a better reduce the noise in the image in this project, I took a low Gaussian smoothing to increase the image read and reduce noise in the original image.

Transformation

The aim of this section is to make the target image into a standard image through common ground that can be recognised by the computer.

1. Find the four control points.

I chose 'org_1.png' as a reference image because it has the right direction and can serve as a baseline for all shifts. Next, load the target image, get the centre of gravity coordinates of the corner circle in the image, and set it as a moving point. As a result, a distortion-free transformation in which the position of the moving point is adjusted to a fixed point and thus the orientation of the target image is corrected can be performed.

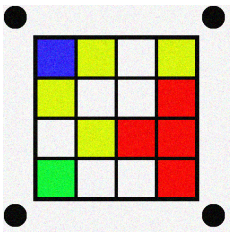


Figure1 org_1

From figure1 we can clearly find that the four black circles in the corner. I extracted these four black circles by the threshold method. But this method cannot be applied to all images. In the official documentation I saw that the canny edge detector can solve this problem very well and does not require edges, as there are obvious black borders in images and filter out the four smallest objects.

2. Rotate the image

Initially I used an image size filter to identify rotated images, and later I found a better way to correct rotation by automatically matching. The Radon transform in Matlab is mainly used, which is essentially a "Hough-like" transformation. I then took an iterative approach to finding all the angles and radii of Matlab's Radon, finding the maximum angle of the line, and correcting the image with the immrotate method such as figure2 and figure3 are shown.

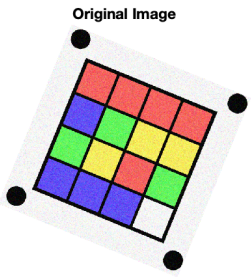


Figure2 Original Image

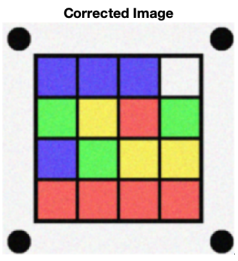


Figure3 Rotated Image

Usually, the output image will be larger than the original image. The image needs to be cropped, I used a centred crop window to extract the desired image.

Colour Recognition

The first step in colour recognition was to identify the squares as separate objects in order to get matrix positions correct in the result output. As mentioned earlier, the Use of Lab Color Space allows good detection of colour. To do this, I take an iterative approach to slicing to make my results more reliable. I only count one of the areas of each colour square, which improves my accuracy. This will make my results more robust. Now, using the same loop as in the reference image, find the colour of each square in the target image by using area and area to identify the squares. Then, mask the pixels in the specified square area on the a and b channels, providing a and b values for the pixels within each square area. Now take the average of the a and b values to get the average colour for each square range. The code then uses the Euclidean distance between the value and a specified series of a and b values to determine the colour of the square. These values are defined at the beginning of the code.

```
resultMatrix =

  4×4 string array

    "G"    "R"    "G"    "W"
    "B"    "B"    "G"    "W"
    "R"    "W"    "R"    "Y"
    "R"    "Y"    "Y"    "W"
```

Figure4 Colour Matrix

4 Problem that can't saved : proj_x Image

proj_1 - proj_5 look like more affine, rotated images, so it's more complicated to handle

My algorithm is not enough to solve these few pictures. I've tried setting control points manually and couldn't fix some of these image.

5 Comments about Photos.zip

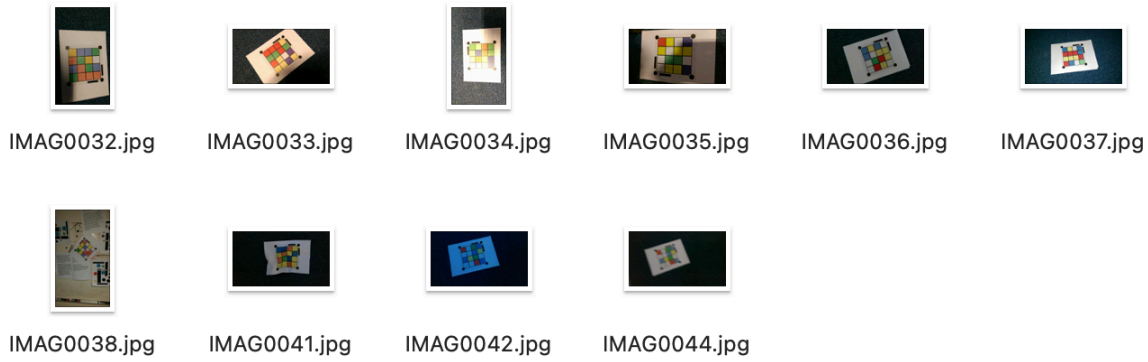


Figure5 photos

If you try to recognise a live image, you must take a reference image with standard orientation, ideal lighting, and shadows so that there are no shadows in the photo.

It can be seen that IMAG0032.jpg, IMAG0034.jpg, IMAG0035.jpg have distinct dark shadows. The IMAG0037 .jpg illumination is most uniform and can be used as a reference image. The IMAG0042 .jpg has a strong blue hue, probably due to the camera equipment. Many live photos have inconsistent lighting with shadow patches, which can be corrected by using morphological operators to even out the lighting in the image. Once you've made corrections for non-uniform illumination, you can select a global threshold to depict each object in the background, and then use an open block to correct for uneven illumination. At the same time, proper pre-treatment and use of the Lab colour space are carried out.

For images with colour illumination, you can use histogram normalisation to correct this. The histogram of the target image and the reference image will be found, and then the intensity range of the target image should be changed to match the intensity range of the reference image. This should remove the lighting colour, but may mean that some squares will change colour slightly. This discolouration should then be fixed using a filter to correct the image.

Directional rotation, photos.zip in the images are all shot at a certain angle, of which IMAG0034 .jpg more balanced. Using my previous algorithm, rotation correction can be done to some extent. The .jpg imaging0041 .jpg and IMAG0044 are blurry, which may be caused by the instability of the camera equipment during shooting. For blurry images, you should use an edge detector, for example, You should use Canny. The Canny method is the strongest edge detection method in Matlab because it uses two different thresholds (detect strong edges and weak edges), and weak edges are included in the output only when weak edges are connected to strong edges. Therefore, this method is least likely to be affected by noise compared to other methods such as Sobel. Once the edges are detected, expanding with square structural elements will help to obtain a mesh as an object. SIFT matching can then be used to automatically convert the target image to the same orientation and scale as the reference image. From this point on, you can use the same method as above to get the colour and position of the square, and finally output the result into the matrix.

Result

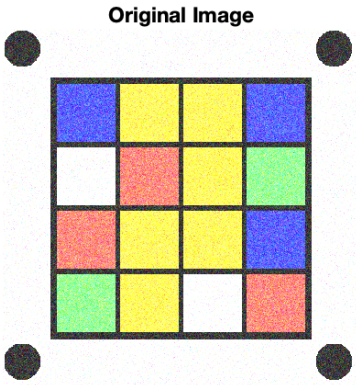
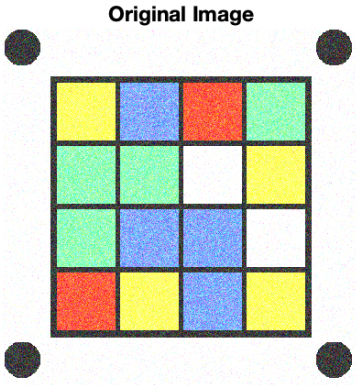
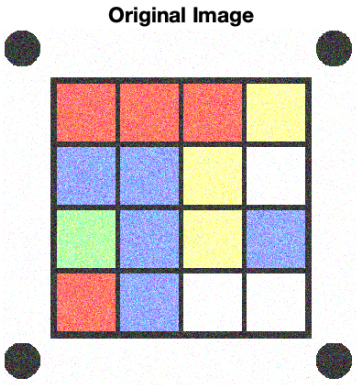
IMAGE	RESULT	
	resultMatrix = 4×4 string array "B" "Y" "Y" "B" "W" "R" "Y" "G" "R" "Y" "Y" "B" "G" "Y" "W" "R"	Image: noise_1.png
	resultMatrix = 4×4 string array "Y" "B" "R" "G" "G" "G" "W" "Y" "G" "B" "B" "W" "R" "Y" "B" "Y"	Image: noise_2.png
	resultMatrix = 4×4 string array "R" "R" "R" "Y" "B" "B" "Y" "W" "Y" "B" "Y" "B" "R" "B" "W" "W"	Image: noise_3.png

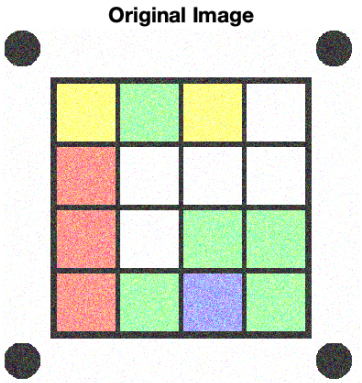
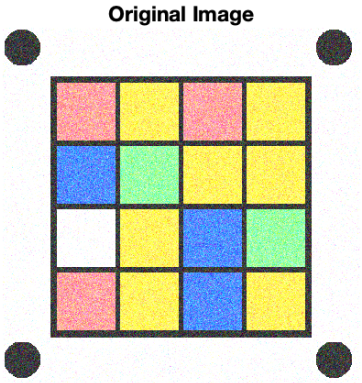
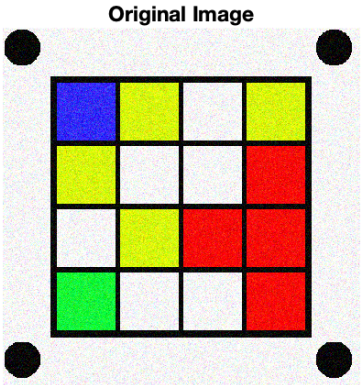
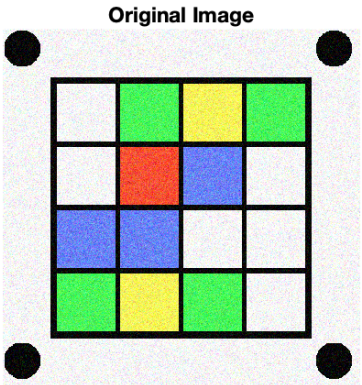
IMAGE	RESULT	
	Command Window resultMatrix = 4×4 string array "Y" "G" "Y" "W" "R" "W" "W" "W" "R" "W" "G" "G" "R" "G" "B" "G" fx >>	Image: noise_4.png
	Command Window resultMatrix = 4×4 string array "R" "Y" "R" "Y" "B" "G" "Y" "Y" "W" "Y" "B" "G" "R" "Y" "B" "Y" fx >>	Image: noise_5.png
	Command Window resultMatrix = 4×4 string array "B" "Y" "W" "Y" "Y" "W" "W" "R" "W" "Y" "R" "R" "G" "W" "W" "R" fx >>	Image: org_1.png
	Command Window resultMatrix = 4×4 string array "W" "G" "Y" "G" "W" "R" "B" "W" "B" "B" "W" "W" "G" "Y" "G" "W" fx >>	Image: org_2.png

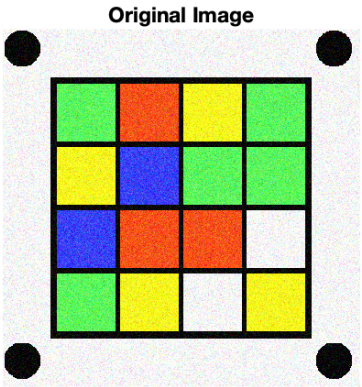
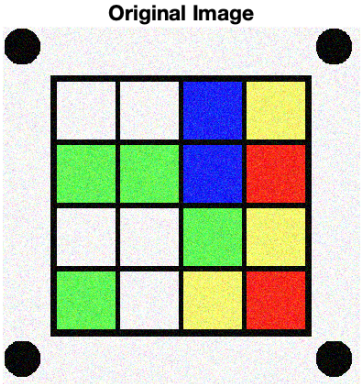
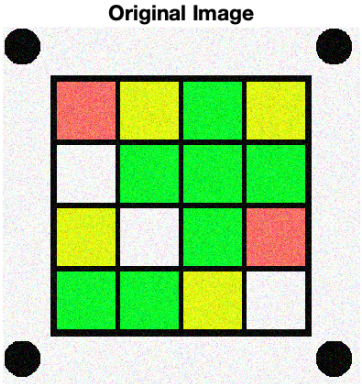
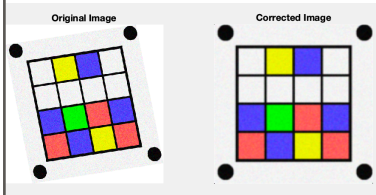
IMAGE	RESULT	
	Command Window resultMatrix = 4×4 string array "G" "R" "Y" "G" "Y" "B" "G" "G" "B" "R" "R" "W" "G" "Y" "W" "Y" fx >>	Image: org_3.png
	Command Window resultMatrix = 4×4 string array "W" "W" "B" "Y" "G" "G" "B" "R" "W" "W" "G" "Y" "G" "W" "Y" "R" fx >>	Image: org_4.png
	Command Window resultMatrix = 4×4 string array "R" "Y" "G" "Y" "W" "G" "G" "G" "Y" "W" "G" "R" "G" "G" "Y" "W" fx >>	Image: org_5.png
	Command Window resultMatrix = 4×4 string array "W" "Y" "B" "W" "W" "W" "W" "W" "B" "G" "R" "B" "R" "B" "Y" "R" fx >>	Image: proj1_1.png

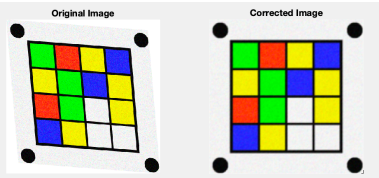
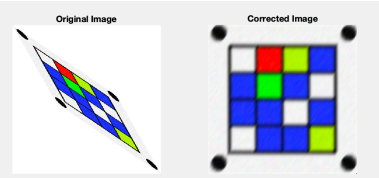
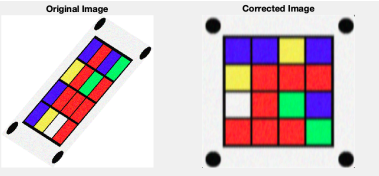
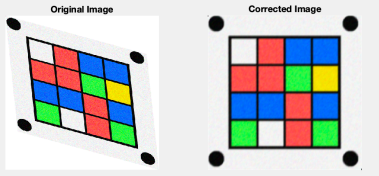
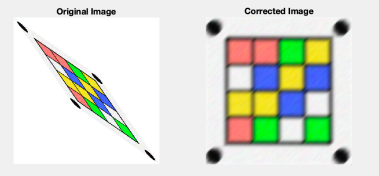
IMAGE	RESULT	
	Command Window resultMatrix = 4×4 string array "G" "R" "Y" "B" "Y" "G" "B" "Y" "R" "G" "W" "Y" "B" "Y" "W" "W" fx >>	Image: proj1_2.png
	resultMatrix = 4×4 string array "W" "R" "G" "B" "B" "G" "B" "W" "B" "B" "W" "B" "W" "B" "B" "G" >>	Image: proj1_3.png
	Command Window resultMatrix = 4×4 string array "B" "B" "Y" "B" "Y" "R" "R" "R" "W" "R" "G" "B" "R" "R" "R" "G"	Image: proj1_4.png
	Command Window resultMatrix = 4×4 string array "W" "R" "B" "B" "R" "R" "G" "Y" "B" "B" "R" "B" "G" "W" "R" "G" fx >>	Image: proj1_5.png
	Command Window resultMatrix = 4×4 string array "R" "R" "G" "Y" "W" "B" "Y" "B" "Y" "Y" "B" "W" "R" "G" "W" "G" fx >>	Image: proj2_1.png

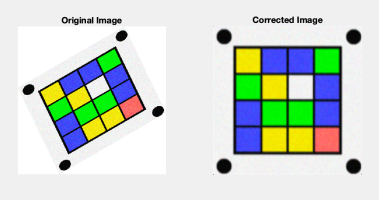
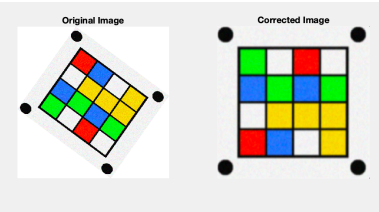
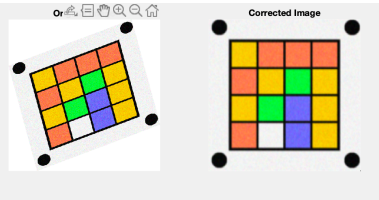
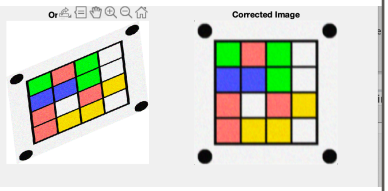
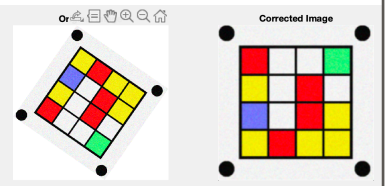
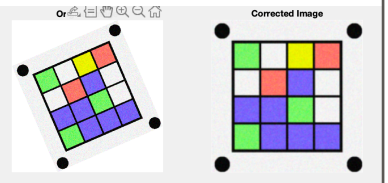
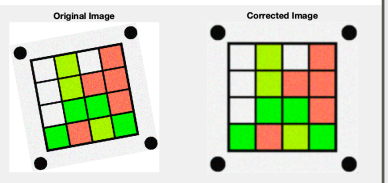
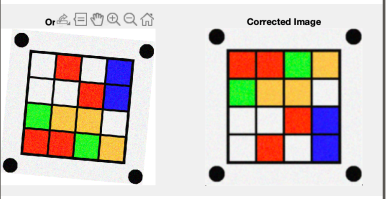
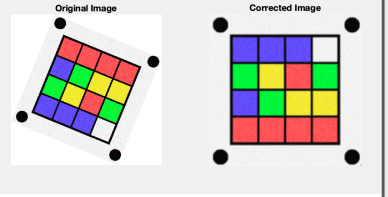
IMAGE	RESULT	
	Command Window resultMatrix = 4×4 string array "Y" "B" "B" "G" "G" "Y" "W" "B" "B" "G" "G" "B" "B" "Y" "Y" "R" fx >>	Image: proj2_2.png
	Command Window resultMatrix = 4×4 string array "G" "W" "R" "W" "B" "G" "B" "G" "W" "Y" "Y" "Y" "R" "B" "W" "Y" fx >>	Image: proj2_3.png
	Command Window resultMatrix = 4×4 string array "Y" "R" "R" "R" "R" "Y" "G" "Y" "Y" "G" "B" "Y" "R" "W" "B" "Y" fx >>	Image: proj2_4.png
	Command Window resultMatrix = 4×4 string array "G" "R" "G" "W" "B" "B" "G" "W" "R" "W" "R" "Y" "R" "Y" "Y" "W" fx >>	Image: proj2_5.png
		Image: proj_1.png My Algorithm can't solve
		Image: proj_2png My Algorithm can't solve
		Image: proj_3.png My Algorithm can't solve
		Image: proj_4.png My Algorithm can't solve
		Image: proj_5.png My Algorithm can't solve

IMAGE	RESULT	
	Command Window resultMatrix = 4×4 string array "R" "W" "W" "G" "Y" "W" "R" "W" "B" "W" "R" "Y" "Y" "R" "Y" "Y" fx >>	Image: rot_1.png
	Command Window resultMatrix = 4×4 string array "G" "W" "Y" "R" "W" "R" "B" "W" "B" "B" "G" "W" "G" "B" "B" "B" fx >>	Image: rot_2.png
	Command Window resultMatrix = 4×4 string array "W" "G" "W" "R" "W" "G" "R" "R" "W" "G" "G" "R" "G" "R" "G" "G" fx >>	Image: rot_3.png
	Command Window resultMatrix = 4×4 string array "R" "R" "G" "Y" "G" "Y" "Y" "W" "W" "W" "R" "B" "W" "R" "W" "B" fx >>	Image: rot_4.png
	Command Window resultMatrix = 4×4 string array "B" "B" "B" "W" "G" "Y" "R" "G" "B" "G" "Y" "Y" "R" "R" "R" "R" fx >>	Image: rot_4.png