Image processing coursework

Introduction

The aim of this project was to create a function in Matlab, using image processing techniques which correctly identifies the individual colour of each square within a group of 16 coloured squares. The results are displayed in a 4x4 matrix with a different number representing each of the 5 colours. The set of images that the code has been written for is the 'SimulatedImages' set, which contained images with noise and images in various orientations and projections. These images had to be corrected before colour processing could be done.

Methodology

This problem was split into two main parts, the first being reading images that were rotated or projected, and then translating them into a standard orientation through finding common points. Once this was achieved the next part was to identify the squares individually, read the average colour of each square to account for noise, and then output the results in a matrix.

Transformation

The first step was to load in a reference image (org_1), this image was in the correct orientation and would serve as the baseline for all transformations. The centroid coordinates of the 4 corner circles were found, which were set as fixed points. Next the target image was loaded in and the centroid coordinates for the corner circles in this image were found and set as moving points. This then allowed an imwarp transform to be performed, matching the location of the moving points to the fixed points, therefore correcting the orientation of the target image.

The reference image can be seen in Figure (1). To find the circles on the reference image, I first converted the image to black and white, with a threshold of 0.5 to allow any slight variations in intensity to be neglected. A filter was then applied to the black and white image to remove noise, which can be seen in Figure (2). An erosion followed by a dilation could also be used instead of a filter to achieve the same results.

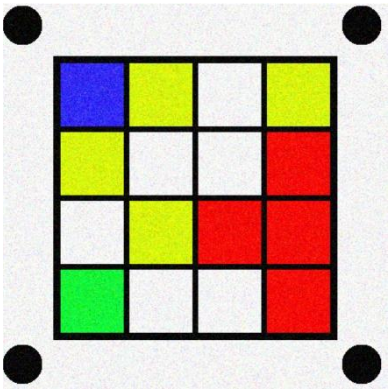

Figure (1) – Reference image

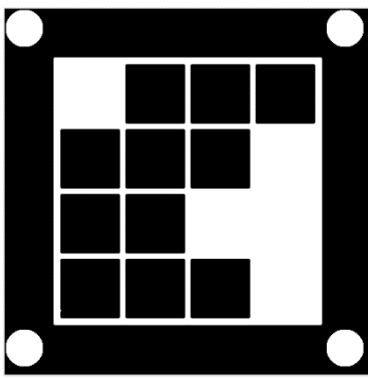

Figure (2) – BW Reference image with filter applied

Next the number of object and object properties in the image were to be found, to identify the circle regions. This was achieved using the image processing toolbox functions bwconncomp and regionprops. A loop was created so that the maximum area in the image (the squares) were removed, which left the circles isolated. The centroid coordinates of each circle were then recorded in a FixedPoints array using the Vec2mat function.

Now the target image was loaded in and converted to black and white with a threshold of 0.5, again to allow slight variations in intensity to be neglected. The noise holes in this image were filled using imfill and a filter was applied Figure (4). For the sake of this report, the target image used to demonstrate is 'proj_1.png' Figure (3).

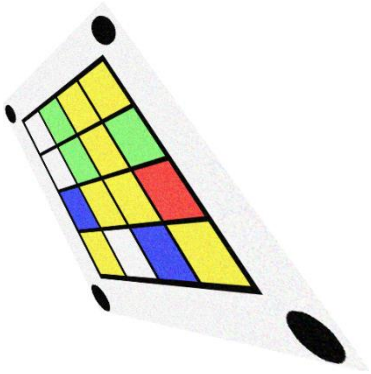

Figure (3) – 'Proj_1.png'

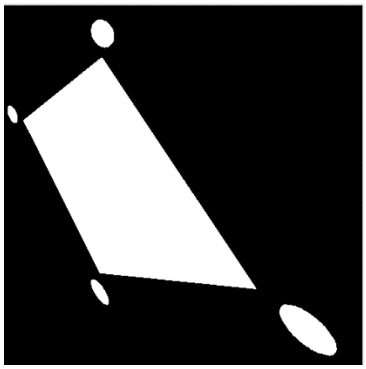

Figure (4) – BW 'Proj_1.png' with noise holes filled and filter applied.

The circle regions in the target image were then found in the same way as in the Reference image, using the functions bwonncomp and regionprops, and again using a loop to remove the maximum area and isolate the circles. The centroid coordinates of each circle were then recorded in a MovingPoints array using the Vec2mat function. Once complete, all points required to perform the transformation of the target image have been obtained.

To perform the transformation, the image processing function fitgeotrans is used on the MovingPoints and FixedPoints arrays, this effectively translates the MovingPoints control points to the FixedPoints, transforming the target image into the orientation and perspective of the reference image. The 'projective' option was used as it allows for the transformation of both rotated images and images in a different perspective. Imwarp is then used to get the corrected image. A problem I encountered here was that the newly corrected image was not the same size as the reference image, making the next part of the assignment, colour correction, difficult. This was solved by using the imref2d and imwarp function to resize the corrected image, as can be seen in Figure(5).

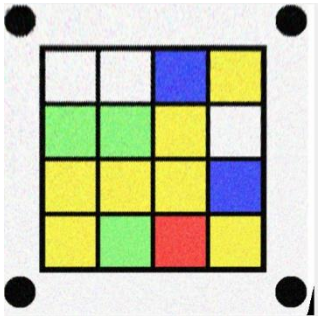

Figure (5) – 'Proj_1.png' corrected

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. This is achieved by isolating the squares and finding each of their centroid coordinates, which are then referenced when creating the colour matrix. To do this, I first converted the reference image into Lab colour space, converted it into black and white and then used a log filter to detect square edges. Noise meant the edges weren't detected completely correctly, so dilation was performed to correct this. The image after dilation can be seen in figure (6) .

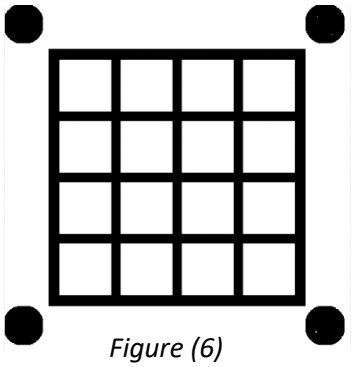

Figure (6)

Next bwconncomp and regionprops were used again in order to get each square as an object, along with their area, extent and centroid values. A loop was created in which Extent was used to determine the objects, as the extent of squares should be approximately 1, and area was used to filter out objects created due to the remaining noise, which would all have a very small area. This looped through each object and if determined to be a square, the centroids were added to an array, ending up as an array of 16 coordinates, representing the centroids of each of the individual squares.

I then converted the target image into Lab colour space, split the image into separate colour channels and applied a median filter to each of the channels, to further smooth the image of noise. This filter helps to obtain more instances of colour within in each square, aiding with the colour identification further on. The Lab colour space was chosen as it allows clear distinction between colours.

The colour of each square in the target image is now found using the same loop as in the reference Image, using extent and area to identify the squares. Pixels in the given square region are then masked over the a and b channels, which gives the a and b values for the pixels within each square region. The mean of the a and b values are now taken to obtain the average colour for each square region. The code then uses the Euclidean distance between the values, and the set of specified a and b values, to determine which colour the square is. These values were defined at the start of the code.

The a and b values for the colours were chosen by checking these values in a variety of the images. I then averaged the values and tried those, which didn't provide accurate colour identification for all all the images. I eventually came up with the values through trial and error, and advice I had been given. I considered using the L channel to help with colour identification but thought due to the various colour shades between images, this could cause inaccurate results.

Results and function performance

Results can be seen in the appendix, with the only image file that can not be processed being proj_2, causing an error message stating that the array created for that image is too large, causing Matlab to run too slowly and become unresponsive, this could be due to my computer having insufficient ram. The time taken for the function to complete is an average of 2.5s for all the image files bar two, these two outliers were 'proj_4' taking 4.1s and 'proj_5' taking much longer at 30s. This extra time has to do with the perspective of these images, as they are both in a similar perspective which I assume, takes longer for the objects to be identified as the images are stretched towards the bottom.

Improvements that could be made include using letters to represent the colours in the output matrix rather than numbers, and further reducing noise in the images to ensure the square regions are as defined as they can be.

Live Images

If trying to identify live images, a reference image must be taken in which has standard orientation, with ideal lighting and shading so that there are no shadows in the photo. Many of the live photos have inconsistent lighting with patches of shadows, this can be corrected through using morphological operators to even out the lighting in the Image. Once corrected for nonuniform illumination, a global threshold could be picked that delineates every object from the background, then an Opening block is used to correct for uneven lighting [1].

For the images with coloured lighting, histogram normalisation could be used to correct this. The histogram for both the target image and reference image would be found, 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 some squares become discoloured slightly. This discolouration should then be fixed with filters to correct the image.

For the blurred images, an edge detector should be used such as Canny should be used. The canny method is the strongest edge detection method in Matlab as it uses two different thresholds (to detect strong and weak edges) and includes the weak edges in the output only if they are connected to a strong edge. [2] This method is therefore the least likely to be affected by noise, when compared to other methods such as Sobel. After edges are detected, dilation with square structuring elements will help obtain the grid as an object. SIFT matching could then be used to automatically transform the target image into the same orientation and scale as the reference image. From this point the same method as performed in the above methodology could be used to obtain the square colours and location, finally outputting the results into a matrix.

Appendix

Legend

1 = White
2 = Blue
3 = Yellow
4 = Green
5= Red

Results

noise_1

```
result =

     2     3     3     2
     1     5     3     4
     5     3     3     2
     4     3     1     5
```

noise_2

```
result =

     3     2     5     4
     4     4     1     3
     4     2     2     1
     5     3     2     3
```

noise_3

```
result =

     5     5     5     3
     2     2     3     1
     3     2     3     2
     5     2     1     1
```

noise_4

```
result =

     3     4     3     1
     5     1     1     1
     5     1     4     4
     5     4     2     4
```

noise_5

```
result =

     5     3     5     3
     2     4     3     3
     1     3     2     4
     5     3     2     3
```

org_1

```
result =

     2     3     1     3
     3     1     1     5
     1     3     5     5
     4     1     1     5
```

org_2

```
result =

     1     4     3     4
     1     5     2     1
     2     2     1     1
     4     3     4     1
```

org_3

```
result =

     4     5     3     4
     3     2     4     4
     2     5     5     1
     4     3     1     3
```

org_4

```
result =

     1     1     2     3
     4     4     2     5
     1     1     4     3
     4     1     3     5
```

org_5

```
result =

     5     3     4     3
     1     4     4     4
     3     1     4     5
     4     4     3     1
```

Proj_1

```
result =

     1     1     2     3
     4     4     3     1
     3     3     3     2
     3     4     5     3
```

Proj_3

```
result =

     1     4     1     2
     3     5     5     3
     4     2     4     3
     2     5     3     2
```

Proj_4

```
result =

     2     5     3     3
     5     4     2     5
     2     3     3     2
     4     4     4     3
```

Proj_5

```
result =

     3     4     3     5
     3     5     1     2
     2     4     2     2
     4     2     4     5
```

Proj1_1

```
result =

     1     3     2     1
     1     1     1     1
     2     4     5     2
     5     2     3     5
```

Proj1_2

```
result =

     4     5     3     2
     3     4     2     3
     5     4     1     3
     2     3     1     1
```

Proj1_3

```
result =

     1     5     3     2
     2     4     2     1
     2     2     1     2
     1     2     2     3
```

Proj1_4

```
result =

     2     2     3     2
     3     5     5     5
     1     5     4     2
     5     5     5     4
```

Proj1_5

```
result =

     1     5     2     2
     5     5     4     3
     2     2     5     2
     4     1     5     4
```

Proj2_1

```
result =

     5     5     4     3
     1     2     3     2
     3     3     2     1
     5     4     1     4
```

Proj2_2

```
result =

     3     2     2     4
     4     3     1     2
     2     4     4     2
     2     3     3     5
```

Proj2_3

```
result =

     4     1     5     1
     2     4     2     4
     1     3     3     3
     5     2     1     3
```

Proj2_4

```
result =

     3     5     5     5
     5     3     4     3
     3     4     2     3
     5     1     2     3
```

Proj2_5

```
result =

     4     5     4     1
     2     2     4     1
     5     1     5     3
     5     3     3     1
```

Rot_1

```
result =

     5     1     1     4
     3     1     5     1
     2     1     5     3
     3     5     3     3
```

Rot_2

```
result =

     4     1     3     5
     1     5     2     1
     2     2     4     1
     4     2     2     2
```

Rot_3

```
result =

     1     3     1     5
     1     3     5     5
     1     4     4     5
     4     5     3     4
```

Rot_4

```
result =

     5     5     4     3
     4     3     3     1
     1     1     5     2
     1     5     1     2
```

Rot5

```
result =

     2     2     2     1
     4     3     5     4
     2     4     3     3
     5     5     5     5
```

Code

```
function result = colourMatrix(filename)

%Set values for colours based on Lab colour space

white = [128,128];
blue = [145,85];
yellow = [116,190];
green = [65,185];
red = [170,150];
colours = [white; blue; yellow; green; red];

                    %Transformation

Reference=imread('SimulatedImages\org_1.png'); %Read image to be reference for transformation.
Refbw=~im2bw(Reference,0.5); % Convert reference image to binary with threshold 0.5
medfilt=medfilt2(Refbw);  %Filter applied

%Find circle regions

CC=bwconncomp(medfilt);
z=regionprops(medfilt,'Eccentricity','Centroid','Area');
Areas=[z.Area];


%Maximum area removed to isolate circles

for i = 1:CC.NumObjects
   if z(i).Area == max(Areas)
      z(i)=[];
      break
   end
end

Centroid=[z.Centroid]; %Take centroids of circles
FixedPoints = vec2mat(Centroid,2); %Set centroids as fixed points


% Load in image to be transformed

targImg=imread(filename) ; %load image for analysis

BW=~im2bw(targImg,0.5); % Convert target image to binary with threshold 0.5
BWfill=imfill(BW,'holes'); %Fill noise holes
medfilt=medfilt2(BWfill); %Filter applied


%Find circle regions

CC=bwconncomp(medfilt);
Z=regionprops(medfilt,'Eccentricity','Centroid','Area');
```

```
Areas=[Z.Area];


%Maximum area removed to isolate circle

for i = 1:CC.NumObjects
   if Z(i).Area == max(Areas)
     Z(i)=[];
     break
   end
end


Centroid2=[Z.Centroid]; %Take centroids of circles
MovingPoints=vec2mat(Centroid2,2); %Set centroids as moving points


%Applying the transformation

tform=fitgeotrans(MovingPoints,FixedPoints, 'Projective');
trans=imwarp(targImg,tform);



R=imref2d(size(Reference)) %Resize transformed target image to same size as reference image
corrected2 = imwarp(targImg,tform,'OutputView',R);
figure(3)
imshow(corrected2); %Display the resized target image, post transformation.
title('Corrected image')



                        % Colour recognition
% Identify squares as objects

C = makecform('srgb2lab');
RefLab = applycform(Reference, C); %Convert image to lab colour space
v=fspecial('log',[3 3],0.5); % log filter used to detect edges
BW=im2bw(Reference,0.1); BWref=im2bw(Reference,0.1);
Imgfilt=imfilter(BW,v,'replicate');Imgfiltref=imfilter(BWref,v,'replicate');
Imgfiltdil = imcomplement(imdilate(Imgfilt,ones(7))); %Dilate to fill holes and sharpen outlines
Imgfiltrefdil = imcomplement(imdilate(Imgfiltref,ones(7))); %Dilate to fill holes and sharpen outlines
h = fspecial('average',3);

% Locate centroids of squares

Z=regionprops(Imgfiltdil, 'Centroid', 'Extent','Area');
CC = bwconncomp(Imgfiltdil);
Centroids = [];
BW2=zeros(CC.ImageSize);
for p=1:CC.NumObjects  %Loop through each object
```

```
if Z(p).Extent < 0.9 || Z(p).Area < 10
continue;
else
Centroids = [Centroids; Z(p).Centroid];
end
end

%Add filter to colour channels to help with colour identification

medfiltImg = corrected2;
medfiltImg(:,:,1) = medfilt2(medfiltImg(:,:,1), [9 9]);
medfiltImg(:,:,2) = medfilt2(medfiltImg(:,:,2), [9 9]);
medfiltImg(:,:,3) = medfilt2(medfiltImg(:,:,3), [9 9]);
RefLab = applycform(medfiltImg, C); % Covert to lab colour space
Imgfiltrefdil = imdilate(im2bw(medfiltImg,0.5),ones(7)); %Dilate to fill holes and sharpen edges.

%Spliting the Image into seperate channels

L = RefLab(:,:,1);
a = RefLab(:,:,2);
b = RefLab(:,:,3);

 % Locating the squares, putting colour values into grids

Gcol = zeros(4,4);


for p=1:CC.NumObjects  %loop through each object
if Z(p).Extent < 0.9 || Z(p).Area < 10
continue;
else
match = abs(Z(p).Centroid -Centroids); % find matching squares between images
matchmin = min(match);
[~,index] = ismember(match,matchmin, 'rows');
index = find(index==1); % find index between reference square centroids

Aav = mean(a(ismember(labelmatrix(CC),p))); % Average a colour of square
Bav = mean(b(ismember(labelmatrix(CC),p))); % Average b colour of square
avAB = [Aav, Bav];
[mindist,colcode] = min(sqrt(sum((colours-avAB).^2,2))); % Difference between reference colour and
square colour
x = ceil(index/4);
y = index -(x-1)*4;
Gcol(y,x) = colcode;
end

end

% Return result
result = Gcol
end
```

References

1 - Mathworks. (2020). *Computer Vision Toolbox*, Correct Nonuniform Illumination, Retrieved April 28, 2020 from https://uk.mathworks.com/help/vision/ug/correct-nonuniform-illumination.html

2 - Mathworks. (2020). *Image Processing Toolbox*, Edge Detection, Retrieved April 28, 2020 from https://uk.mathworks.com/help/images/edge-detection.html#buh9ylp-13