Warren Boult 186959

Image Processing Coursework

Introduction

The goal of this project is to use image processing techniques to identify coloured squares in a selection of images, similar to how a computer might need to identify a grid of colours for various tasks. We were provided with a set of images of grids of squares in various projections and orientations, with varying amounts of noise, and the goal was to build a Matlab function which would read in one of these images, and return a matrix containing the colour codes for each grid entry.

Methodology

There were 3 key parts to this problem: registering images that were either rotated or projected, getting the images into a standard orientation; the identification of the squares in their correct grid positions to be able to fill the colour matrix in the right positions; and accounting for noise in the images, to allow the colours to be identified correctly.

I firstly set about building automatic image registration. This would be achievable by identifying the 4 circles at the corners of the original image (org_1.png), and using the centroid coordinates of these as the fixed points for an imwarp transform in Matlab, using the centroid coordinates of the circles of the target image as the moving points.

To find the circles, I first average filtered over the images (figure 1), to blur the images slightly and reduce the noise a bit, to ensure the shapes I found would be similar. Then, I converted the images to black and white (figure 2a), thresholding at 0.4 to ensure any slight variations in intensity would still form the desired image, and then inverted the image, and filled the holes (figure 2b). Thus, as can be seen in figure 2b, I was left with the 4 circles, and a big square in the middle of the image. Figure 3 shows one of the projected images in this state, as proof that the same process gave the same results for all the other images.

Then, Matlab's bwconncomp and regionprops (using properties 'Area' and 'Centroid') functions are used to get the number of objects and properties of objects in the image. Firstly for the base image I found the index of the object with the greatest area (ie the grid square), and then looped over the objects and read the centroid coordinates of each non-max-area object (ie the circles) into the fixedPoints array. Then I did the same thing for the target image (ignoring objects with area < 10, ie noise), taking the centroids of the projected/rotated circle objects, and reading these into my movingPoints array. Thus, the points required to imwarp transform the target image are obtained.

Finally, to achieve the registered image, fitgeotrans is called with the movingPoints and fixedPoints arrays, using the 'projective' option, since this registers both simple rotated images, and images with a perspective projection, thus catching all cases for the test images. Then imwarp is called to get the final image, setting the size to be the same as the base image:

```
projimg = imwarp(targimg,mytform,'OutputView',imref2d(size(basicimg)));
```

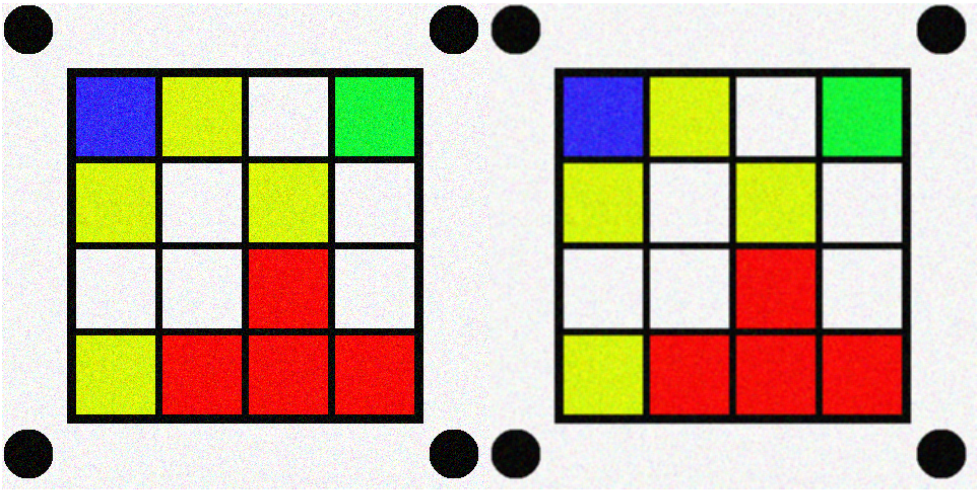

Figure 1 - a) base image; b) average filtered base image

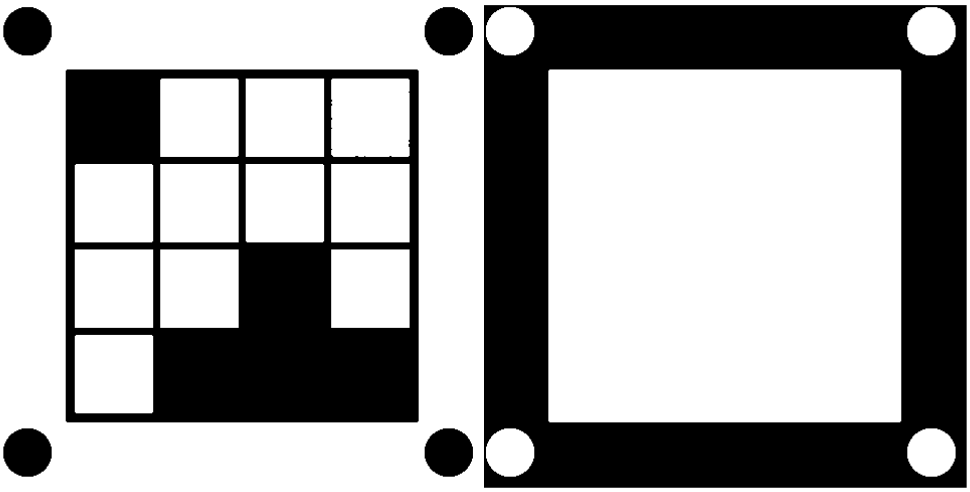

Figure 2 - a) base image converted to b&w, threshold 0.4; b) inverted b&w image with holes filled

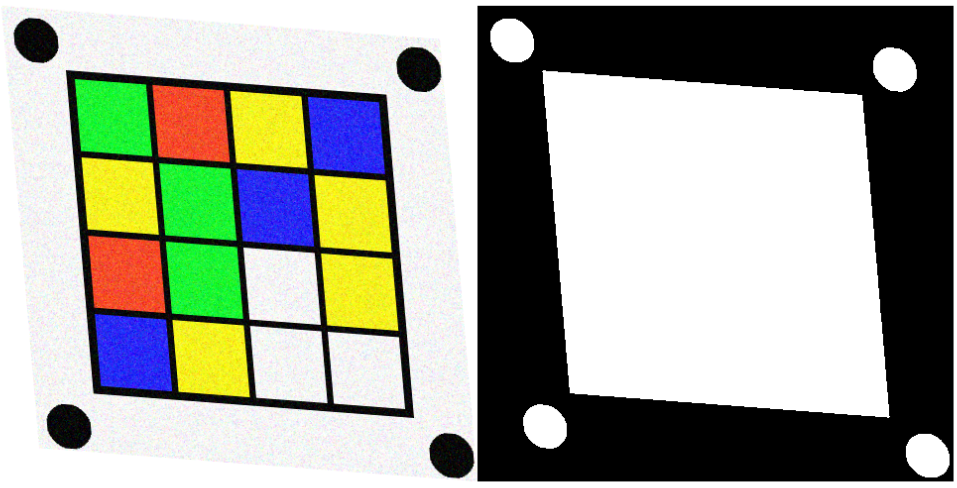

Figure 3 - a) projected image (proj1_2) in original form; b) converted to just have circles and grid square

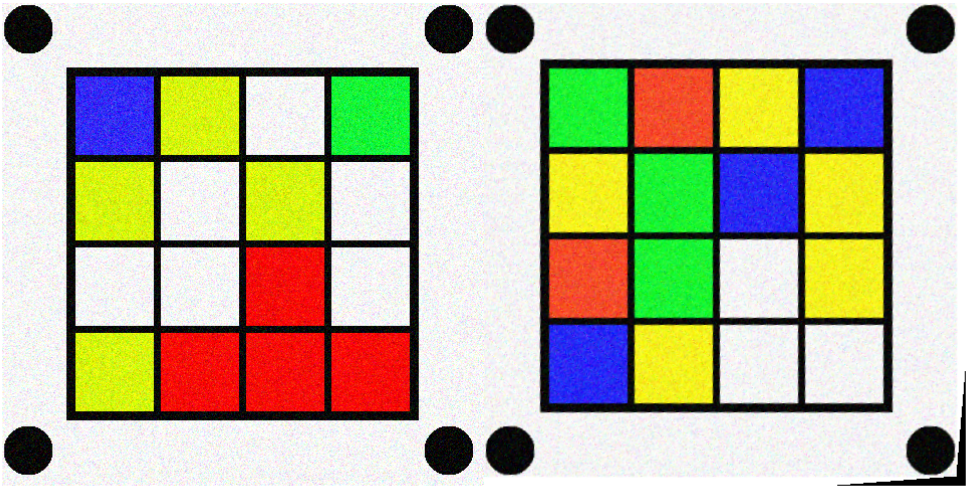

Figure 4 - a) base image (org_1), reference for registering images; b) proj1_2 image after registration

Figure 4 shows the result of registering image proj1_2 alongside the base image org_1. During my attempts to find the circles for image registration, I also played around with using dilation and erosion. In one experiment, I converted the images to Lab colour space, thresholded the L values to obtain a b&w image, then dilated with a structuring element of size 7 to fill in the noise, and did an imfill and imcomplement to obtain the circles and gridsquare. This worked well on the noisy untransformed images, but not so well on the projected images, as figure 5 shows. The Matlab script is shown below.

```
projimg = imread('SimulatedImages/noise_2.png');
C = makecform('srgb2lab');
labimg = applycform(projimg, C);
L = labimg(:,:,1);
thresh = L>252;
threshdilate = imdilate(thresh, ones(7));
finalim = imfill(imcomplement(threshdilate),'holes'));
```

When combining an erosion with a dilation, the results achieved were similar to the average filtering, but this was more complicated in the end so I used the average filter instead.

Next, the goal was to find the squares, in order to get correct grid positions for the final colour matrix for an image. My overall approach was to isolate the grid squares in the base image, and then find the squares as objects, identifying their centroid coordinates to use as reference for filling in the colour matrix.

Firstly, I used a Laplacian of a Gaussian filter to obtain the edges of the objects in the base image (figure 6a). The noise caused some edge artifacts within the objects, so I then simply performed a dilation and inverted the image to be left with figure 6b. With the image in this state, the objects that would be found would be the grid squares, the white region enclosing the grid (ie the background), and the little speck of noise in the bottom right circle.

Thus, I use bwconncomp and regionprops again to get the objects, and their area, extent and centroids. Then, I use the extent and area to filter the objects; the extent gives the ratio

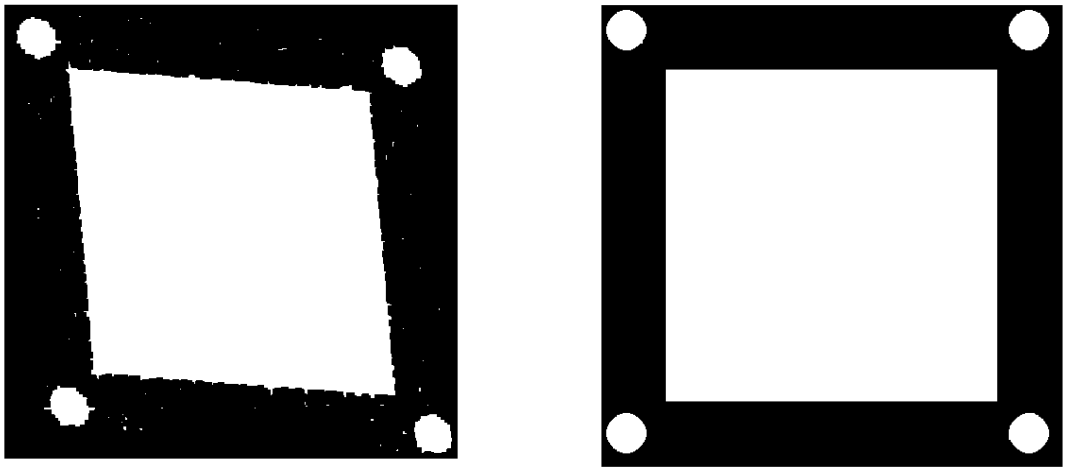

Figure 5 - a) proj1_2 dilate transform; b) noise_2 dilate transform

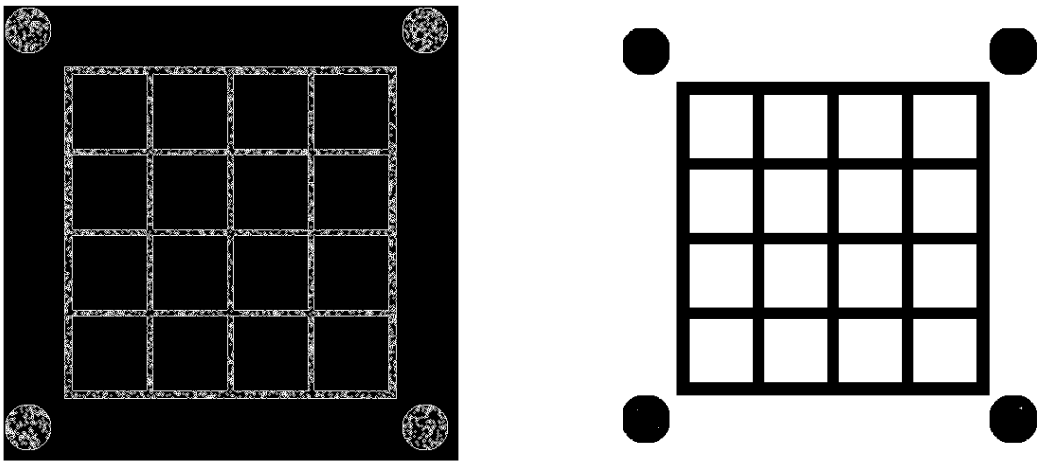

Figure 6 - a) LoG filtered base image; b) inverted dilated version of a

of the given object's area to its bounding box area, meaning that squares and almost squares will have an extent of nearly 1. I also filter on the area, ignoring objects with a very small area as these are likely noise. Looping through each object, I then add its centroid to the array of centroids if it is a square, obtaining a 16 entry array of centroid coordinates. Other properties and approaches to finding the squares could have achieved similar results, including using major/minor axis length ratio properties, or using Machine Learning tools to identify squares, however my approach felt the simplest technique to use and achieved the results I needed.

With the centre coordinates of each of the squares found, I get the correct grid position by looping over all the objects again, finding which entry in the centroid array has the minimum distance from the current object's centroid (ie a distance of (0,0)), and converting that index (whose range is 1-16) into an x,y grid position:

```
x = ceil(index/4);
y = index - (x-1)*4;
```

The x coordinate is simply the ceiling of dividing the index by 4, and the y coordinate is the index minus 4 times the column number minus 1, so for example 11 gives 11 – (3-1)*4 = 3.

The final task was then to successfully obtain the colours of the squares of a target image. I first converted the target image into Lab colour space, and median filtered the a and b channels to obtain a noise smoothed image; the median filter means that I'll obtain more instances of the target colour in each square region, rather than, say and average filter which would give an average value of the neighbourhood and so may make the colours harder to find. I chose the Lab colour space since its a and b channels range from green to red, and blue to yellow respectively, with white sitting in the middle of the two channels, so it can distinguish well between these colours that need to be identified.

The colour of each square region in the target image is then found by looping over each object as above in the base image, and using the pixels of the given square region as a mask over each of the a and b channels in the target image. This retrieves the a and b values just for the pixels in the given square region; I then take the mean of the values to get my average colour for the region:

```
Aav = mean(a(ismember(labelmatrix(CC),p))); % get average a colour
Bav = mean(b(ismember(labelmatrix(CC),p))); % get average b colour
```

And finally use the Euclidean distance between these values, and the set of already specified a, b values for the colours being searched for:

```
[mindist,colcode] = min(sqrt(sum((colours-avs).^2,2)));
```

The set of a,b values for the 5 colours being searched for were obtained by me checking the a,b values for a number of the provided images, and taking what seemed to be sensible middle values of a number of these images. This required some playing around to see what worked well for all the images, as a few of the noisy images, and some where some of the greens and yellows were quite similar, caused problems in identifying the colours. I also initially used the L channel when doing the colour comparisons, however this didn't really bring any added value, and would make the colour retrieval more dependent on the lightness/darkness of the colour, meaning it could perform worse at identifying the full range of values of a colour.

The results are returned as a 4x4 matrix, with each entry holding a number between 1 and 5; 1 is white, 2 is blue, 3 is yellow, 4 is green and 5 is red. To run the code, the reference image 'org_1.png' must be in a folder 'SimulatedImages' in the same directory as the code.

Results discussion

The results are presented in the appendix, tagged by the filename of their corresponding image. My function manages to guess every single image 100% correctly, and the script runs in only 12 seconds for all the images. This good speed performance I believe is down to my preference for using uncomplicated methods. However, the speed could definitely be

improved, since there is some redundancy in my code due to my using more for loops than was strictly necessary, mostly for clarity in the separate functions of the loops.

Meanwhile, the good square identification performance might partly be down to my 'overfitting' the data as it were, by basing the colours that I compare to on the colours of squares in the image. Furthermore, my function might not be robust to more distorted images, and those where the scale of the target image is different, as in some of the live image set, since I am mostly using the base image org_1 to create my square masks, so if the target image's squares didn't align well with the base image, I would be getting the average colour value of other regions of the image rather than just the desired square.

Live images discussion

To aid in detecting the live images, it would be a good idea to take a photo of a grid in a standard orientation and with ideal lighting. Let's call this the reference image.

Firstly, I will discuss dealing with the blurring of some of the live images. We can use a strong edge detector, such as the Sobel edge detector, to broadly detect the edges in the blurred images. There will be some gaps in the line edges detected due to the blurring, but dilation with sufficiently large structuring elements and then filling the image could allow us to obtain the whole grid as a single white square. From this we could use SIFT matching to transform the target image to a standard orientation and scale. This would involve using the reference image and the target image to transform the target to the correct format. From here we could use a technique similar to what I used in the main assignment; we use the reference image to obtain masks for each grid square location, and run this over the target image to get the region of pixels covered by each grid square, and average colour to find colours.

To deal with the inconsistent lighting images, where some regions of the image are well lit and others not so, we could try and utilise morphological operators. We could perform an opening operation, to obtain an estimate of the background, and then subtract this from the original image.

To deal with the images where the lighting colour seems to be different, we could utilise histogram normalisation. Taking our reference image, we obtain its histogram, and then obtain the target image's histogram, and map the intensity range of the target into the reference. This will remove the overall lighting colour of the target image and align it with the reference; it may cause some of the squares to become discoloured slightly but this could be accounted for with filtering.

To deal with the squashing in some of the images, it appears to me a little more difficult. I would suggest that perhaps a similar SIFT based feature matching and transformation as mentioned above could be used (however this may not actually work), perhaps obtaining more points than would usually be required to build a more robust transformation.

Once lighting and blurring has been dealt with, the aforementioned SIFT techniques or the same technique as I used in the main assignment could be used to deal with any perspective distorted images.

Appendix

Results:

```
 'noise_1.png'
ans =

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

 'noise_2.png'
ans =

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

'noise_3.png'
ans =

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

'noise_4.png'
ans =

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

'noise_5.png'
ans =

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

'org_1.png'
ans =

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

```
'org_2.png'
ans =

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

'org_3.png'
ans =

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

'org_4.png'
ans =

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

'org_5.png'
ans =

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

'proj1_1.png'
ans =

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

'proj1_2.png'
ans =

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

'proj1_3.png'
ans =

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

```
'proj1_4.png'
ans =

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

'proj1_5.png'
ans =

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

'proj2_1.png'
ans =

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

'proj2_2.png'
ans =

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

'proj2_3.png'
ans =

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

'proj2_4.png'
ans =

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

'proj2_5.png'
ans =

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

```
'proj_1.png'
ans =

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

'proj_2.png'
ans =

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

'proj_3.png'
ans =

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

'proj_4.png'
ans =

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

'proj_5.png'
ans =

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

'rot_1.png'
ans =

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

'rot_2.png'
ans =

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

```
'rot_3.png'
ans =

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

'rot_4.png'
ans =

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

'rot_5.png'
ans =

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

<u>Code</u>

```
function result = colourMatrix(filename)
%%%%%% Specify lab values of colours %%%%%%%
% Colours chosen based upon the colour of the noisy images %
white = [128,128]; % colour code 1
blue = [145,85]; % colour code 2
yellow = [116,190]; % colour code 3
green = [65,185]; % colour code 4
red = [170,150]; % colour code 5
colours = [white; blue; yellow; green; red];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Read in images %%%%%%%%
basicimg = imread('SimulatedImages/org_1.png');
targimg = imread(filename);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% Image registration %%%%%%%%%
%%% Filter images for image registration
% Average filter, then fill holes to be left with the corner circles, and the grid
% filled in as one big square
h = fspecial('average',3);
filtered_imgp=imfilter(targimg,h,'replicate');filtered_imgb=imfilter(basicimg,h,'replicate');
% filtered_imgp=medfilt1(im2double(projimg));
filtered_imgp=im2bw(filtered_imgp,0.4);filtered_imgb=im2bw(filtered_imgb,0.4);
filledimp = imfill(imcomplement(filtered_imgp),'holes');filledimb = imfill(imcomplement(filtered_imgb),'holes');
% figure(1)
% imshow(filledimp);

%%% Find regions
```

```
CC = bwconncomp(filledimp);CCb = bwconncomp(filledimb);
s  = regionprops(filledimp, 'Area','Centroid');sb  = regionprops(filledimb, 'Area','Centroid');

% Filter objects in base image on region prop
% Largest object will be the square grid, others will be the 4 circles
areasb = [sb.Area];
allcentroidsb = [sb.Centroid];allcentroidsb=vec2mat(allcentroidsb,2);
[maxvalb,maxidxb] = max(areasb);
fixedPoints = [];
% filledimb(ismember(labelmatrix(CCb),maxidxb)) = 0;
for p=1:CCb.NumObjects  %loop through each image
    if p ~= maxidxb % then object is one of the circles
       fixedPoints = [fixedPoints; allcentroidsb(p,:)]; %set the fixed points
    end

end

% Filter objects in target image on region prop
% Largest object will be the square grid, others will be the 4 circles
areas = [s.Area];
allcentroids = [s.Centroid];allcentroids=vec2mat(allcentroids,2);
[maxval,maxidx] = max(areas);
movingPoints = [];
filledimp(ismember(labelmatrix(CC),maxidx)) = 0;
% figure(5);
% imshow(filledimp);
for p=1:CC.NumObjects  %loop through each image
    if p ~= maxidx && s(p).Area > 10 % then object is one of the circles
       movingPoints = [movingPoints; allcentroids(p,:)]; %set the moving points
    end

end

% Transform the target image to the ref image
mytform = fitgeotrans(movingPoints, fixedPoints, 'projective');
projimg = imwarp(targimg,mytform,'OutputView',imref2d(size(basicimg)));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%% Filter images for square identfication %%%%%%%%%
C = makecform('srgb2lab');
% labimg = applycform(refimg, C);
w=fspecial('log',[3 3],0.5); % log filter for detecting edges
BW=im2bw(basicimg,0.1);BWref=im2bw(projimg,0.1);
filtered_img=imfilter(BW,w,'replicate');filtered_imgref=imfilter(BWref,w,'replicate');
filledim = imcomplement(imdilate(filtered_img,ones(7))); % dilate to fill circles and grid outlines
filledimref = imcomplement(imdilate(filtered_imgref,ones(7)));
h = fspecial('average',3);
filterim = imfilter(projimg,h,'replicate');
filterim=imcomplement(im2bw(filterim,0.4));
% figure(2);
% imshow(filledim);

%%% Median filter rgb channels to aid colour identification
medfilimg = projimg;
medfilimg(:,:,1) = medfilt2(medfilimg(:,:,1), [9 9]);
medfilimg(:,:,2) = medfilt2(medfilimg(:,:,2), [9 9]);
medfilimg(:,:,3) = medfilt2(medfilimg(:,:,3), [9 9]);
```

```
labimg = applycform(medfilimg, C); % convert to lab
% filledimref = imdilate(im2bw(medfilimg,0.5),ones(7));
% figure(2);
% imshow(labimg);
L = labimg(:,:,1);
a = labimg(:,:,2);
b = labimg(:,:,3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Find centroids of squares of base image %%%%%%%
s = regionprops(filledim, 'Centroid', 'Extent','Area');
CC = bwconncomp(filledim);

centroids = [];
% BW2=zeros(CC.ImageSize);
for p=1:CC.NumObjects  %loop through each object of base image
   if s(p).Extent < 0.9 || s(p).Area < 10 % squares will have extent ~1
       % regions with small area just noise so ignore
      % filledim(ismember(labelmatrix(CC),p)) = 0; % for display purposes
      continue;
   else
       centroids = [centroids; s(p).Centroid];
   end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Find the squares, read a and b values into grids %%%%
grida = zeros(4,4);
gridb = zeros(4,4);
gridl = zeros(4,4);
gridcol = zeros(4,4);

% imshowpair(refimg,medfilimg,'montage');
% filledimb(ismember(labelmatrix(CC),maxidxb)) = 0;

for p=1:CC.NumObjects  %loop through objects of original image
   if s(p).Extent < 0.9 || s(p).Area < 10 % squares will have extent ~1
%        filledimref(ismember(labelmatrix(CC),p)) = 0;
      continue;
   else
      temp = abs(s(p).Centroid - centroids); % find matching square in ref image
      tempmin = min(temp);
      [tf,index] = ismember(temp,tempmin, 'rows');
      index = find(index==1); % find index between 1-16 of ref square centroids

      Lav = mean(L(ismember(labelmatrix(CC),p)));
      Aav = mean(a(ismember(labelmatrix(CC),p))); % get average a colour of square
      Bav = mean(b(ismember(labelmatrix(CC),p))); % get average b colour of squar
      avs = [Aav, Bav];

      [mindist,colcode] = min(sqrt(sum((colours-avs).^2,2))); % find euclidean distance between ref colour and average square colour
      x = ceil(index/4);
      y = index - (x-1)*4; % convert to full row index to grid coords
      grida(y,x) = Aav;
      gridb(y,x) = Bav;
```

```
        gridl(y,x) = Lav;
        gridcol(y,x) = colcode;
    end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Display ref image squares, return final colour grid of results %%%%%
% figure(4);
% imshow(filledim);
result = gridcol;
% avals = grida
% bvals = gridb
end
```