

Image Processing Lab Project Report

1. Introduction

This project aims to imitate the game Lego's Life of George, where an image is displayed on the screen for a few seconds, then the player must memorize and reconstruct the Lego blocks as seen in the image. Given a set of images with different flaws, such as noise, rotation, and projections, a program was written using MATLAB to fix the images and return an array representing the 16 square blocks' colours as strings and as accurately as possible. In this report, images2 were used as the set of images for processing.

2. Approach and Functions

First, the images were uploaded into MATLAB, and each was examined separately to outline the approach that will be used. In the images2 folder, three types of different images are present, original images, noisy images, rotated images, and projected images. Now to achieve the main objective of the project, which is to return an array of strings representing the colours of the squares, three functions were written, which fix the orientation of the image, detect the circles and the last was to detect the colours of the 16 squares. In this section, the functionality of each function will be explained.

2.1 LoadImage Function

The LoadImage function takes in the image as type unit8, and either converts it into double if the input image was unit8 or returns an error message if the input type was not a unit8 image. Working with type double is preferred since it can increase precision on some operations done on an image in MATLAB.

2.2 findCircles Function

The findCircles function takes the double image, then it finds the coordinates of the circles (centres and radii). Also, it plots the identified circles' boundaries on the image using the MATLAB command viscircles.

- This method was chosen because MATLAB's imfindcircles function can accurately detect circles and return their radii and centers.
- All the circles' coordinates were identified correctly (100% accuracy), as viscircles function can draw the boundaries of the circles on all the images correctly.

2.3 correctImage Function

The main purpose of this function is to fix any rotations or projections in the image. To fix those images a reference point was needed, so the four circles' centroids in the original image (org_1.png) were used as a reference point to carry out the fitgeotrans MATLAB function, which fixes the orientation of an image based on a set of reference points, and the size of the output image is equal to the size of the original image. In this case, because a projected/rotated image is being fixed, the findCircles function was not able to detect the circles correctly, therefore, a manual circle identification and the following steps were carried out:

- The LoadImage function was used to load the image.
- The image was converted to binary, and filtering and filling out the holes was carried out.
- Bwconncomp and regionprops were used to find properties of the objects in the image (Areas and Centroids).
- A for loop was used to separate the circles by looping over the objects and separating them based on area.
- Bwlabel was used to label the circles.
- Circles' centroids were used as the varying points for the projected/rotated image and fixed points of the org_1.png image were the static points.
- Fitgeotrans was then used with projective transformation to carry out the pixel transformation based on the varying and static centroids of the images. And org_1.png was used as a size reference.
- Imwrap function was used to carry out the transformation described above, and that is how projections and rotations were fixed.
- Choosing this method resulted in 100% accuracy for the transformation/fixation of the projections and rotations in all the images. Taking the centers of the circles of the org_1 image as reference, and transforming other images accordingly was the first idea that came to mind. Regardless of the orientation, as some images might be flipped. Also, the process is fully automated, just pass the image, and the function will fix it.

2.4 findColours Function

The findColours function takes the double image and returns a string matrix of colours present in each of the 16 squares, the colours present in this case were blue, red, green, yellow, and white. Some further image pre-processing was required in this function as some images were too noisy, thus making it hard for the colours to be detected correctly. The output of the colours in two images (rot_2, proj_1) was not 100% accurate. Here is a description of the steps carried out to obtain the desired output.

- Some further erosion, thresholding and filtering along with an increase in contrast, was done to denoise the image and increase colour quality for easier recognition.
- The image was converted to grayscale, surrounding white space was removed, also some pixels were removed using bwareopen function to give better results, which is also known as area opening.

- The image was then segmented using bwlabel function, then a for loop was constructed to loop over the squares and get the mean colours of the pixels in each square in type double representation. i.e. [1,1,1] which is RGB colour space.
- The centroid of each square was then found using regionprops, the min and max centroids limits were found, and the centroids then were rounded between the min and max values (1,4).
- An index was then assigned to each square using sub2ind and the colour samples were reordered.
- Color names and references in RGB were then specified to their corresponding strings in two arrays, colorNames and colorReferences.
- Then the closest colour distance in RGB for each square was calculated and assigned to the index of the closest match.
- Finally, the function looks up colour names and returns a MATLAB cell array of the strings of the colours representing each square.

2.5 colourMatrix Function

The colourMatrix function calls the three functions mentioned above on each image to produce the desired matrix of strings colours representing each square in the image.

- The function correctly identified the colours in 28 images out of 30.
- Two images (rot_2 and proj_1) had three squares mistaken green for yellow, so 3/16 squares = 18.75% error rate for these two images.

3. Real Photos

As it can be noticed, the images solved by the function (images_2), were a set of simulated images. After examining the set of real Photos, the following suggestions to improve the quality of the photos are made:

- Fixing the angle of photography, choosing the same angle for all the images would be better.
- Use the same light source, ensuring good lighting for the pictures, as all of them suffer from shadow effects.
- Use a better camera to avoid blurring, and give better pixel quality images, the camera and the lighting are the most important factors in yielding a good quality image.
- Make sure the paper on which the image is on is straight and put the image on a flat surface.

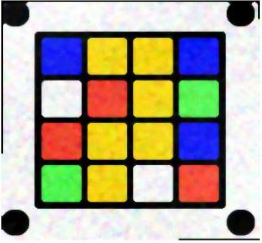
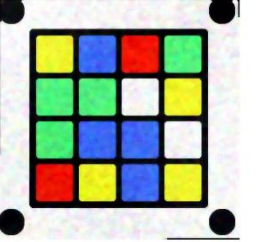
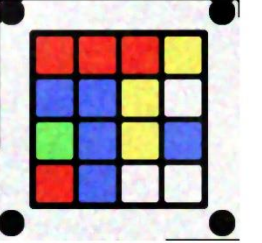
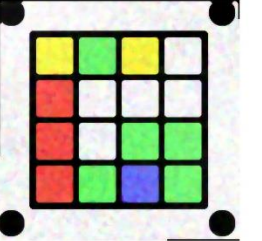
4. Conclusion

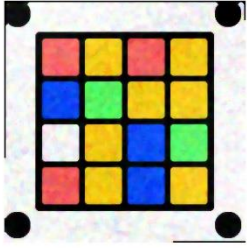
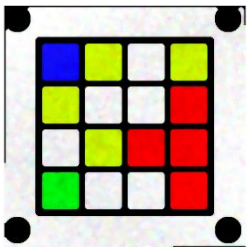
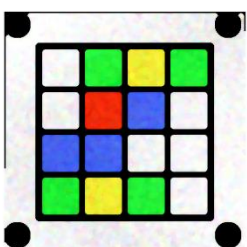
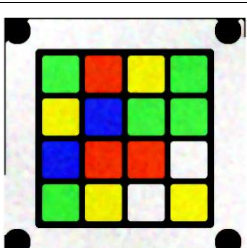
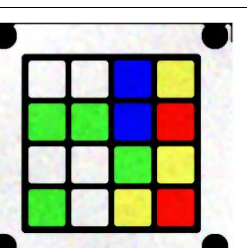
The main objective of this report which was identifying the colours of the squares in each image using MATLAB was successfully fulfilled for 28 images out of 30 having 100 % accuracy, while two images (rot_2 and proj_1) had an error rate of 18.75% in colour recognition by misclassifying the colours of three squares. Therefore, the total accuracy is $28/30 = 93.3\%$. Regarding circle detection and fixing the orientation of the rotated/projected images the function had an overall 100% success rate.

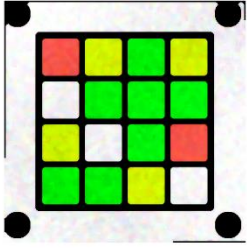
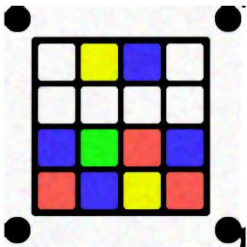
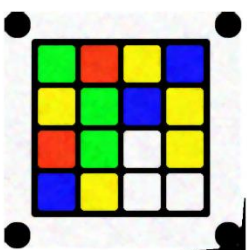
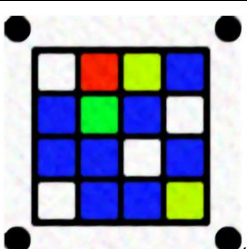
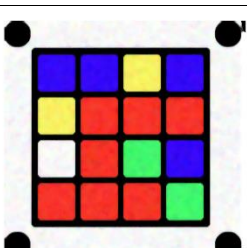
Appendix

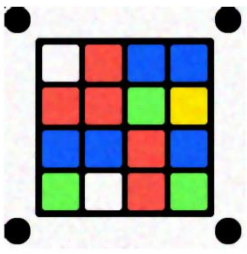

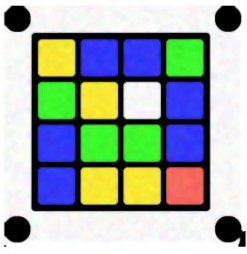
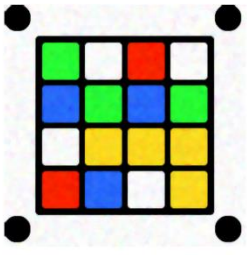
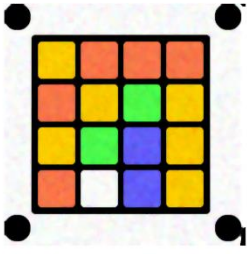
Table of Results

The following table shows the results for all the images after applying the colourMatrix function on each image.

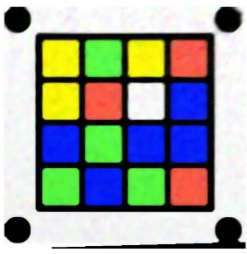
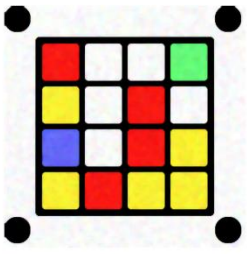
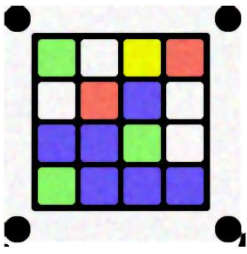
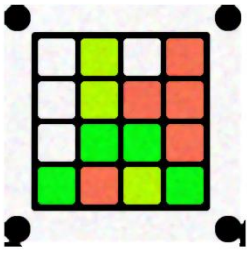
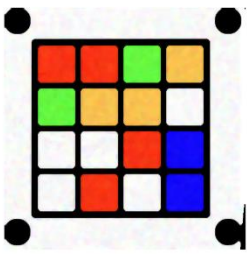
File Name	Image	Output	Success/Notes
Noise_1		<pre>{'blue' } {'yellow'} {'yellow'} {'blue' } {'white' } {'red' } {'yellow'} {'green'} {'red' } {'yellow'} {'yellow'} {'blue' } {'green'} {'yellow'} {'white' } {'red' }</pre>	Successful
Noise_2		<pre>{'yellow'} {'blue' } {'red' } {'green' } {'green' } {'green' } {'white' } {'yellow'} {'green' } {'blue' } {'blue' } {'white' } {'red' } {'yellow'} {'blue' } {'yellow' }</pre>	Successful
Noise_3		<pre>{'red' } {'red' } {'red' } {'yellow'} {'blue' } {'blue' } {'yellow'} {'white' } {'green' } {'blue' } {'yellow'} {'blue' } {'red' } {'blue' } {'white' } {'white' }</pre>	Successful
Noise_4		<pre>{'yellow'} {'green'} {'yellow'} {'white'} {'red' } {'white' } {'white' } {'white'} {'red' } {'white' } {'green' } {'green'} {'red' } {'green' } {'blue' } {'green' }</pre>	Successful

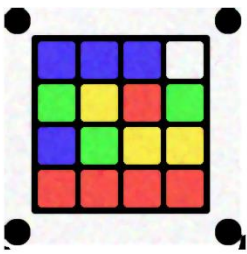
Noise_5		{'red' } {'yellow' } {'red' } {'yellow' } {'blue' } {'green' } {'yellow' } {'yellow' } {'white' } {'yellow' } {'blue' } {'green' } {'red' } {'yellow' } {'blue' } {'yellow' }	Successful
Org_1		{'blue' } {'yellow' } {'white' } {'yellow' } {'yellow' } {'white' } {'white' } {'red' } {'white' } {'yellow' } {'red' } {'red' } {'green' } {'white' } {'white' } {'red' }	Successful
Org_2		{'white' } {'green' } {'yellow' } {'green' } {'white' } {'red' } {'blue' } {'white' } {'blue' } {'blue' } {'white' } {'white' } {'green' } {'yellow' } {'green' } {'white' }	Successful
Org_3		{'green' } {'red' } {'yellow' } {'green' } {'yellow' } {'blue' } {'green' } {'green' } {'blue' } {'red' } {'red' } {'white' } {'green' } {'yellow' } {'white' } {'yellow' }	Successful
Org_4		{'white' } {'white' } {'blue' } {'yellow' } {'green' } {'green' } {'blue' } {'red' } {'white' } {'white' } {'green' } {'yellow' } {'green' } {'white' } {'yellow' } {'red' }	Successful

Org_5		{'red' } {'yellow'} {'green' } {'yellow'} {'white' } {'green' } {'green' } {'green' } {'yellow' } {'white' } {'green' } {'red' } {'green' } {'green' } {'yellow' } {'white' }	Successful
Proj1_1		{'white' } {'yellow' } {'blue' } {'white' } {'white' } {'white' } {'white' } {'white' } {'blue' } {'green' } {'red' } {'blue' } {'red' } {'blue' } {'yellow' } {'red' }	Successful
Proj1_2		{'green' } {'red' } {'yellow' } {'blue' } {'yellow' } {'green' } {'blue' } {'yellow' } {'red' } {'green' } {'white' } {'yellow' } {'blue' } {'yellow' } {'white' } {'white' }	Successful
Proj1_3		{'white' } {'red' } {'yellow' } {'blue' } {'blue' } {'green' } {'blue' } {'white' } {'blue' } {'blue' } {'white' } {'blue' } {'white' } {'blue' } {'blue' } {'yellow' }	Successful
Proj1_4		{'blue' } {'blue' } {'yellow' } {'blue' } {'yellow' } {'red' } {'red' } {'red' } {'white' } {'red' } {'green' } {'blue' } {'red' } {'red' } {'red' } {'green' }	Successful

Proj1_5		{'white'} {'red'} {'blue'} {'blue'} {'red'} {'red'} {'green'} {'yellow'} {'blue'} {'blue'} {'red'} {'blue'} {'green'} {'white'} {'red'} {'green'}	Successful
Proj2_1		{'red'} {'red'} {'green'} {'yellow'} {'white'} {'blue'} {'yellow'} {'blue'} {'yellow'} {'yellow'} {'blue'} {'white'} {'red'} {'green'} {'white'} {'green'}	Successful
Proj2_2		{'yellow'} {'blue'} {'blue'} {'green'} {'green'} {'yellow'} {'white'} {'blue'} {'blue'} {'green'} {'green'} {'blue'} {'blue'} {'yellow'} {'yellow'} {'red'}	Successful
Proj2_3		{'green'} {'white'} {'red'} {'white'} {'blue'} {'green'} {'blue'} {'green'} {'white'} {'yellow'} {'yellow'} {'yellow'} {'red'} {'blue'} {'white'} {'yellow'}	Successful
Proj2_4		{'yellow'} {'red'} {'red'} {'red'} {'red'} {'yellow'} {'green'} {'yellow'} {'yellow'} {'green'} {'blue'} {'yellow'} {'red'} {'white'} {'blue'} {'yellow'}	Successful

Proj2_5		{'green'} {'red'} {'green'} {'white'} {'blue'} {'blue'} {'green'} {'white'} {'red'} {'white'} {'red'} {'yellow'} {'red'} {'yellow'} {'yellow'} {'white'}	Successful
Proj_1		{'white'} {'white'} {'blue'} {'yellow'} {'yellow'} {'yellow'} {'yellow'} {'white'} {'yellow'} {'yellow'} {'yellow'} {'blue'} {'yellow'} {'yellow'} {'red'} {'yellow'}	Unsuccessful in detecting green, mistaken for yellow. 3/16=18.75% error rate.
Proj_2		{'yellow'} {'yellow'} {'red'} {'red'} {'yellow'} {'white'} {'green'} {'white'} {'yellow'} {'red'} {'white'} {'red'} {'yellow'} {'yellow'} {'blue'} {'green'}	Successful
Proj_3		{'white'} {'green'} {'white'} {'blue'} {'yellow'} {'red'} {'red'} {'yellow'} {'green'} {'blue'} {'green'} {'yellow'} {'blue'} {'red'} {'yellow'} {'blue'}	Successful
Proj_4		{'blue'} {'red'} {'yellow'} {'yellow'} {'red'} {'green'} {'blue'} {'red'} {'blue'} {'yellow'} {'yellow'} {'blue'} {'green'} {'green'} {'green'} {'yellow'}	Successful

Proj_5		{'yellow'} {'green'} {'yellow'} {'red'} {'yellow'} {'red'} {'white'} {'blue'} {'blue'} {'green'} {'blue'} {'blue'} {'green'} {'blue'} {'green'} {'red'}	Successful
Rot_1		{'red'} {'white'} {'white'} {'green'} {'yellow'} {'white'} {'red'} {'white'} {'blue'} {'white'} {'red'} {'yellow'} {'yellow'} {'red'} {'yellow'} {'yellow'}	Successful
Rot_2		{'yellow'} {'white'} {'yellow'} {'red'} {'white'} {'red'} {'blue'} {'white'} {'blue'} {'blue'} {'yellow'} {'white'} {'yellow'} {'blue'} {'blue'} {'blue'}	Unsuccessful in detecting green, mistaken for yellow. $3/16=18.75\%$ error rate.
Rot_3		{'white'} {'yellow'} {'white'} {'red'} {'white'} {'yellow'} {'red'} {'red'} {'white'} {'green'} {'green'} {'red'} {'green'} {'red'} {'yellow'} {'green'}	Here the function classified the light green as yellow. Even for humans it debatable if these squares should be yellow or green, so I don't know if this is an error or not.
Rot_4		{'red'} {'red'} {'green'} {'yellow'} {'green'} {'yellow'} {'yellow'} {'white'} {'white'} {'white'} {'red'} {'blue'} {'white'} {'red'} {'white'} {'blue'}	Successful

Rot_5		{'blue'} {'blue'} {'blue'} {'white'} {'green'} {'yellow'} {'red'} {'green'} {'blue'} {'green'} {'yellow'} {'yellow'} {'red'} {'red'} {'red'} {'red'}	Successful
-------	---	---	------------

Note: for rot_3, three squares had a light green/ yellowish colour, so even if humans were asked to classify the colours of these three squares it would be debatable, thus, for this case, I didn't know if this should be considered an error done by the function, as this colour could be classified as light green or yellow.

MATLAB Code

1) LoadImage Function:

```
function [ image] = LoadImage( filename )
%function to load the image and convert it to type double
image=imread(filename); %read the image file
if isa(image,'uint8')%check if the image is type uint8
    image=double(image)/255; %change the image type to double
else
    error('The image is of unknown type');%return error message if image is
not type uint8
end
end
```

2) findColours Function:

```
function colours=findColours(image)
%function to find the colours of the squares
image = imerode(image,ones(5));%eroding the image
image = medfilt3(image,[11 11 1]); % median filter to suppress noise
image = imadjust(image,stretchlim(image,0.05)); % increase contrast of the
image
figure(),imshow(image)
imageMask= rgb2gray(image)>0.08; %converting the image into greyscale
imageMask = bwareaopen(imageMask,100); % removing positive specks
```

```

imageMask = ~bwareaopen(~imageMask,100); % removing negative specks
imageMask = imclearborder(imageMask); % removing outer white region
imageMask = imerode(imageMask,ones(10)); % eroding to exclude edge effects
% segmenting the image
[L N] = bwlabel(imageMask);
% getting average color in each image mask region
maskColors = zeros(N,3);
for p = 1:N % stepping throughout the patches
    imgmask = L==p;
    mask= image(imgmask(:,:, [1 1 1]));
    maskColors(p,:) = mean(reshape(mask,[],3),1);
end
% trying to snap the centers to a grid
Stats = regionprops(imageMask, 'centroid'); %obtaining the centroids
Centroids = vertcat(Stats.Centroid);%concatenatoin
centroidlimits = [min(Centroids,[],1); max(Centroids,[],1)];
Centroids= round((Centroids-centroidlimits(1,:))./range(centroidlimits,1)*3 + 1);
% reordering colour samples
index = sub2ind([4 4],Centroids(:,2),Centroids(:,1));
maskColors(index,:) = maskColors;
% specify colour names and their references
colorNames = {'white','red','green','blue','yellow'};
colorReferences = [1 1 1; 1 0 0; 0 1 0; 0 0 1; 1 1 0];
% finding color distances in RGB
distance = maskColors - permute(colorReferences,[3 2 1]);
distance = squeeze(sum(distance.^2,2));
% finding index of closest match for each patch
[~,index] = min(distance,[],2);
% looking up colour names and returning a matrix of colour names
colours = reshape(colorNames(index),4,4)
end

```

3) findCircles Function:

```

function [centers,radii] = findCircles(image)

```

```

[Centers,radii]= imfindcircles(image,[20
25], 'ObjectPolarity','dark','Sensitivity',0.92,'Method','twostage'); %Detect
and find the Circles with their centers and Radii.

circle= viscircles(Centers,radii) %Draw the Cricles boundaries on the image.
centers=[Centers] %centers of the circles
radii=[radii];%raddii of the circles
end

```

4) correclImage Function:

```

function correctedImage= correctImage(filename)
%function that fix the projections/Rotations
orgImage=LoadImage(filename);
BandWimage=~im2bw(orgImage,0.5);%converting the image to black and white
BandWfill=imfill(BandWimage,'holes');%filling any holes in the image
medianFilt=medfilt2(BandWfill);%filtering using a median filter
connectedComps=bwconncomp(medianFilt);%finding the conncted components in
the image
stats=regionprops(medianFilt,'Area','Centroid');%getting stats of the
image(area,centroid)
Areas=[stats.Area];
Circles=zeros(connectedComps.ImageSize);%Isolating the circles
for p = 1:connectedComps.NumObjects
    if stats(p).Area<max(Areas)
        Circles(connectedComps.PixelIdxList{p}) = 1;
    end
end
%figure,imshow(Circles,[0,1]), title("Input's Image Isolated Circles")
circlesLabeled=bwlabel(Circles, 8);%returning the label matrix L for 8
conncted objects (labeling the circles).
circlesProps=regionprops(circlesLabeled,'Area','Centroid');%Stats of the
circles
circlesCentroids=[circlesProps.Centroid];%centroids of the circles
varyingPoints=reshape(circlesCentroids,2,[]);%seting centroids of the image
as reference points for transformation.
MovingPointsT=transpose(varyingPoints);%transposing the matrix to get it in
the correct form
staticPoints=flip(findCircles(LoadImage('org_1.png')));%Setting the fixed
points as centroids of org_1.png

```

```

transformation=fitgeotrans(MovingPointsT,staticPoints,'Projective');%applying
the transformation
Reference=imref2d(size(LoadImage('org_1.png')));%referencing the size of
org_1.png
correctedImage=imwarp(orgImage,transformation,'OutputView',Reference);%determ
ining pixel values for output image by mapping locations of the output to the
input image
end

```

5) colourMatrix Function:

```

%Function which calls other functions
%reading the Image and converting it to type double, then fixing the image
%for any projections or rotations, also finding the circles
correctedImage=correctImage(filename);%fixing projections and rotations
figure(),imshow(correctedImage)
findCircles(correctedImage);%finding the circles
%Find The Colours of the Squares
colorRecognition=findColours(correctedImage);
finalImage=colorRecognition;
end

```

Resources:

- Image processing Labs
- Mathworks for guidance on how to use several MATLAB functions.
- Video explaining how to fix projections/ Rotations
<https://www.youtube.com/watch?v=Dg05eIN5InM&t=1697s>
- Image Processing tutorial videos: <https://www.youtube.com/watch?v=DDh8-2HkiVs&list=PL1pxneANaikCO1-Z0XTaljLR3SE8tgRXY&index=4>