

Image Processing- Lab Project Report

by Pranov Sarath

April 2023

Introduction:

The lab project involved loading an image from the hard disk, that contained a set of squares arranged in a 4x4 matrix fashion and the task was to identify the colours of each matrix and output them as a 4x4 matrix of strings. Each element in the matrix is supposed to represent the colour of the square in the position. What makes this task challenging is that the images contain noise and the squares matrix may be rotated, scaled, warped etc. There are 4 black circles in the image which denote the 4 corners of the image.

Task Breakdown:

I will describe my approach to solve this problem using MATLAB and provide a walkthrough of the functions that I wrote to accomplish this task.

1) Loading the image - **loadImage(filename)**

The first task in the image processing task is to load the source file or image. This task is performed by the function 'loadImage' which takes the 'filename' as an input parameter and converts the RGB image from an integer datatype to a double-precision floating point data type and returns it.

2) Denoising the image - **meanFilterImage(image, filterSize)**

Most of the source RGB images used for this task contained a lot of salt and pepper noise. The most straightforward way to get rid of this noise was to use a median filter. But this was not very effective and also darkened the images a little bit. After several attempts, I decided to stick with the mean filter. The mean filter takes the mean of the pixel intensities of the specified number of surrounding pixels and smoothens the image and gets rid of noise. This meanFilterImage MATLAB function takes 2 parameters as inputs. One is the image and the other is the size of the mean filter to be defined.

3) Finding the circle coordinates - **findCircles(referenceImageDouble)**

The next step was to load the double image returned by the `loadImage` function and tries to identify the coordinates of the centroids of each one of the 4 circles. This function takes the image as input, automatically identifies a threshold. Then, I converted the image to black and white using the built-in MATLAB function `'imbinarize'` based on the threshold. I took the complement of this image, so that the black circles would now appear as bright white spots and performed a fill operation to remove the holes in the image.

Once this was done, I used another MATLAB built-in function called – `'imfindcircles'` to identify the bright white circles in the image. This function utilises a circular Hough transform to find circles in an image. After setting the minimum and maximum radii of the circle between 10 and 200 pixels, the circles were identified and then centroid of each of the circles were calculated. These coordinates are returned as a 4x2 array by the function.

4) Correcting the image: **`correctImage(circleCoordinates, image)`**

Once the current image's circle centroids have been identified, the next step is to correct the image's rotation/warping/scaling etc and bring it back into the original space. This makes further down the line processing steps easier. To do this, I use the `'org_1.png'` file as a reference image, as this image is not very polluted by noise and the circles and squares are in their standard positions. I find the coordinates of the centroid of this image and use the MATLAB built-in function `'fitgeotrans'` to find out the transform matrix that will bring the current image that may have been warped, rotated or scaled back into its original space. Once the transform matrix has been calculated, I used the `'imwarp'` function to transform the input image back to its original space with the help of the transformation matrix.

5) Identifying the centres of squares in the image: **`findSquareCenters()`**

In this function, I used the image – `'org_1.png'` as reference, and converted the RGB image to a binary image after denoising the image. Once this was done, I used the `'bwareaopen'` MATLAB function to remove small, isolated bright and dark spots less than 100 pixels in area from the image. This helps to denoise the image even further. I then used the `'imclearborder'` MATLAB function to remove objects that are connected to the border to remove the black circles. Then I performed a morphological erosion operation using a 10x10 square structuring element to identify all the squares. Once this was done, the centroids of the squares were identified, and the coordinates were sorted. The centroids were then returned as output by the function.

6) Identifying the colours of the squares and generating output colour matrix - **`detectSquareColours(rgbImage, squareCenters)`**

This function takes the `rgbImage` and the coordinates of the square centres as input and returns the 4x4 matrix of strings containing the colour of each square. To do this, I converted the RGB image to the LAB colour space. This splits the images into 3 channels, L, A and B. I used some trial-and-error methods to find out the maximum and minimum thresholds for each of the 5 colours – red, blue, green, white and yellow. Once this was done, I constructed a square around the centre of each square which was 50 pixels wide and computed the mean values for the 3 channels. If the computed mean value was between the minimum and maximum thresholds of any of the colours, the square is assigned that particular colour. The colours were added to a list and then converted into a 4x4 matrix.

7) Orchestrating the colour detection process – **`findColours(filename)`**

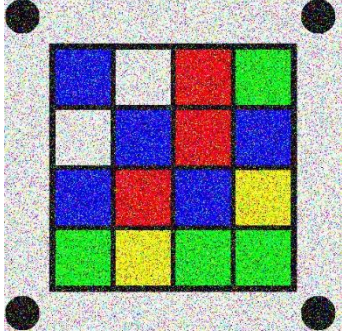
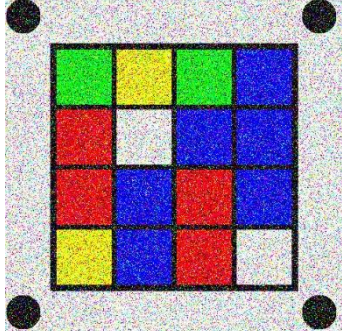
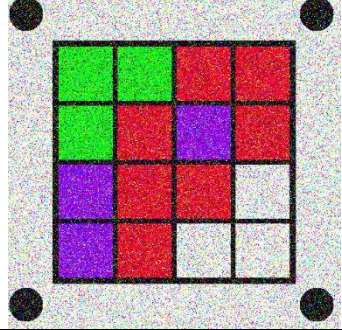
This function is used to orchestrate the entire process and call each of the functions described above for the colour detection and classification workflow.

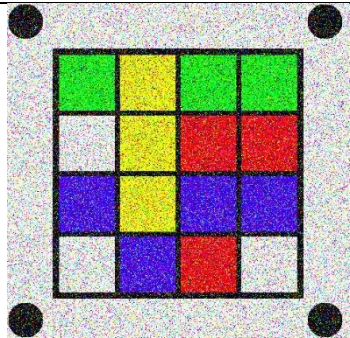
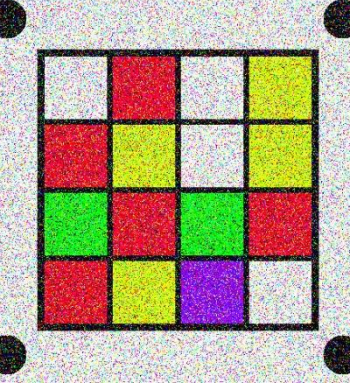
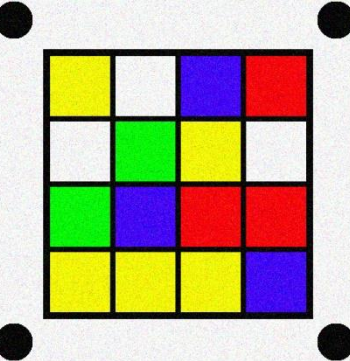
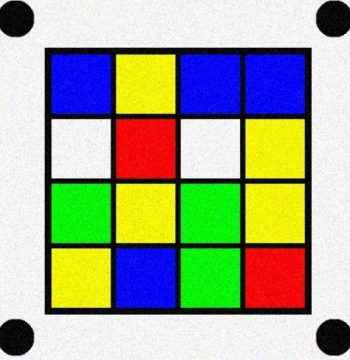
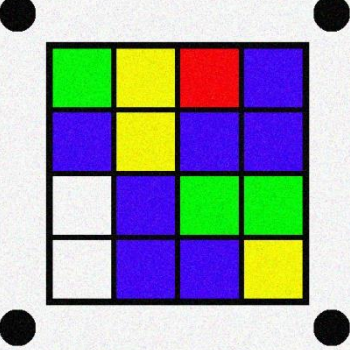
Apart from this, the helper functions that were supplied was used to read all the images in the 'images' folder, to process each image one by one and validate the output produced against the correct colour data.

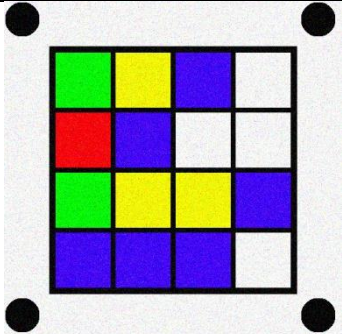
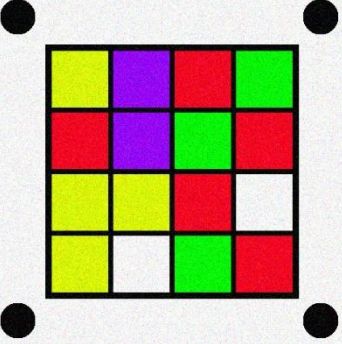
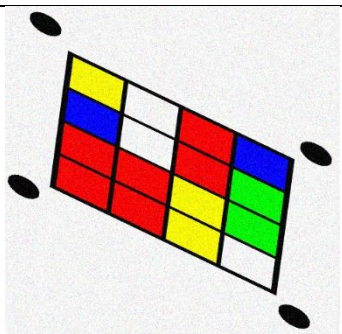
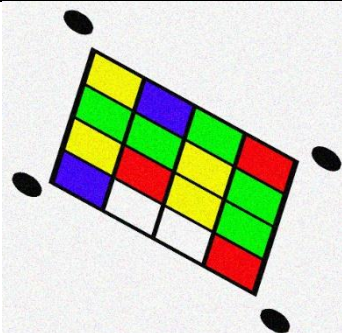
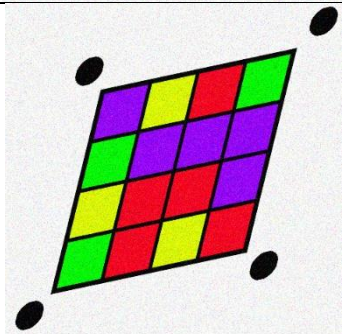
Conclusion:

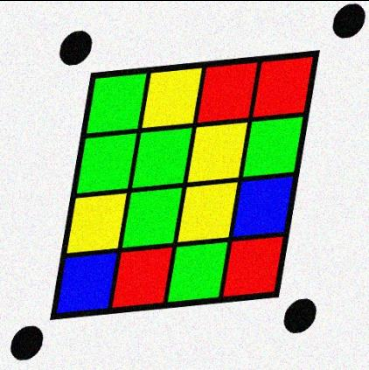
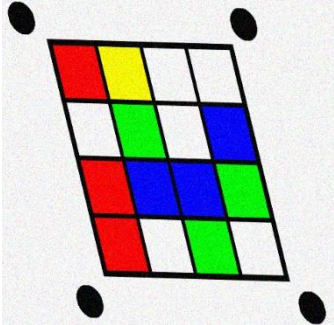
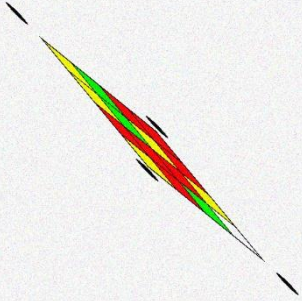
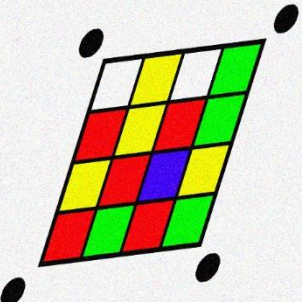
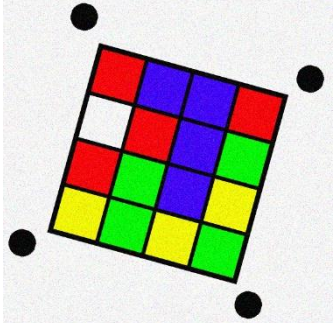
The colour detection task was carried out successfully and achieved a final score of 83.806818. The code execution failed for the image 'proj_6.png' as the 'imfindcircles' function was unable to identify the circles in the image as they were very heavily warped. The function 'imfindcircles' has very low accuracy for detecting circles under 5 pixels in size. All other images were handled quite well and the colours were identified as expected.

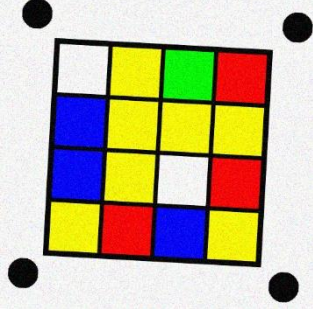
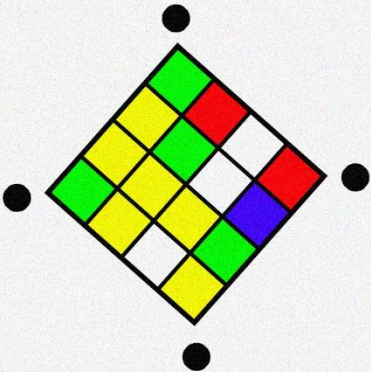
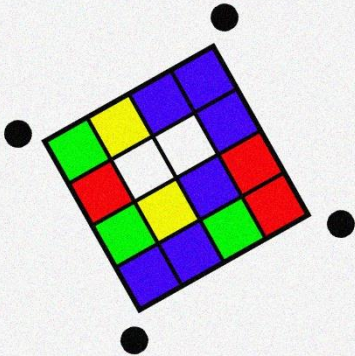
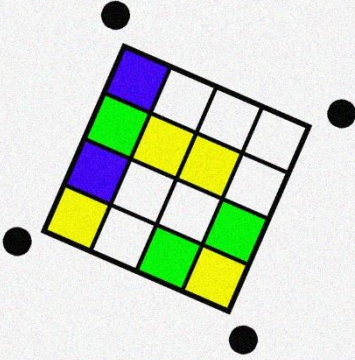
Table of Inputs and Output:

Filename	Image	Output	Success	Notes
noise_1.png		<i>b w b g</i> <i>w b r y</i> <i>r r b g</i> <i>g b y g</i>	Passed	Score - 100
noise_2.png		<i>g y g b</i> <i>r w b b</i> <i>r b r b</i> <i>y b r w</i>	Passed	Score - 100
noise_3.png		<i>g g r r</i> <i>g r b r</i> <i>b r r w</i> <i>b r w w</i>	Passed	Score - 100

noise_4.png		<i>g w b w</i> <i>y y y b</i> <i>g r b r</i> <i>h r b w</i>	Passed	Score - 100
noise_5.png		<i>w r g r</i> <i>r y r y</i> <i>w w g b</i> <i>y y r w</i>	Partial Match	Score – 31.25 Code intentionally reads wrong file? Comparing with noise_1.png
org_1.png		<i>y w b r</i> <i>w g y w</i> <i>g b r r</i> <i>y y y b</i>	Passed	Score - 100
org_2.png		<i>b y b b</i> <i>w r w y</i> <i>g y g y</i> <i>y b g r</i>	Passed	Score - 100
org_3.png		<i>g y r b</i> <i>b y b b</i> <i>w b g g</i> <i>w b b y</i>	Passed	Score - 100

org_4.png		<i>g y b w</i> <i>r b w w</i> <i>g y y b</i> <i>b b b w</i>	Passed	Score - 100
org_5.png		<i>y b r g</i> <i>r b g r</i> <i>y y r w</i> <i>y w g r</i>	Partial Match	Score – 31.25 Code intentionally reads wrong file? Comparing with noise_1.png
proj_1.png		<i>y w r b</i> <i>b w r g</i> <i>r r y g</i> <i>r r y w</i>	Passed	Score - 100
proj_2.png		<i>y b g r</i> <i>g g y g</i> <i>y r y g</i> <i>b w w r</i>	Passed	Score - 100
proj_3.png		<i>b y r g</i> <i>g b b b</i> <i>y r r b</i> <i>g r y r</i>	Passed	Score - 100

proj_4.png		<pre> g y r r g g y g y g y b b r g r </pre>	Passed	Score - 100
proj_5.png		<pre> r w r r y g b w w w b g w b g w </pre>	Partial Match	Score – 25 Code intentionally reads wrong file? Comparing with noise_1.png
proj_6.png			Failed	Failed to find circles and code threw an error. Handled by try catch block.
proj_7.png		<pre> w r y r y y r g w r b r g g y g </pre>	Passed	Score - 100
rot_1.png		<pre> r b b r w r b g r g b y y g y g </pre>	Passed	Score - 100

rot_2.png		<i>w y g r</i> <i>b y y y</i> <i>b y w r</i> <i>y r b y</i>	Passed	Score - 100
rot_3.png		<i>g r w r</i> <i>y g w b</i> <i>y y y g</i> <i>g y w y</i>	Partial Match	Score - 31.25 Comparing with noise_1.png
rot_4.png		<i>g r g b</i> <i>y w y b</i> <i>b w b g</i> <i>b b r r</i>	Passed	Score - 100
rot_5.png		<i>b w w w</i> <i>g y y w</i> <i>b w w g</i> <i>y w g y</i>	Passed	Score - 100

Filename	Remarks
IMAG0032.jpg	The image has very dark and bright parts. The image can also be cropped a little bit to only capture the area of interest. We can do histogram equalisation to get better brightness.
IMAG00033.jpg	The image is rotated, but the brightness is more even than the last image. We can crop the image and find the transform matrix to rotate the image back to its original space.

IMAG0034.jpg	Since the image is really bright, we can perform histogram equalisation, gamma correction or thresholding to get a more equalised image
IMAG0035.jpg	Histogram equalization. Cropping and scaling.
IMAG0036.jpg	Cropping, gamma correction, brightness and contrast adjustment
IMAG0037.jpg	Rotation, Cropping, scaling.
IMAG0038.jpg	Cropping, rotation, histogram equalisation, gamma correction, brightness and contrast adjustment.
IMAG0041.jpg	This is a challenging task, as the image is very dark and the piece of paper is a little bit warped and crumpled. It will pose a challenge to identify the square boundaries/shapes in the image.
IMAG0042.jpg	This image has a dominant blue colour tone. So the image might require colour correction. It will also need to be cropped and the blur also needs to be reduced.
IMAG0044.jpg	This will be a serious challenge to process as the image is very blurred.

Code Appendix:

```

clc, clearvars, close all

% find all file png files
D=dir('images/*.png');

score = [];

%load and process each file in turn.
for ind=1:length(D)

    %name of png file
    filename = fullfile(D(ind).folder,D(ind).name);

    %name of answer file .mat

    [folder, baseFileName, ~] = fileparts(filename);
    mat_filename = fullfile(folder, sprintf('%s.mat',baseFileName));

    %test result

    % load wrong file to test every for error detection
    if mod(ind,5)==0
        filename = fullfile(D(3).folder,D(1).name);
    end

```



```

%call the actual findColours function - this is the function that the
% student needs to write
try
    res = findColours(filename);
catch ME
    % display the error message
    disp(['Error occurred: ', ME.message]);
end

% check the answers.
mm = check_answer(res,mat_filename);

score=[score,mm];
%if mm ~= 100.00
%    mat_filename
%end

end
%print out the score.
str= repmat('%0.2f ', 1, length(score));
fprintf('Score is: ');
fprintf(str,score);
fprintf('\nMean score %f\n',mean(score));

%s = findColours('images/org_1.png');

function colorMatrix = findColours(filename)
% Function to find the colors of the squares in the image
% Inputs:
%     filename - string variable that contains the path and filename
% Outputs:
%     colorMatrix - a 4x4 matrix containing the colors of each square in the
%     image

% Load the raw image file
image = loadImage(filename);
%figure(1)
%imshow(image)
% Find the coordinates of the 4 edge circles in the image
circleCoordinates = findCircles(image);
% Project the rotated image back to its original space
% based on the standard centroid coordinates
%figure(2)
correctedImage = correctImage(circleCoordinates,image);
%imshow(correctedImage)
% Calculate the centroid coordinates of each square in a standard unrotated
% image
referenceSquareCenters = findSquareCenters();

% Perform mean filtering on the image to remove salt and pepper and any
% other type of noise
image_dn = meanFilterImage(correctedImage, 6);

% Display the corrected, denoised image with the square centroids
% highlighted
% figure(21)
% imshow(image_dn)
% hold on;

```

```

% scatter(referenceSquareCenters(:,1), referenceSquareCenters(:,2), 'w',
'filled');
% hold off;
colorMatrix = detectSquareColours(image_dn, referenceSquareCenters);
colorMatrix
% imshow(image_dn, []);
end

function colorImageDouble = loadImage(filename)
% Function to load the image from a given filename
% Inputs:
%   filename- string containing the path and filename of the image
% Outputs:
%   colorImageDouble: The RGB image converted into double format
colorImage = imread(filename);
% convert the image to double
colorImageDouble = im2double(colorImage);
end

function centroids = findCircles(image)
% Function to find the 4 circles denoting the 4 corners of the image
% Inputs:
%   image: a double representation of the RGB image
% Outputs:
%   centroids: An array containing the x & y coordinates of the 4 circle
%   centroids in the given image

% Convert the image to grayscale
grayImage = rgb2gray(image);
% Apply a median filter to remove any remaining noise
filteredImage = medfilt2(grayImage);
% Automatically calculate the threshold from the image
% and use it to convert the image into black and white
threshold = graythresh(filteredImage);
binaryImage = imbinarize(filteredImage, threshold);

% Invert the binary image so that black pixels are now white and vice versa
invertedImage = imcomplement(binaryImage);
% Fill in any holes in the binary image
filledImage = imfill(invertedImage, 'holes');

% Use built-in MATLAB function to detect circles
% Second parameter of the function determines the minimum and maximum radii
% of the circles
% The function will detect circles brighter than the background and
% sensitivity is set high to detect maximum circles.
[centers, radii] = imfindcircles(filledImage, [10 200], 'ObjectPolarity',
'bright', 'Sensitivity', 0.85);

% Create binary image with detected circles
blackCirclesInImage = zeros(size(grayImage));
for i = 1:length(radii)
    % This creates a matrix representing the coordinates of each pixel in
    % the image and computes if the pixel belongs inside the image
    % If it belongs inside the image, the pixel is marked 1
    [x, y] = meshgrid(1:size(grayImage, 2), 1:size(grayImage, 1));

```

```

        blackCirclesInImage = blackCirclesInImage | ((x - centers(i, 1)).^2 + (y -
centers(i, 2)).^2) <= radii(i)^2;
end

%figure(310)
%performs morphological area filtering and removes all connected components
%that has an area smaller than 4 pixels
circleImage = bwareafilt(blackCirclesInImage,4);
circle_s = regionprops(circleImage,'centroid');
%get the centroids of the 4 circles
centroids = cat(1,circle_s.Centroid);
%imshowpair(image, circle_img, 'blend')
%hold on;
%scatter(centroids(:,1), centroids(:,2), 'r', 'filled');
%hold off;

% Sort the centroids based on their distance from the top left corner
% This is to make sure all centroids are returned in order
% If centroid coordinates are not returned in order, then the geotrans
% and imwarp functions will fail to correct the image.
[~, idx] = sort(sum(centroids .^ 2, 2));
centroids = centroids(idx, :);
end

function correctedImage = correctImage(circleCoordinates, image)
%Function used to project the rotated, translated image back into its
%original position
%load a reference image to use as baseline to get circle coordinates
%In this case, we choose the org_1.png image to act as a standard image
%since this contains only small amount of noise and is not rotated.
referenceImage = loadImage('images/org_1.png');
referenceImageDouble = im2double(referenceImage);

%Circle centroid coordinates of the reference image.
referenceCircleCoordinates = findCircles(referenceImageDouble);

%Circle centroid coordinates of the current image being processed
currentCentres = im2double(circleCoordinates);

% find the transform matrix using these two sparse images
mytform = fitgeotrans(currentCentres,referenceCircleCoordinates,'projective');

%figure(8)
%imshow(image)
%figure(9)
%imshow(referenceImageDouble)
% correct the distorted image using the transform matrix found above
correctedImage = imwarp(image,mytform,'OutputView',imref2d(size(referenceImage)));
%figure(10)
%imshow(correctedImage)
%figure(11)

%Display the input image and corrected image one on top of the other.
imshowpair(image, correctedImage, 'blend')
end

```

```

function meanFilteredImage = meanFilterImage(image, filterSize)
% Function to perform mean filtering on the image. Results in a denoised
% image
% Inputs:
%   image - The double image containing noise
%   filterSize - This indicates the number of pixels used to calculate the
%               mean
% Outputs:
% meanFilteredImage - Denoised, mean filtered image
meanFilter = fspecial('average', filterSize);
meanFilteredImage = imfilter(image, meanFilter);
end

function squareCenters = findSquareCenters()
% This function identifies the squares arranged in 4x4 fashion in a
% reference image and returns the coordinates of the centres of all squares
% in order
% Inputs:
%   None
% Outputs:
%   squareCenters: An array containing the x and y coordinates of all the
%                 squares in the reference image
referenceImage = imread('images/org_1.png');
meanFilteredImage = meanFilterImage(referenceImage, 4);
% Set a threshold mask that considers pixels that have an intensity above
% the threshold value of 35
thresholdMask = rgb2gray(meanFilteredImage)>35;
% figure(1)
% imshow(thresholdMask)

% Removes small, isolated bright regions in the thresholdMask.
thresholdMask = bwareaopen(thresholdMask,100);
% Remove small, isolated dark regions in the thresholdMask
thresholdMask = ~bwareaopen(~thresholdMask,100);

% Remove edge effects
% Removes the outermost connected component in the binary image
thresholdMask = imclearborder(thresholdMask);
% Performs morphological erosion on the thresholdMask using a 10x10 square
% structuring element
thresholdMask = imerode(thresholdMask,ones(10));
%figure(5)
%imshow(thresholdMask)

% Label each square in the image
L = bwlabel(thresholdMask);
% Get the centroids of each labeled square
s = regionprops(L, 'centroid');
centroids = cat(1, s.Centroid);

% Sort the centroids of each square by row and then by column
[~, sorted_indices] = sort(centroids(:, 2)*size(thresholdMask, 1) + centroids(:,
1));
centroids_sorted = centroids(sorted_indices, :);

% Display the sorted centroids
squareCenters = centroids_sorted;
end

```



```

function color_array = detectSquareColours(rgbImage, squareCenters)
% Detects the color of each square arranged in 4x4 fashion
% in an RGB image by computing the mean L*a*b* values of a square region
% around each center coordinate.
% Inputs:
%   - rgbImage: a double RGB image matrix
%   - squareCenters: a 16x2 matrix containing the x and y coordinates of
%     the center of each square, in the order left to right, top to bottom
% Outputs:
%   - color_array: a 4x4 cell array containing the name of the color of each
%     square

% Convert the RGB image to LAB color space for better color detection
labSpaceImage = rgb2lab(rgbImage);
%lab_image = imfilter(lab_image, meanFilter);
%lab_image = imadjust(lab_image, [0.1, 0.9],[]);

%figure(80)
%imshow(labSpaceImage)

% Define the max and min thresholds for the L,A,B channels for each color
% These are identified through trial and error
red_min_thresh = [45      62      35];
red_max_thresh = [58      83      67];
green_min_thresh = [74     -89      65];
green_max_thresh = [90     -70      86];
yellow_min_thresh = [80    -34  75];
yellow_max_thresh = [95    -14  96.54];
blue_min_thresh = [28      60     -105];
blue_max_thresh = [46      85     -70];
white_min_thresh = [85      -5      -5];
white_max_thresh = [102     5       5];

% Initialize the output color array
color_array = cell(4, 4);

% Define the size of the square region to use for color detection
square_size = 50;

% Loop over each square center and detect its color
for i = 1:size(squareCenters, 1)
    % Extract the square region around the center
    x_center = round(squareCenters(i, 1));
    y_center = round(squareCenters(i, 2));
    square_region = labSpaceImage(max(1,y_center-
square_size/2):min(size(rgbImage,1),y_center+square_size/2), ...
                                max(1,x_center-
square_size/2):min(size(rgbImage,2),x_center+square_size/2), :);

    % Compute the mean L*a*b* values of the square region
    lab_values = mean(mean(square_region, 1), 2);
    lab_values = reshape(lab_values, 1, 3);
    %squareCenters(i,1)
    %squareCenters(i,2)
    %lab_values

    % Determine the color of the square based on the color thresholds

```

```

        if all(all(lab_values >= red_min_thresh) & all(lab_values <= red_max_thresh))
            color_array{i} = 'red';
        elseif all(all(lab_values >= green_min_thresh) & all(lab_values <=
green_max_thresh))
            color_array{i} = 'green';
        elseif all(all(lab_values >= yellow_min_thresh) & all(lab_values <=
yellow_max_thresh))
            color_array{i} = 'yellow';
        elseif all(all(lab_values >= blue_min_thresh) & all(lab_values <=
blue_max_thresh))
            color_array{i} = 'blue';
        elseif all(all(lab_values >= white_min_thresh) & all(lab_values <=
white_max_thresh))
            color_array{i} = 'white';
        else
            color_array{i} = 'unknown';
        end
    end

% Reshape the color array into a 4x4 matrix
color_array = transpose(reshape(color_array, 4, 4));
end

```