



**Image Processing
Lab Project Report**

Submitted by

Sanjana Neeli Nagaraj
(277227)

MSc in Data Science

Under the guidance of

Dr. Phil Birch

School of Mathematical and Physical Sciences

LAB PROJECT

Aim:

The aim of this project is to process an input image by performing geometric transformations to correct distortion and identify colors within specific regions of the image.

Functions Used:

1. *'mainFunction'*:

- Description:
 - ✓ The 'mainFunction' serves as the entry point to the image processing pipeline. It loads an image, identifies circles, corrects image distortion, and identifies colors within specific regions.
- Design Decisions:
 - ✓ Error handling for cases where fewer than 4 circles are detected.
 - ✓ Sequential execution of sub-functions for clarity and modularity.
- Performance Assessment:
 - ✓ The function performs adequately for most images but may encounter issues with highly complex or noisy images.
 - ✓ Execution time may vary based on the size and complexity of the image.
- Suggestions for Improvement:
 - ✓ Implement more robust methods for circle detection to handle challenging cases.
 - ✓ Optimize color identification algorithm for improved accuracy and efficiency.

2. *'findColors'*:

- Description:
 - ✓ The 'findColors' function identifies colors within specific regions of the input image by converting it to LAB color space and comparing color points with a predefined color scale.
- Design Decisions:
 - ✓ Utilization of LAB color space for better color representation.
 - ✓ Defining color points from specific regions of the image and calculating their LAB values.
 - ✓ Using a predefined color scale to map LAB values to color names.
 - ✓ Reshaping color names into a 4x4 matrix representing color regions.
- Performance Assessment:
 - ✓ The function accurately identifies colors within specified regions.
 - ✓ Computational complexity may increase with larger images due to the iteration over color points.

- ✓ However, it may not handle variations in lighting conditions or color representation issues effectively.
- Suggestions for Improvement:
 - ✓ Implement parallel processing techniques to improve performance, especially for large images.
 - ✓ Incorporate adaptive color thresholding or clustering techniques to improve robustness to lighting variations.
 - ✓ Consider alternative color representations or color space transformations for better color discrimination.

3. *'loadImage'*:

- Description:
 - ✓ The *'loadImage'* function reads an image from the file specified by the input filename, converts it to double precision, and displays it.
- Design Decisions:
 - ✓ Using *'imread'* to read the image file.
 - ✓ Converting the image to double precision using *'im2double'*.
 - ✓ Displaying the loaded image using *'imshow'*.
- Performance Assessment:
 - ✓ This function efficiently loads and displays the image.
 - ✓ No significant performance issues observed.
 - ✓ However, memory usage may be a concern for very large images due to the conversion to double precision.
- Improvements:
 - ✓ Error handling could be enhanced to deal with invalid file formats or file not found situations.
 - ✓ Implement memory-efficient techniques for loading and handling large images.
 - ✓ Add support for additional image formats if required.

4. *'findCircles'*:

- Description:
 - ✓ The *'findCircles'* function identifies circular regions in the image using a thresholding and blob detection approach. It detects circular objects using connected component analysis and identifies their coordinates.
- Design Decisions:
 - ✓ Utilization of thresholding and blob detection for circle identification.
 - ✓ Conversion of the image to grayscale and binary thresholding for isolating circular objects.
 - ✓ Sorting and selecting the largest circular blobs.

- ✓ Sorting blob coordinates in clockwise order starting from the bottom-left.
- Performance Assessment:
 - ✓ The function generally effectively detects circular regions in the image.
 - ✓ However, it may encounter issues with highly irregular shapes that resemble circles.
 - ✓ Performance may vary depending on the complexity of the scene and lighting conditions.
- Suggestions for Improvement:
 - ✓ Experiment with alternative circle detection algorithms, such as the Hough Transform, for improved robustness.
 - ✓ Implement adaptive thresholding techniques to handle varying lighting conditions more effectively.
 - ✓ Enhance the algorithm's ability to handle complex scenes with overlapping or irregularly shaped objects.

5. '*correctImage*':

- Description:
 - ✓ The '*correctImage*' function corrects image distortion based on detected circle coordinates by applying a projective transformation. It enhances image contrast and reduces glare to improve overall image quality.
- Design Decisions:
 - ✓ Definition of a fixed reference box to map the distorted image to.
 - ✓ Calculation of the transformation matrix using '*fitgeotrans*'.
 - ✓ Application of the transformation to the input image and cropping the result to the fixed box size.
 - ✓ Utilization of flat-field correction and contrast adjustment techniques to enhance image quality.
- Performance Assessment:
 - ✓ The function effectively corrects image distortion and improves image quality by reducing glare and enhancing contrast.
 - ✓ However, it may encounter challenges with severe distortions or complex scenes.
- Suggestions for Improvement:
 - ✓ Explore alternative image correction techniques for better handling of complex distortions.
 - ✓ Fine-tune parameters for flat-field correction and contrast adjustment to adapt to various distortion levels.
 - ✓ Investigate advanced methods or algorithms for distortion correction to improve performance in challenging scenarios.

Conclusion:

Overall, the image processing functions presented in the code provide a solid foundation for basic image analysis tasks. They effectively correct image distortion and identify colors within specific regions. While these functions perform adequately for many cases, there is room for improvement in terms of accuracy, efficiency, and robustness.

Suggestions for Improvements:

- Implementing advanced algorithms or techniques for circle detection and image distortion correction could improve accuracy and robustness.
- Enhancing color identification methods to handle diverse lighting and color conditions would make the functions more versatile.
- Integrating error handling and feedback mechanisms to improve usability and reliability, especially when dealing with unexpected inputs or scenarios.
- Conducting comprehensive testing and validation across a wide range of image datasets to assess performance and identify areas for improvement.

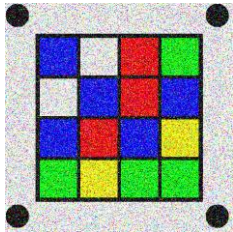
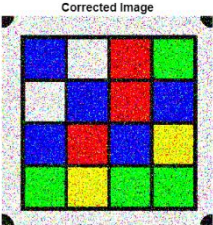
By incorporating these suggestions and refining the functionality of the image processing functions, their performance and usability can be significantly enhanced, making them more suitable for a variety of applications.

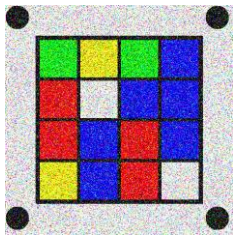
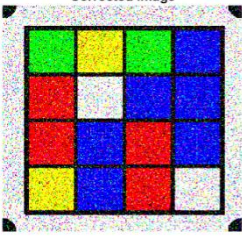
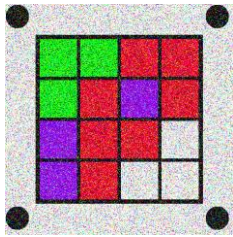
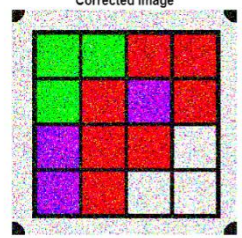
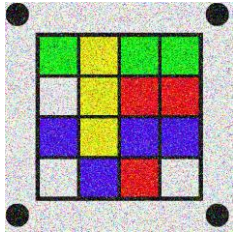
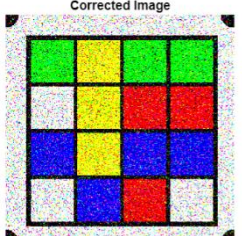
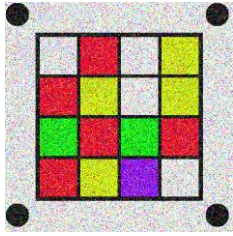
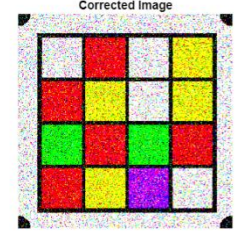
Results:

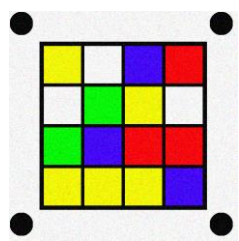
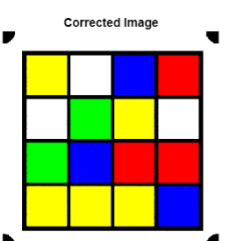
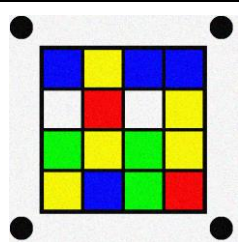
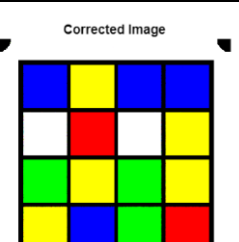
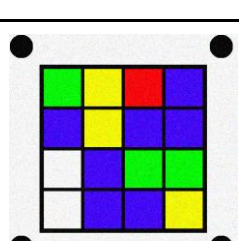
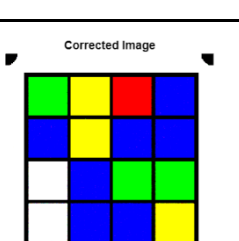
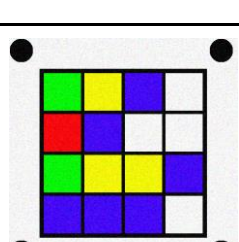
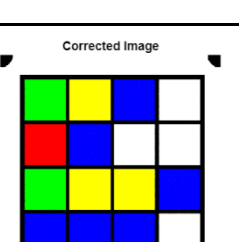
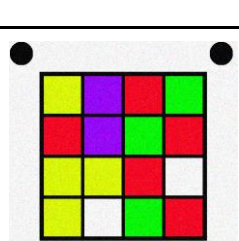
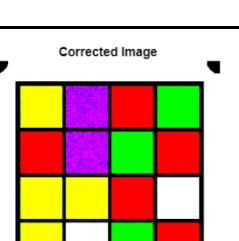
Passed

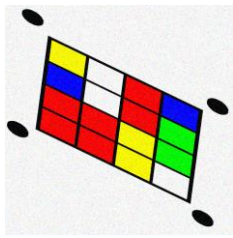
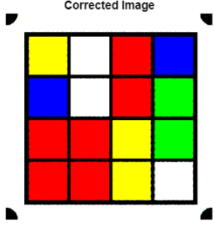
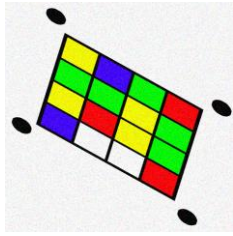
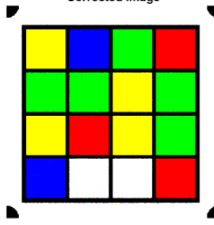
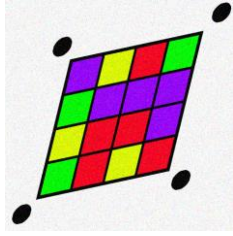
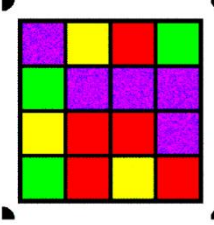
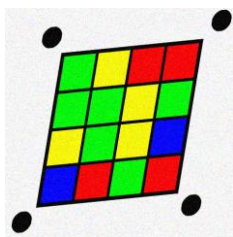
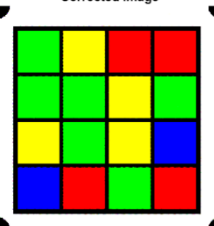
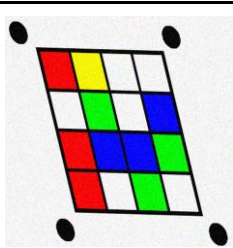
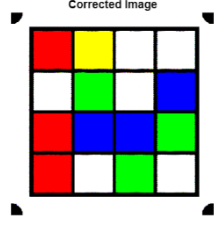
Score is: 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00
100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00

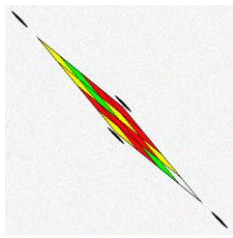
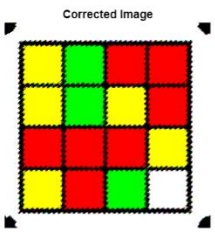
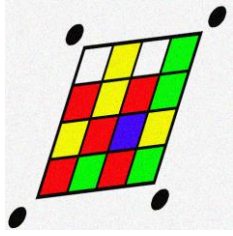
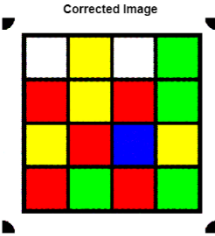
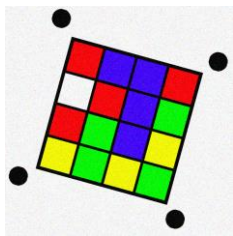
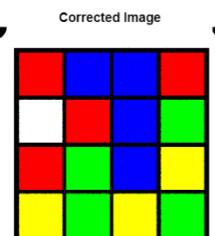
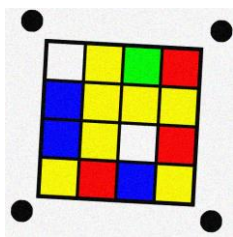
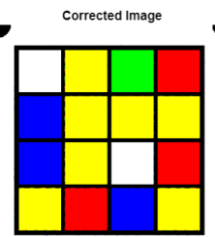
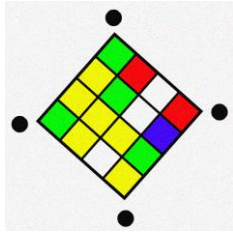
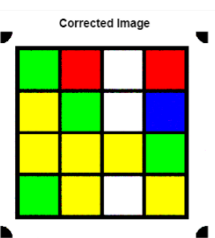
Mean score 100.000000

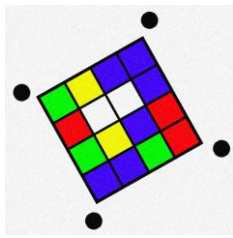
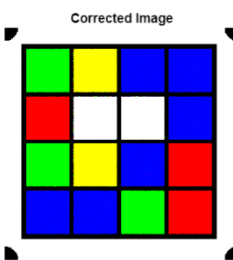
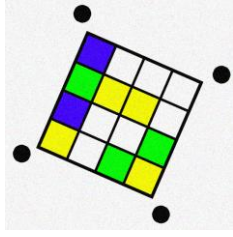
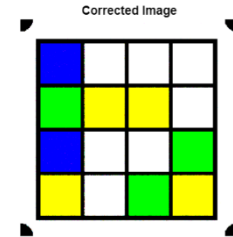
File Name	Input Image	Output Image	Colour Output	Success	Notes
noise_1.png			$\begin{bmatrix} b & w & r & g \\ w & b & r & b \\ b & r & b & y \\ g & y & g & g \end{bmatrix}$	Yes	Partial removal of noise. Detects blue, white, red, green and yellow color.

noise_2.png			$\begin{bmatrix} g & y & g & b \\ r & w & b & b \\ r & b & r & b \\ y & b & r & w \end{bmatrix}$	Yes	Partial removal of noise. Detects blue, white, red, green and yellow color.
noise_3.png			$\begin{bmatrix} g & g & r & r \\ g & r & b & r \\ b & r & r & w \\ b & r & w & w \end{bmatrix}$	Yes	Partial removal of noise. Detects white, red, green, and yellow colour. But it's detecting purple as blue.
noise_4.png			$\begin{bmatrix} g & y & g & g \\ w & y & r & r \\ b & y & b & b \\ w & b & r & w \end{bmatrix}$	Yes	Partial removal of noise. Detects blue, white, red, green and yellow color.
noise_5.png			$\begin{bmatrix} w & r & w & y \\ r & y & w & y \\ g & r & g & r \\ r & y & b & w \end{bmatrix}$	Yes	Partial removal of noise. Detects white, red, green, and yellow colour. But it's detecting purple as blue.

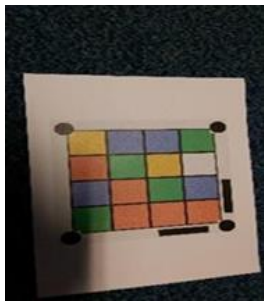

org_1.png			$\begin{bmatrix} y & w & b & r \\ w & g & y & w \\ g & b & r & r \\ y & y & y & b \end{bmatrix}$	Yes	Detects all the co-ordinates and gives much clearer corrected image.
org_2.png			$\begin{bmatrix} b & y & b & b \\ w & r & w & y \\ g & y & g & y \\ y & b & g & r \end{bmatrix}$	Yes	Detects all the co-ordinates and gives much clearer corrected image.
org_3.png			$\begin{bmatrix} g & y & r & b \\ b & y & b & b \\ w & b & g & g \\ w & b & b & y \end{bmatrix}$	Yes	Detects all the co-ordinates and gives much clearer corrected image.
org_4.png			$\begin{bmatrix} g & y & b & w \\ r & b & w & w \\ g & y & y & b \\ b & b & b & w \end{bmatrix}$	Yes	Detects all the co-ordinates and gives much clearer corrected image.
org_5.png			$\begin{bmatrix} y & b & r & g \\ r & b & g & r \\ y & y & r & w \\ y & w & g & r \end{bmatrix}$	Yes	Detects all the co-ordinates and gives much clearer corrected image.

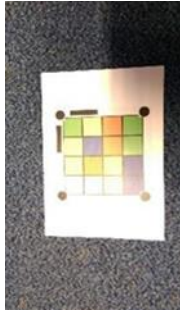
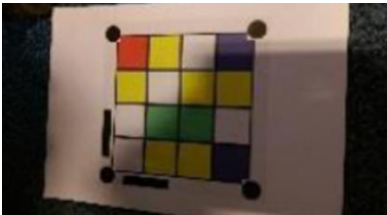
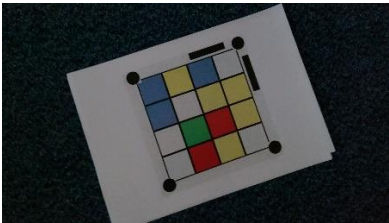
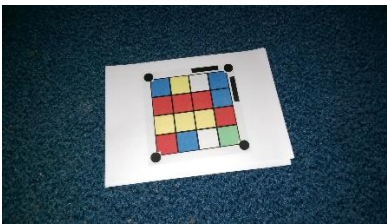

proj_1.png		Corrected Image 	$\begin{bmatrix} b & r & w & y \\ g & r & w & b \\ g & y & r & r \\ w & y & r & r \end{bmatrix}$	Yes	Projected image is now tilted and displayed as straight image.
proj_2.png		Corrected Image 	$\begin{bmatrix} y & b & g & r \\ g & g & y & g \\ y & r & y & g \\ b & w & w & r \end{bmatrix}$	Yes	Projected image is now tilted and displayed as straight image.
proj_3.png		Corrected Image 	$\begin{bmatrix} b & y & r & g \\ g & b & b & b \\ y & r & r & b \\ g & r & y & r \end{bmatrix}$	Yes	Projected image is now tilted and displayed as straight image.
proj_4.png		Corrected Image 	$\begin{bmatrix} g & y & r & r \\ g & g & y & g \\ y & g & y & b \\ b & r & g & r \end{bmatrix}$	Yes	Projected image is now tilted and displayed as straight image.
proj_5.png		Corrected Image 	$\begin{bmatrix} r & y & w & w \\ w & g & w & b \\ r & b & b & g \\ r & w & g & w \end{bmatrix}$	Yes	Projected image is now tilted and displayed as straight image.

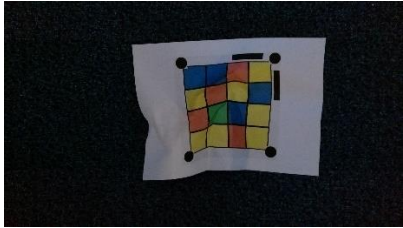
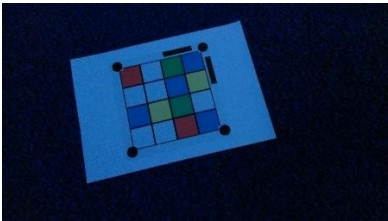

proj_6.png		Corrected Image 	$\begin{bmatrix} y & g & r & r \\ y & g & y & r \\ r & r & r & y \\ y & r & g & w \end{bmatrix}$	Yes	Projected image is now tilted and displayed as straight image.
proj_7.png		Corrected Image 	$\begin{bmatrix} w & y & w & g \\ r & y & r & g \\ y & r & b & y \\ r & g & r & g \end{bmatrix}$	Yes	Projected image is now tilted and displayed as straight image.
rot_1.png		Corrected Image 	$\begin{bmatrix} r & b & b & r \\ w & r & b & g \\ r & g & b & y \\ y & g & y & g \end{bmatrix}$	Yes	Image gets flipped after detecting the circle co-ordinates.
rot_2.png		Corrected Image 	$\begin{bmatrix} w & y & g & r \\ b & y & y & y \\ b & y & w & r \\ y & r & b & y \end{bmatrix}$	Yes	Image gets flipped after detecting the circle co-ordinates.
rot_3.png		Corrected Image 	$\begin{bmatrix} g & r & w & r \\ y & g & w & b \\ y & y & y & g \\ g & y & w & y \end{bmatrix}$	Yes	Image gets flipped after detecting the circle co-ordinates.

rot_4.png			$\begin{bmatrix} g & y & b & b \\ r & w & w & b \\ g & y & b & r \\ b & b & g & r \end{bmatrix}$	Yes	Image gets flipped after detecting the circle co-ordinates.
rot_5.png			$\begin{bmatrix} b & w & w & w \\ g & y & y & w \\ b & w & w & g \\ y & w & g & y \end{bmatrix}$	Yes	Image gets flipped after detecting the circle co-ordinates.

Views on given Real Images:

File Name	Images	Comments
IMAG0032.jpg		It's a geometric arrangement of coloured squares with black circles on each corner of the grid.
IMAG0033.jpg		It's the same image as above but clicked from a different angle, with comparatively bright lighting.

IMAG0034.jpg		It's a glare image making difficult to identify the colours in the grid.
IMAG0035.jpg		Shadowed image, bright on the first half and dull on the other.
IMAG0036.jpg		Comparatively much clearer image clicked.
IMAG0037.jpg		Image captured in a bright light, evident colours for detection.
IMAG0038.jpg		There's not just one, but many text and images to process. It's been zoomed out losing focus.

IMAG0041.jpg		Some congestion is noticed and can be rectified using desired/inbuilt functions.
IMAG0042.jpg		Blue filter has to be removed before detecting the colours.
IMAG0044.jpg		Blurry image makes the detection challenging as it might be tricky to detect the co-ordinated and identify the colours present.

Implemented Code:

```

clc
clear all

% Upload the image
[filename, filepath] = uigetfile('*.png', 'Select the required image file');
if isequal(filename, 0)
    disp('No file selected');
else
    mainFunction(fullfile(filepath, filename));
end

function mainFunction(filename)
    % Load and preprocess image
    img = loadImage(filename);
    figure();imshow(img)

    % Find circle locations
    circles = findCircles(img);

    % Check if at least 4 circles are detected
    if size(circles, 1) < 4

```

```

        disp('Less than 4 circles are detected. Therefore, cannot perform
geometric transformation.');
```

return;

end

% Correct image distortion based on circles

```
[adjusted_img, circles] = correctImage(circles, img);
```

% Display the corrected image

```
imshow(adjusted_img);
title('Corrected Image');
```

% Identify colors within specific regions

```
color_matrix = findColors(adjusted_img);
disp("COLOUR-MATRIX OF IMAGE:");
disp(color_matrix); % Display color matrix
```

end

%% Find the colors in the image

```
function color_matrix = findColors(img)
    lab_img = rgb2lab(img); % Converting RGB colorspace to LAB
    f = fspecial('average', 11); % Using mean-filter (f) to remove noise
    lab_img = imfilter(lab_img, f);

    % Coordinates (c) from where we find the colors
    c = [80 172 259 369];

    % Finding all 16 points
    color_points = zeros(16, 3);
    count = 0;
    for i = 1:4
        for j = 1:4
            count = count + 1;
            x = c(1, i);
            y = c(1, j);
            temp = lab_img(x:x + 56, y:y + 56, :);
            color_points(count, : ) = mean(reshape(temp, [], 3), 1);
        end
    end

    % Defining the colors in RGB and LAB
    rgb_scale = [1 0 0; 0 1 0; 0 0 1; 1 1 0; 1 1 1; 0.2824, 0.2392, 0.5451]; %
Red, Green, Blue, Yellow, White, Violet
    color_names = {'r', 'g', 'b', 'y', 'w', 'p'}; % r=red, g=green, b=blue,
y=yellow, w=white, p=purple
```

```

lab_scale = rgb2lab(rgb_scale); % converting the colors to lab

% Calculating distances between color points and color scale
d = pdist2(color_points, lab_scale, 'euclidean');

% Assigning colors based on minimum distances
[~, idx] = min(d, [], 2);
patchnames = color_names(idx);

% Reshaping the color matrix
color_matrix = reshape(patchnames, 4, 4)';
end

% Loads an image from the file specified by filename, and returns it as type
double.
function image = loadImage(filename)
    % Read in the image using imread
    img = imread(filename);
    % Convert the image to double precision
    image = im2double(img);
    figure();imshow(image)
    title('Input Image');
end

% Find the coordinates of the black circle
function circleCoordinates = findCircles(image)
    img = image;
    gray_img = rgb2gray(img); % Changing the given image to grey image
    threshold = graythresh(gray_img);
    binary_img = imbinarize(gray_img, threshold);
    inverted_binary_img = imcomplement(binary_img); % inverting the binary
image
    cc = bwconncomp(inverted_binary_img);
    areas = cellfun(@numel, cc.PixelIdxList);
    [sorted_areas, sorted_indices] = sort(areas, 'descend'); % Sorting in
descending order

    % Getting the coordinates of the first four largest black blobs
    num_blobs = 5;
    blob_coords = zeros(num_blobs, 2);
    for i = 2:num_blobs
        blob_indices = cc.PixelIdxList{sorted_indices(i)};
        [rows, cols] = ind2sub(size(inverted_binary_img), blob_indices);
        blob_coords(i, :) = [mean(cols), mean(rows)];
    end
end

```

```

end

% Removing the first coordinate from the blob_coords matrix
blob_coords(1, :) = [];
% Sort the coordinates in clockwise order starting from bottom-left
sortedCoordinates = sortrows(blob_coords);

if sortedCoordinates(2, 2) < sortedCoordinates(1, 2)
    % If the second coordinate is below the first, swap them
    sortedCoordinates([1 2], :) = sortedCoordinates([2 1], :);
end
if sortedCoordinates(4, 2) > sortedCoordinates(3, 2)
    % If the fourth coordinate is above the third, swap them
    sortedCoordinates([3 4], :) = sortedCoordinates([4 3], :);
end
circleCoordinates = sortedCoordinates;
end

% Correct the distorted images
function [outputImage, correctedCoordinates] = correctImage(coordinates,
image)
    boxf = [[0, 0]; [0, 480]; [480, 480]; [480, 0]]; % Define a fixed box with
coordinates
    disp(coordinates)

    % Calculating the transformation matrix from the given Coordinates to
transform the matrix to the fixed box using projective transformation
    TF = fitgeotrans(coordinates, boxf, 'projective');
    % Create an image reference object with the size of the input image
    outview = imref2d(size(image));
    % Apply the calculated transformation matrix to the input image and create
a new image with fill value 255 (white) outside the boundaries of the input
image
    B = imwarp(image, TF, 'fillvalues', 255, 'OutputView', outview);
    % Crop the image to a size of 480x480
    B = imcrop(B, [0, 0, 480, 480]);
    % Try to suppress the glare in the image using flat-field correction
    B = imflatfield(B, 40);
    % Adjust the levels of the image to improve contrast
    B = imadjust(B, [0.4, 0.65]);
    % Assign the corrected image to the outputImage variable
    outputImage = B;
    correctedCoordinates = boxf;
end

```

REFERENCES:

- [1] <https://uk.mathworks.com/help/images/noise-removal.html>
- [2] <https://uk.mathworks.com/help/images/ref/imwarp.html>
- [3] <https://uk.mathworks.com/matlabcentral/answers/456973-how-to-make-an-image-straightener-like-that-of-cam-scanner-app>
- [4] <https://uk.mathworks.com/matlabcentral/answers/547560-detecting-colors-on-a-rgb-image>
- [5] <https://uk.mathworks.com/matlabcentral/answers/1664934-returning-an-array-of-colors-from-a-double-image>