School of Engineering and Informatics

Image Processing Assignment 1

Lab Project

Candidate No:
276236

Module Convenor:
Dr. Phil Birch

Module:
Image Processing

Lab Project

Introduction:

This project presents an image processing pipeline in MATLAB for correcting distortion and identifying colors in specific regions of an image. The pipeline combines various functions to detect and correct distorted images containing colored patches and four black circles using techniques like color space conversion, thresholding, contour detection, and geometric transformations. The code also identifies and extracts colors from the corrected image, assigning color labels to each region to create a 4x4 color grid. The implementation involves design decisions and trade-offs for accuracy, efficiency, and robustness, utilizing MATLAB's Image Processing Toolbox. The code is modular and readable, with separate functions for each task, allowing for easier maintenance and future enhancements. The pipeline's performance has been assessed based on speed, accuracy, and robustness to different image conditions, and while satisfactory for most scenarios, improvements can be made in handling complex backgrounds, non-uniform illumination, and advanced color perception models.

Here is a description of each function:

1. *findColours* function:

- How it works:
  - Converts the input image from the RGB color space to the LAB color space.
  - Defines specific color points (coordinates) from where the colors are extracted.
  - Extracts color values from the specified color points.
  - Calculates the distances between the extracted color points and a predefined color scale.
  - Assigns colors to each color point based on the minimum distance.
  - Reshapes the color matrix into a 4x4 grid.
- Design decisions:
  - Uses the LAB color space for better perceptual uniformity compared to RGB.
  - Extracts colors from specific color points instead of the entire image.
  - Defines a color scale with RGB and LAB values for red, green, blue, yellow, white, and purple.
- Performance assessment:
  - The function is efficient and fast, as it only processes specific color points instead of the entire image.

- ➢ The use of the LAB color space improves color accuracy compared to using RGB.

❖ Suggestions for improvement:

- ➢ Consider using more color points to capture a wider range of colors in the image.
- ➢ Experiment with different color spaces, such as CIELAB or CIELUV, for enhanced color perception.

2. *loadImage* function:

❖ How it works:

- ➢ Uses the imread function to read the image file.
- ➢ Converts the image to double precision using the im2double function.

❖ Design decisions:

- ➢ Utilizes standard MATLAB functions for reading and converting image files.

❖ Performance assessment:

- ➢ The function is fast and efficient, as it relies on built-in MATLAB functions.

❖ Suggestions for improvement:

- ➢ Add error checking to handle invalid filenames or unreadable image files.

3. *findCircles* function:

❖ How it works:

- ➢ Converts the input image to grayscale.
- ➢ Applies a binary threshold to the grayscale image using Otsu's method.
- ➢ Finds the contours of the resulting binary image.
- ➢ Calculates the moments of each contour to find the centroids (centers) of the circles.
- ➢ Sorts the circle centers in clockwise order starting from the bottom-left corner.

❖ Design decisions:

- ➢ Uses a binary threshold for segmenting the image into foreground (circles) and background regions.
- ➢ Employs the regionprops function to calculate contour moments for finding circle centers.

- Performance assessment:
  - The function is relatively fast and accurate for detecting circles in most cases.
  - It may struggle with circles having non-uniform brightness or complex backgrounds.
- Suggestions for improvement:
  - Consider using more advanced circle detection algorithms, such as the Circular Hough Transform, to handle non-uniform brightness and complex backgrounds.

4. *correctImage* function:

- How it works:
  - Takes the coordinates of the four black circles as input.
  - Calculates the transformation matrix using projective transformation to map the input coordinates to a target rectangular shape.
  - Applies the transformation matrix to the input image to correct distortion.
  - Crops the corrected image to a specified size (480x480 pixels).
  - Performs additional image enhancement techniques, such as flat-field correction, contrast adjustment, and denoising, to improve image quality.
- Design decisions:
  - Uses projective transformation to handle perspective distortion in the image.
  - Applies image enhancement techniques to improve the visual quality and clarity of the corrected image.
- Performance assessment:
  - The function effectively corrects image distortion and produces a visually enhanced output image.
  - The use of projective transformation ensures accurate mapping of the input coordinates to the target shape.
- Suggestions for improvement:
  - Implement error checking to validate the input coordinates' size and format.
  - Clearly document the assumption about the clockwise order of coordinates or add error checking to handle different coordinate orders.

Overall design:

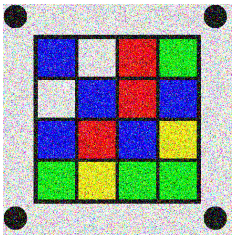
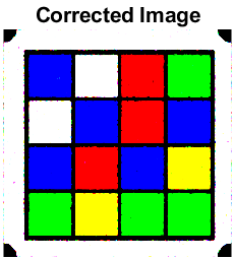
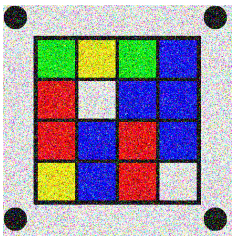
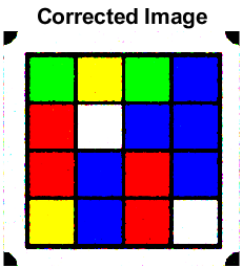
- How it works:
  - The program follows a modular design, with separate functions for each major step of the image processing pipeline.
  - The main script orchestrates the execution of these functions in a sequential manner.

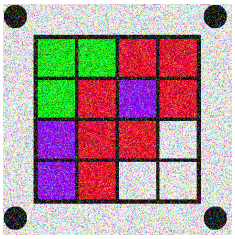
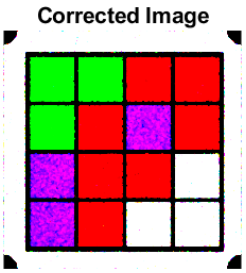
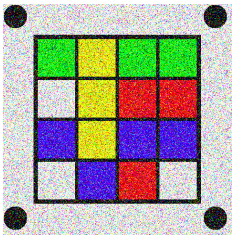
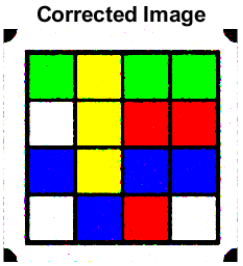
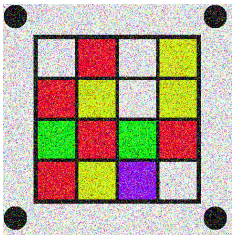
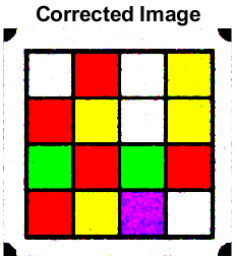
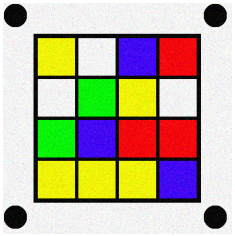
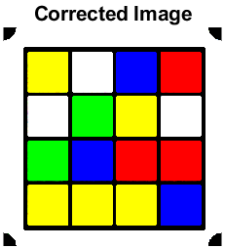
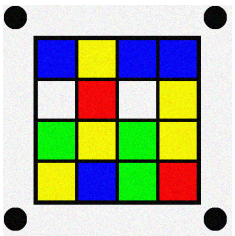
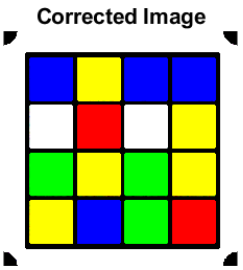
- Design decisions:
  - The modular design promotes code reusability, maintainability, and readability.
  - The use of separate functions allows for easier debugging and future enhancements.
- Performance assessment:
  - The overall pipeline effectively corrects image distortion, identifies colors, and produces a color grid representation.
  - The modular design contributes to efficient execution and resource utilization.
- Potential improvements:
  - Further modularize the program by creating more granular functions for specific tasks.
  - Implement a user interface with interactive elements for real-time parameter adjustments.
  - Optimize performance by vectorizing loops and employing efficient data structures.
  - Conduct thorough testing and validation to assess the robustness and accuracy of the pipeline under various image conditions.

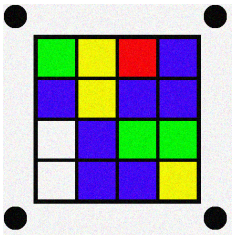
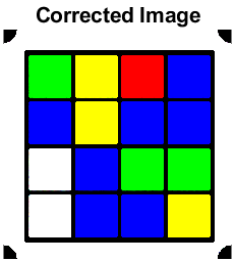
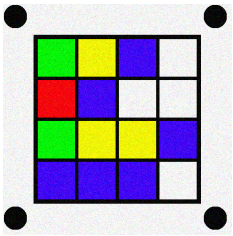
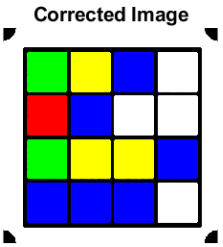
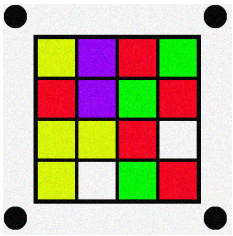
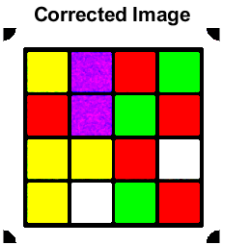
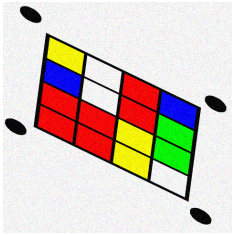
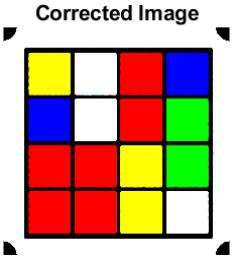
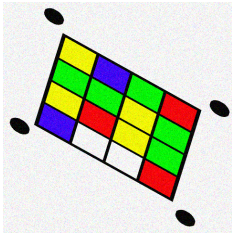
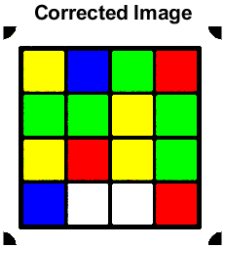
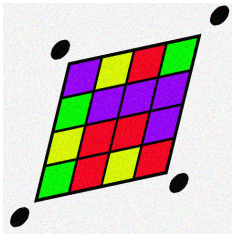
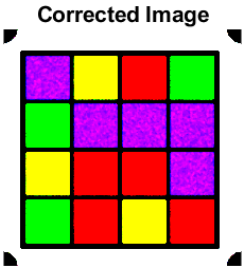
Results & Output :

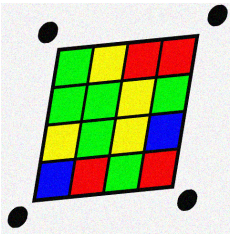
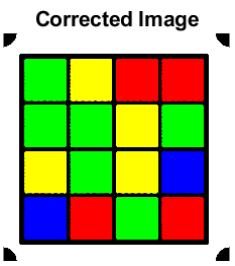
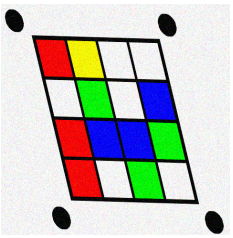
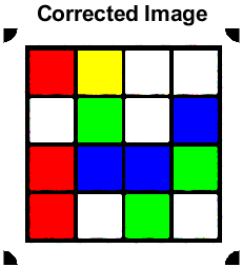
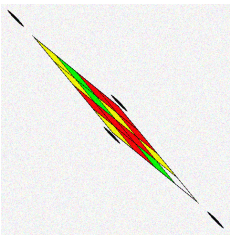
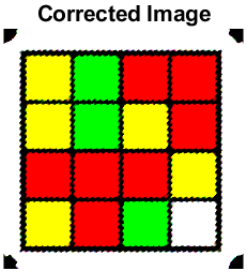
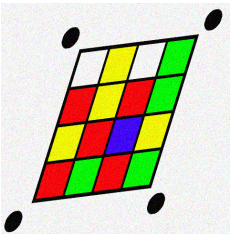
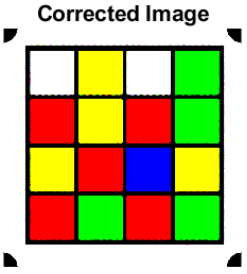
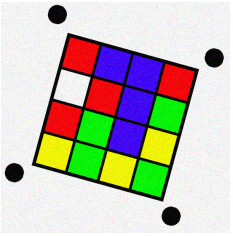
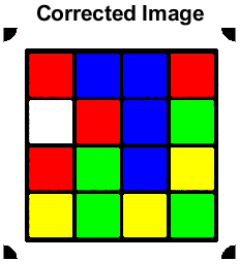
Score is: *100.00 100.00*

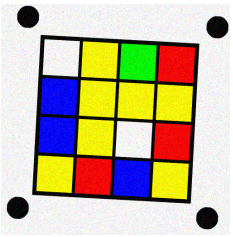
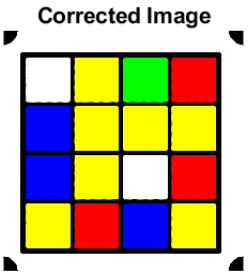
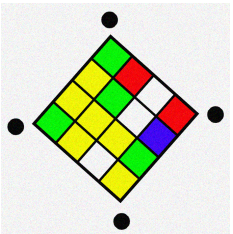
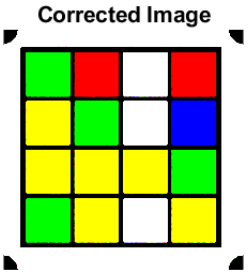
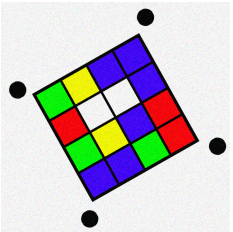
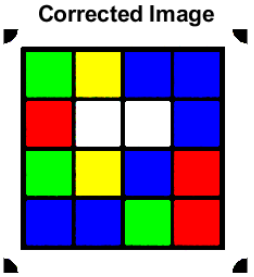
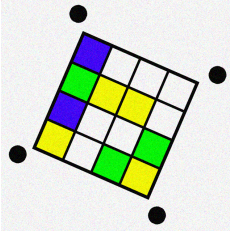
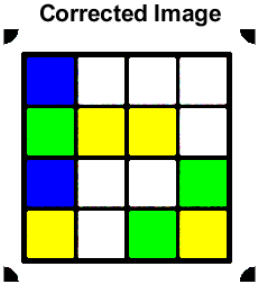
Mean score *100.000000*

Filename	Image	Output Image	Output	Success	Note
noise_1.png			Color Grid of the Image: {'b'} {'w'} {'r'} {'g'} {'w'} {'b'} {'r'} {'b'} {'b'} {'r'} {'b'} {'y'} {'g'} {'y'} {'g'} {'g'}	Yes	Denoised the image and successfully detected all the colors
noise_2.png			Color Grid of the Image: {'g'} {'y'} {'g'} {'b'} {'r'} {'w'} {'b'} {'b'} {'r'} {'b'} {'r'} {'b'} {'y'} {'b'} {'r'} {'w'}	Yes	Denoised the image and successfully detected all the colors

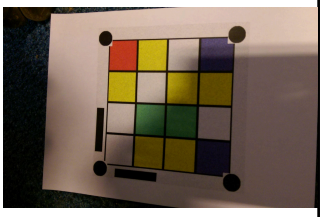
noise_3.png			Color Grid of the Image: {'g'} {'g'} {'r'} {'r'} {'g'} {'r'} {'b'} {'r'} {'b'} {'r'} {'r'} {'w'} {'b'} {'r'} {'w'} {'w'}	Yes	Denoised the image and successfully detected all the colors except purple as blue as required in results
noise_4.png			Color Grid of the Image: {'g'} {'y'} {'g'} {'g'} {'w'} {'y'} {'r'} {'r'} {'b'} {'y'} {'b'} {'b'} {'w'} {'b'} {'r'} {'w'}	Yes	Denoised the image and successfully detected all the colors
noise_5.png			Color Grid of the Image: {'w'} {'r'} {'w'} {'y'} {'r'} {'y'} {'w'} {'y'} {'g'} {'r'} {'g'} {'r'} {'r'} {'y'} {'b'} {'w'}	Yes	Denoised the image and successfully detected all the colors except purple as blue as required in results
org_1.png			Color Grid of the Image: {'y'} {'w'} {'b'} {'r'} {'w'} {'g'} {'y'} {'w'} {'g'} {'b'} {'r'} {'r'} {'y'} {'y'} {'y'} {'b'}	Yes	Successfully detected all the colors
org_2.png			Color Grid of the Image: {'b'} {'y'} {'b'} {'b'} {'w'} {'r'} {'w'} {'y'} {'g'} {'y'} {'g'} {'y'} {'y'} {'b'} {'g'} {'r'}	Yes	Successfully detected all the colors

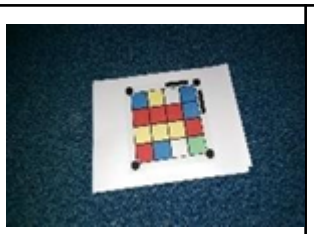
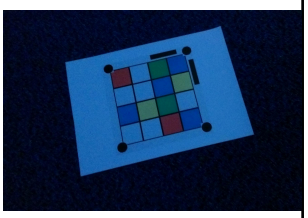
org_3.png			Color Grid of the Image: {'g'} {'y'} {'r'} {'b'} {'b'} {'y'} {'b'} {'b'} {'w'} {'b'} {'g'} {'g'} {'w'} {'b'} {'b'} {'y'}	Yes	Successfully detected all the colors
org_4.png			Color Grid of the Image: {'g'} {'y'} {'b'} {'w'} {'r'} {'b'} {'w'} {'w'} {'g'} {'y'} {'y'} {'b'} {'b'} {'b'} {'b'} {'w'}	Yes	Successfully detected all the colors
org_5.png			Color Grid of the Image: {'y'} {'b'} {'r'} {'g'} {'r'} {'b'} {'g'} {'r'} {'y'} {'y'} {'r'} {'w'} {'y'} {'w'} {'g'} {'r'}	Yes	Successfully detected all the colors except purple as blue as required in results
proj_1.png			Color Grid of the Image: {'y'} {'w'} {'r'} {'b'} {'b'} {'w'} {'r'} {'g'} {'r'} {'r'} {'y'} {'g'} {'r'} {'r'} {'y'} {'w'}	Yes	Successfully corrected the projected image and detected colors
proj_2.png			Color Grid of the Image: {'y'} {'b'} {'g'} {'r'} {'g'} {'g'} {'y'} {'g'} {'y'} {'r'} {'y'} {'g'} {'b'} {'w'} {'w'} {'r'}	Yes	Successfully corrected the projected image and detected colors
proj_3.png			Color Grid of the Image: {'b'} {'y'} {'r'} {'g'} {'g'} {'b'} {'b'} {'b'} {'y'} {'r'} {'r'} {'b'} {'g'} {'r'} {'y'} {'r'}	Yes	Successfully corrected the projected image and detected colors except

					purple as blue as required in results
proj_4.png			Color Grid of the Image: {'g'} {'y'} {'r'} {'r'} {'g'} {'g'} {'y'} {'g'} {'y'} {'g'} {'y'} {'b'} {'b'} {'r'} {'g'} {'r'}	Yes	Successfully corrected the projected image and detected colors
proj_5.png			Color Grid of the Image: {'r'} {'y'} {'w'} {'w'} {'w'} {'g'} {'w'} {'b'} {'r'} {'b'} {'b'} {'g'} {'r'} {'w'} {'g'} {'w'}	Yes	Successfully corrected the projected image and detected colors
proj_6.png			Color Grid of the Image: {'y'} {'g'} {'r'} {'r'} {'y'} {'g'} {'y'} {'r'} {'r'} {'r'} {'r'} {'y'} {'y'} {'r'} {'g'} {'w'}	Yes	Successfully corrected the projected image and detected colors
proj_7.png			Color Grid of the Image: {'w'} {'y'} {'w'} {'g'} {'r'} {'y'} {'r'} {'g'} {'y'} {'r'} {'b'} {'y'} {'r'} {'g'} {'r'} {'g'}	Yes	Successfully corrected the projected image and detected colors
rot_1.png			Color Grid of the Image: {'r'} {'b'} {'b'} {'r'} {'w'} {'r'} {'b'} {'g'} {'r'} {'g'} {'b'} {'y'} {'y'} {'g'} {'y'} {'g'}	Yes	Successfully rotated and detected all colors

rot_2.png			Color Grid of the Image: {'w'} {'y'} {'g'} {'r'} {'b'} {'y'} {'y'} {'y'} {'b'} {'y'} {'w'} {'r'} {'y'} {'r'} {'b'} {'y'}	Yes	Successfully rotated and detected all colors
rot_3.png			Color Grid of the Image: {'g'} {'r'} {'w'} {'r'} {'y'} {'g'} {'w'} {'b'} {'y'} {'y'} {'y'} {'g'} {'g'} {'y'} {'w'} {'y'}	Yes	Successfully rotated and detected all colors
rot_4.png			Color Grid of the Image: {'g'} {'y'} {'b'} {'b'} {'r'} {'w'} {'w'} {'b'} {'g'} {'y'} {'b'} {'r'} {'b'} {'b'} {'g'} {'r'}	Yes	Successfully rotated and detected all colors
rot_5.png			Color Grid of the Image: {'b'} {'w'} {'w'} {'w'} {'g'} {'y'} {'y'} {'w'} {'b'} {'w'} {'w'} {'g'} {'y'} {'w'} {'g'} {'y'}	Yes	Successfully rotated and detected all colors

Thoughts and comments on the given real images:

Filename	Image	Comments
IMAG0032.jpg		This image is having two rectangles addition to four circles which can be useful to set a perfect position of grid and there is also a shadow in bottom left.
IMAG0033.jpg		This image is oriented than previous one and have perfect light which can be usefull to detect colors.
IMAG0034.jpg		Too much of light glare in the image which makes it difficult to detect the exact colors and circle positions.
IMAG0035.jpg		Half of the image with covered with little bit of shadow still with the helo of various filters we can remove it and will be able to find colors in this image and perform desired tasks on it.
IMAG0036.jpg		This image is good for processing but its little bit blur.

IMAG0037.jpg		This image is good for processing but there is little bit light coming from the bottom and this image is slightly projected.
IMAG0038.jpg		Too many objects are present in the image and there are many shapes in it like square and circles, so it will be little bit difficult to locate exact desired circles from the images for processing.
IMAG0041.jpg		The paper in the image is shrunk and we need to preprocess this image with some functions to work on it.
IMAG0042.jpg		This image is having a blue filter over it which can be removed using some techniques
IMAG0044.jpg		Too much blurry image it will be so difficult to detect edges and circles.

Code Appendix:

```
   %Candidate Number - 276236

   clc
   clear all

   % Prompt user to select an image file
   [filename, filepath] = uigetfile({'*.png;*.jpg;*.bmp;*.tif', 'Image Files
(*.png, *.jpg, *.bmp, *.tif)'}, 'Select an image file');
   if isequal(filename, 0)
       disp('No file selected');
   else
       processImage(fullfile(filepath, filename));
   end

   function processImage(filename)
       % Load and preprocess the image
       inputImage = loadImage(filename);

       subplot(2, 2, 1);
       imshow(inputImage);
       title('Input image');

       % Detect circle locations
       circleLocations = detectCircles(inputImage);

       % Check if at least 4 circles are detected
       if size(circleLocations, 1) < 4
           disp('Less than 4 circles detected. Cannot perform geometric
transformation.');
           return;
       end

       % Correct image distortion based on detected circles
       [correctedImage, ~] = correctImageDistortion(circleLocations,
inputImage);

       % Display the corrected image
       subplot(2, 2, 4);
       imshow(correctedImage);
       title('Corrected Image');

       % Identify colors within specific regions
       colorGrid = identifyColorRegions(correctedImage);
       disp("Color Grid of the Image:");
       disp(colorGrid); % Display color grid
   end

   %% identifyColorRegions Function
```

```
    function colorGrid = identifyColorRegions(inputImage)
        % Converting RGB colorspace to LAB
        labImage = rgb2lab(inputImage);

        colorPoints = [80 172 259 369];


        colorPointValues = zeros(16, 3);
        count = 0;

        for i = 1:4
            for j = 1:4
                count = count + 1;
                x = colorPoints(1, i);
                y = colorPoints(1, j);
                temp = labImage(x:x + 56, y:y + 56, :);
                colorPointValues(count, :) = mean(reshape(temp, [], 3), 1);
            end
        end

        % Defining the colors in RGB and LAB
        % Red, Green, Blue, Yellow, White, Purple
        rgbScale = [1 0 0; 0 1 0; 0 0 1; 1 1 0; 1 1 1];
        % r=red, g=green, b=blue, y=yellow, w=white
        colorNames = {'r', 'g', 'b', 'y', 'w'};
        labScale = rgb2lab(rgbScale);
        distances = pdist2(colorPointValues, labScale, 'euclidean');


        [~, colorIndices] = min(distances, [], 2);
        colorPatches = colorNames(colorIndices);


        colorGrid = reshape(colorPatches, 4, 4)';
    end

    % Load an image from the specified file and return it as a double precision
matrix
    function image = loadImage(filename)
        img = imread(filename);
        image = im2double(img);
    end

    % Detect the coordinates of the black circles in the image
    function circleCoordinates = detectCircles(image)
        grayImage = rgb2gray(image);

        % Threshold the image to obtain a binary image
        threshold = graythresh(grayImage);
        binaryImage = imbinarize(grayImage, threshold);
```

```
        % Invert the binary image
        invertedBinaryImage = imcomplement(binaryImage);

        % Label connected components in the inverted binary image
        cc = bwconncomp(invertedBinaryImage);

        % Calculate the area of each connected component
        areas = cellfun(@numel, cc.PixelIdxList);

        % Sort areas in descending order
        [~, sortedIndices] = sort(areas, 'descend');

        % Get the coordinates of the first four largest black blobs
        numBlobs = 5;
        blobCoordinates = zeros(numBlobs, 2);
        for i = 2:numBlobs
            blobIndices = cc.PixelIdxList{sortedIndices(i)};
            [rows, cols] = ind2sub(size(invertedBinaryImage), blobIndices);
            blobCoordinates(i, :) = [mean(cols), mean(rows)];
        end

        % Remove the first coordinate from the blobCoordinates matrix
        blobCoordinates(1, :) = [];

        % Sort the coordinates in clockwise order starting from bottom-left
        sortedCoordinates = sortClockwise(blobCoordinates);

        subplot(2, 2, 2);
        imshow(grayImage);
        viscircles(sortedCoordinates, 20, 'EdgeColor', 'b'); % Display detected
circles
        title('Circles in image');

        circleCoordinates = sortedCoordinates;
    end

    % Sort coordinates in clockwise order starting from bottom-left
    function sortedCoordinates = sortClockwise(coordinates)
        sortedCoordinates = sortrows(coordinates);
        if sortedCoordinates(2, 2) < sortedCoordinates(1, 2)
            sortedCoordinates([1 2], :) = sortedCoordinates([2 1], :);
        end
        if sortedCoordinates(4, 2) > sortedCoordinates(3, 2)
            sortedCoordinates([3 4], :) = sortedCoordinates([4 3], :);
        end
    end

    % Correct image distortion using geometric transformation
    function [outputImage, correctedCoordinates] =
correctImageDistortion(coordinates, image)
        targetBox = [[0, 0]; [0, 480]; [480, 480]; [480, 0]];
```

```
        % Calculate the transformation matrix using projective transformation
        transformationMatrix = fitgeotrans(coordinates, targetBox, 'projective');

        % Create an image reference object with the size of the input image
        outputView = imref2d(size(image));

        % Apply the transformation matrix to the input image
        transformedImage = imwarp(image, transformationMatrix, 'fillvalues', 255,
'OutputView', outputView);

        % Crop the image to a size of 480x480
        croppedImage = imcrop(transformedImage, [0, 0, 480, 480]);

        % Suppress glare in the image using flat-field correction
        flatFieldCorrectedImage = imflatfield(croppedImage, 40);

        % Adjust the levels of the image to improve contrast
        contrastedImage = imadjust(flatFieldCorrectedImage, [0.4, 0.65]);

        % Denoise the image to improve image quality
        redChannel = medfilt2(contrastedImage(:,:,1), [5 5]);
        greenChannel = medfilt2(contrastedImage(:,:,2), [5 5]);
        blueChannel = medfilt2(contrastedImage(:,:,3), [5 5]);
        outputImage = cat(3, redChannel, greenChannel, blueChannel);

        correctedCoordinates = targetBox;
    end
```

References:

1. MATLAB. (2023). uigetfile. Retrieved from https://ch.mathworks.com/help/matlab/ref/uigetfile.html
2. MATLAB. (2023). viscircles. Retrieved from https://ch.mathworks.com/help/images/ref/viscircles.html
3. MATLAB. (2023). pdist2. Retrieved from https://ch.mathworks.com/help/stats/pdist2.html
4. MATLAB. (2023). fitgeotrans. Retrieved from https://www.mathworks.com/help/images/ref/fitgeotrans.html
5. MATLAB. (2023). imwarp. Retrieved from https://www.mathworks.com/help/images/ref/imwarp.html