Image Processing Homework
Candidate No: 244835

Abstract

This is a report about the process of building an image processing application for a game. The game requires players to reconstruct images from memory after being shown an image for a short period of time. The aim of the application is to interpret and compare brick patterns created by the player against original brick patterns. The methods employed in building this application includes thresholding and colour segmentation using hue values. Images were preprocessed by denoising and filtering with a gaussian filter.Distorted images were corrected using coordinates of the black circles as anchors.

Results obtained showing an overall 83.3% accuracy in the function. Challenges faced were the incorrect representation of green and yellow in certain images. Application will perform better with presence of less noise in images.

Introduction

This is an Image processing application for a game - Lego's life of George. Lego's life of George is a game where the player is required to reconstruct a block from memory after being shown an image for a short period of time. The aim of this image processing application is to process the image reconstructed by the player. By processing the reconstructed image, it can be compared for correctness with the correct image. The basic requirement of this application is that the brick patterns from an image be correctly interpreted despite limiting circumstances. Limiting circumstances can include the possible conditions of the submitted image. Conditions such as image lighting, noise,certain degrees of rotation and camera angles(perspective from which the player captures the image), distracting background can be a barrier to the correct interpretation of colours present in the image.This application was created using the image processing toolbox in MATLAB.

Methods

This section gives an overview and description of the techniques used in the creation of this application. Design decisions are explained and possible methods to improve performance. A family function *colourMatrix* employs smaller functions which carry out independent tasks. These smaller functions are *loadImage, removeSmallElements, smoothedChannels, findCircles, correctImage and findColour.* MATLAB functions are present in Appendix A.2.

Function *loadImage*

This function reads in images of type uint8 and outputs them to type double in the range [0, 1]. The conversion from an unsigned 8 bit integer system to a floating point system enables smooth

computation and manipulation of pixel data. This is necessary for operations such as denoising, affine transformation on the images. Conversion to a floating point system also expands the range of possible values to negative values.

Function *removeSmallElements*

This function removes the presence of micro objects in the images.This objects,although insignificant to the eye, may affect computation later on. This process can be viewed as a first layer of denoising the image. This is done by eroding the image and then dilating the resulting image. The same structural element was used to maintain the form of the image as much as possible.

Function *smoothedChannels*

This function further denoises an image and outputs a filtered image. Filtering is done with MATLAB's inbuilt function imgaussfilt. To prevent colour mixing,filtering is done in individual channels r,g, and b. After which each filtered channel is concatenated to make a new,filtered rgb channel for the image.

Function *findCircles*

This function returns the coordinates of circles/ellipses within an image. Fundamentally, this is built using the knowledge that the circles/ellipses are the darkest part of the image. The image is preprocessed within the function by converting it into a floating point system,denoising and filtering. After preprocessing, the image is binarized and thresholded to include only the darkest point of the image. The number of objects is then detected using the *bwlabel* 8 connected components system, and the centroid of each consequently calculated. The reasoning for the use of centroid is to accommodate asymmetric shapes that may result due to distortion. Imfindcircles is also not preferred as it requires fine tuning, and hard to generalise using one parameter setting.

Function *correctImage*

This function corrects a noisy or distorted image.Noisy images were denoised and filtered using prior functions *smoothedChannels* and *removeSmallElements* An undistorted image is required for accurate computation of the brick patterns. However, distortion could take many form.It could be a change in camera angle or rotation of the image. To build a robust function capable of accurately undistorting any image one has to consider major forms of distortion.The implemented function *correctImage* was however implemented using an MATLAB inbuilt function *fitgeotrans* which uses the four black circle coordinates of a reference image which is undistorted as the fixed point, Moving points are coordinates of the black circles/ellipses in the distorted image.Various conditional statements were created within by switching the options

parameter in *fitgeotrans* between 'projective',and ‘affine’ depending on the form of distortion. This provides a transformation matrix which is used to transform the distorted image. Image is undistorted using *imwarp* and the size constrained using *Rfixed.* A form of improvement would be the ability to detect the form of distortion from image data instead of the file name as implemented using the set of simulated images.

Function *findColours*

The input of this function is the smoothed rgb double array to be interpreted.The function converts the rgb format to hsv using the *rgb2hsv* function. Output from this is then separated into three different arrays, each one representing the hue, saturation and value of the pixels in the image.The reason behind this is that separation of colours is sufficient using hues alone, as each colour in the image represents a hue, as opposed to the rgb scheme which could have values in more than one array (as certain secondary colours are made up of a mixture of certain proportions of red, green or blue)

After the separation into hues, two for loops are created to segment the images vertically and horizontally into smaller squares, such that the median of hue pixel intensities in the square represents the hue of the square. The output of the median is then scaled to 360(a scale which represents all hue values). Conditional statements using the standard hue scale and outputs the string r,g,b or y depending on the hue value present.

This is why the importance of the denoising is emphasised, the presence of noise will heavily skew the values thus leading to a wrong output of present colours,the median was also used as a sort array will have values for noise as outliers. This method also hinges on the correct and precise representation of each box taking into account the starting coordinates and the padding between each box. Values used in this report are a box size of 75 and padding of 10. Empty string arrays were also initialised to improve the speed of the program.

Function colourMatrix

This is a family function which implements and combines prior functions mentioned. This function was hard coded with ‘org_1_png’ as the reference image. Circle coordinates were obtained from this reference image using *findCircle* and then was used as input to *correctImage* in addition to the input image.The output is a colour matrix obtained with the function *findColours.*

Results and Discussion

Results

The results shown in Appendix A.1 are for a set of the simulated images. Presence of gaussian noise was noticed in this set of images.This affected output despite denoising and filtering. Certain bricks with tints of yellows and greens were unable to be detected possibly due to the

presence of noise which pushes the median hue intensity beyond the hard coded values in *findColours. loadImage* recorded an accuracy of 100%, *findColours* recorded an accuracy of 83.3%, unable to correctly interpret 5 of the 30 simulated images.correctImage recorded an accuracy of 93.3%, unable to correct 2 out of 30 images.By extension the main function *colourMatrix* recorded an accuracy of 83.3%(What is the accuracy)

Discussion of Real Photos

For the set of images below, the conditions of the image are assessed, and the implications for our image processing application analysed. The coordinates of the black circles and thin rectangular stripes are used as fixed points to reorient the image in *fitgeotrans* to create consistent orientation across the set of images. Functions are written in *italics* and present in Appendix B

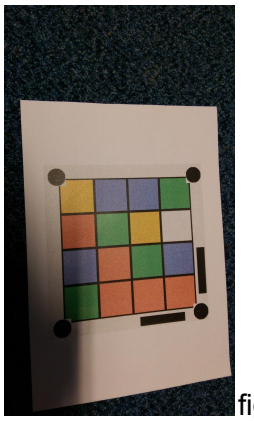


fig 1 : IMAG0032

Figure 1 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix as output. The unbalanced lighting condition should not pose much of an issue, because the affected area is minor and the image processing application implements hues as our basis for interpretation. Shades of blue and green will still be interpreted as green hues.Although the image looks slightly warped, the function *undistortimage* corrects using the coordinates of the rectangular strips as moving points for this to give a square .

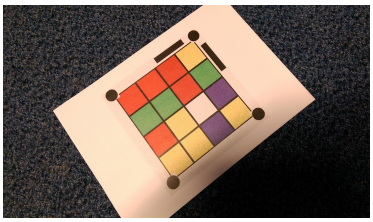


fig2: IMAG0033

Figure 2 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix. Unbalanced Lighting conditions will not cause a problem in interpretation as all the square bricks are evenly lit. Figure 2 is rotated and the function *undistortimage* corrects for this to give a square along the x- axis and y-axis. Rotation correction done with

undistortimage is carried out by setting the “options” parameter in matlab’s inbuilt function fitgeotrans to “nonreflectivesimilarity”. Orientation is also corrected using the rectangular strips coordinates as “movingpoints” for consistency with the other images in this set.

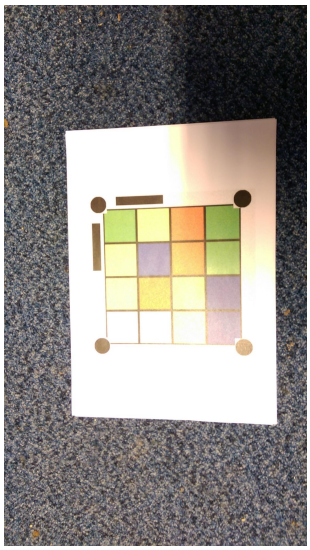


fig3:IMAG0034

Figure 3 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix. Lighting condition is bad, and will lead to an incorrect interpretation. This is due to unbalanced lighting conditions where the colours at the lower section of the image are bleached.Yellows can be misinterpreted as white, reds as yellow due to this overexposure. Orientation is also corrected using the coordinates of the rectangular strips as moving points in the function *undistortimage*

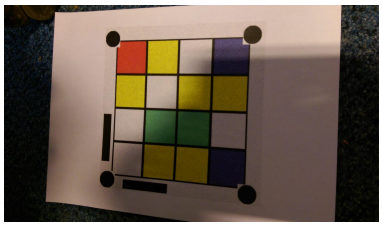


fig4:IMAG0035

Figure 4 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix. Figure 4 is slightly warped and has an unbalanced lighting condition. The unbalanced lighting condition could pose a problem in the interpretation of white as a colour within the squares. Image position and orientation is corrected using *correctImage*

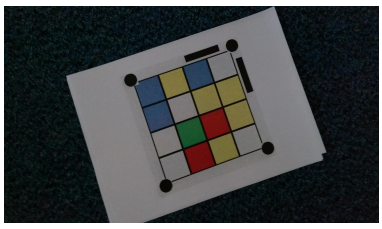


fig5:IMAG0036

Figure 5 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix. Figure 5 is slightly warped and has an even lighting condition. An accurate representation of the brick patterns is expected after a correction in orientation using correctImage.

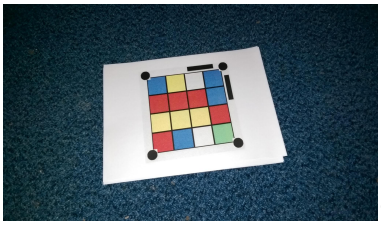
 fig6:IMAG0037

Figure 6 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix. Figure 6 is slightly warped and has an even lighting condition.After a correction in the orientation to match other images in this set, the image processing application should conveniently interpret the colour patterns on this image

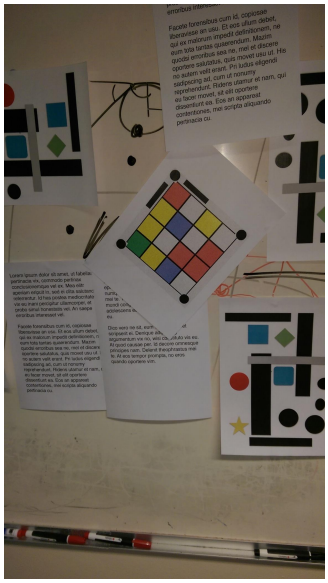
 fig7:IMAG0038

Figure 7 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix as output. Figure 7 is rotated, has an even lighting condition but there are lots of other distracting patterns present. The presence of other distracting patterns will interfere with the output, a possible solution is to crop the image to the region of interest before passing it to the image processing application.

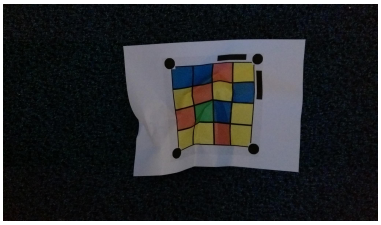
 fig8:IMAG0041

Figure 8 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix as an output. Figure 8 is an image of a squeezed piece of paper where vertical and horizontal lines are now curves of varying degrees and has an even lighting condition.The lighting condition does not affect the output, but the distorted vertical and horizontal lines do. This is corrected with the function *correctImage,* the “option” parameter in matlab’s inbuilt function *fitgeotrans* is changed to “polynomial”. This corrects the curved lines and outputs a correct colormatrix

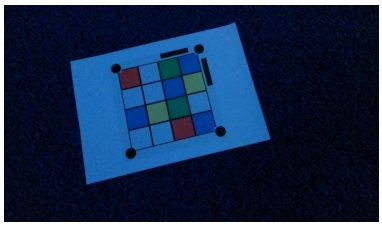


fig8:IMAG0042

Figure 9 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix. Figure 9 is an image with a blue tint(probably shooting with a cool white balance) and has an balanced lighting condition.The effect of the blue tint on the image is on individual pixel intensities. Provided the conditional statements in *findColours* are recalibrated to factor in the effect of the blue tint across the image, the colour matrix outputted will be correct else the output may not be entirely accurate. Orientation is also corrected for consistency in the position of the rectangular strips across the images in this set.

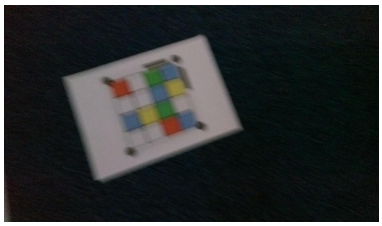
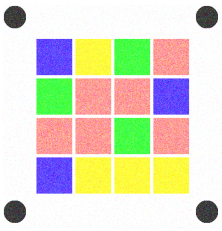


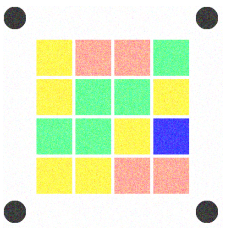
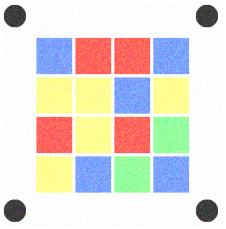
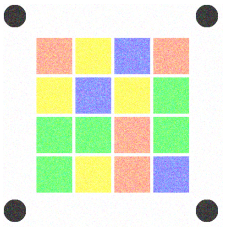
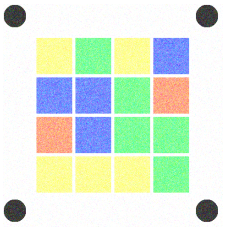
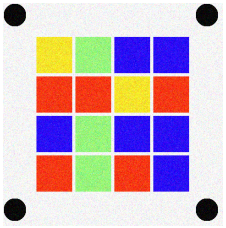
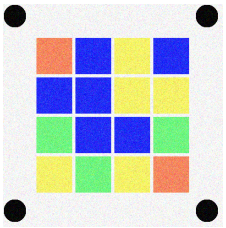
fig9:IMAG0044

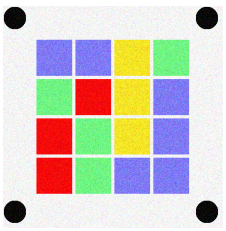
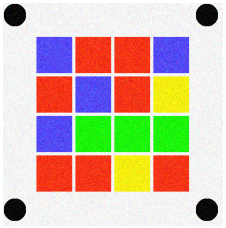
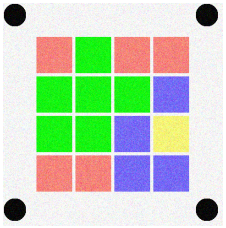
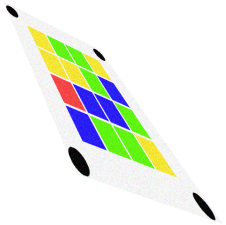
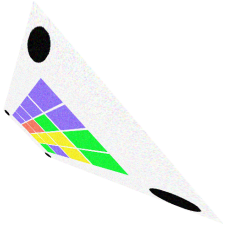
Figure 10 is an image of brick patterns for our image processing application to interpret and thus produce a colormatrix. Figure 9 is a blurred rotated image with even lighting conditions. The blur may be due to the camera shake at the point of taking the picture. The problem this blur causes is edge detection of objects within the scene as it results in overlaps of objects. This renders the *findcoordinates* function inaccurate and consequently the color matrix obtained.

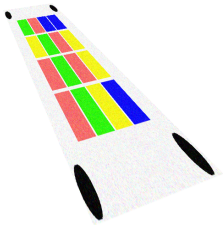
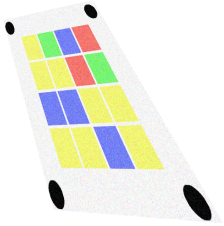
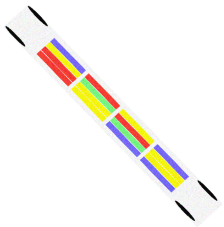
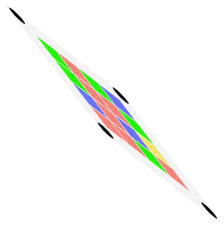
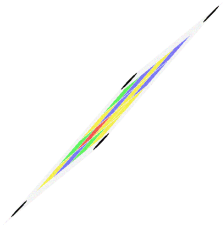
Appendix A

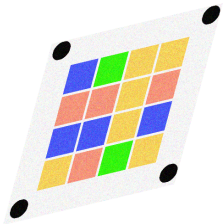
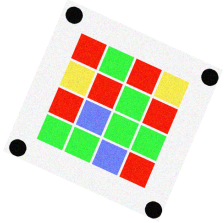
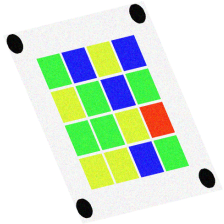
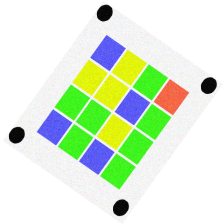
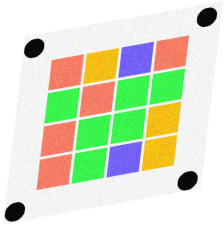
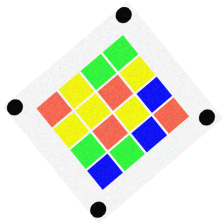
A.1 Results

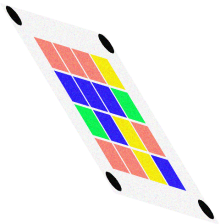
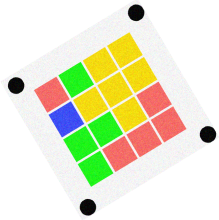
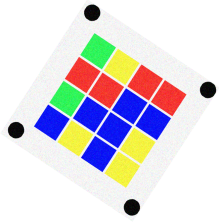
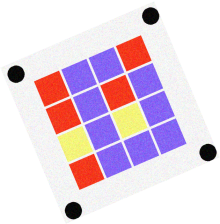
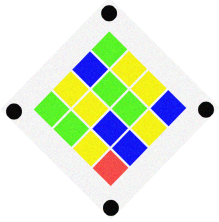
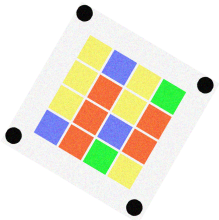
Filename	Image	Output	Success	Notes
Noise_1		b y g r g r r b r r g r b y y y	YES	

Noise_2		y r r "" y "" "" y "" "" y b y y r r	NO	
Noise_3		b r r b y y b y r y r g y b g b	YES	
Noise_4		r y b r y b y g g g r g g y r b	YES	
Noise_5		y g y b b b g r r b g g y y y g	YES	
org_1		y g b b r r y r b g b b r g r b	YES	
org_2		r b y b b b y y g b b g y g y r	YES	

org_3		b b y g g r y b r g y b r g b b	YES	
org_4		b r r b r b r y b g g g r r y r	YES	
org_5		r g r r g g g b g g b y r r b b	YES	
proj_1		y g g y y y g b b b b g b g g y	YES	
proj_2			NO	Circles/ellipses are too small to detect
proj_3		r r g r r y g g b b b g r b y y	YES	

proj_4		g y y y r g g g b y y y b y y b	NO	Orientation of image has changed - a counterclockwise rotation of less than 90 degrees followed by a flip on it's x-axis. -certain bricks of red are interpreted as yellow
proj_5		y g b r y y r g b b y y y y b y	YES	
proj1_1		r y b b r y g y y g r y b r y b	YES	
proj1_2		g r g b g r r g b r b y r r g r	YES	
proj1_3			NO	Circles are too small to be identified

proj1_4		y r g y b b y b r r y r b g y y	YES	
proj1_5		g g b r r b g g y r g r r g r y	YES	Orientation of image is flipped about it's x-axis followed by a counterclockwis e rotation of less than 90 degrees
proj2_1		g b y b y g b g g g y r y y b g	YES	
proj2_2		b g b g g g y g y y b g b y g r	YES	Orientation of image is flipped about it's x-axis followed by a counterclockwis e rotation of less than 90 degrees
proj2_3		r g b y r g g y g r g g r y b r	YES	
proj2_4		r y g g y y r y g r y b b g b r	YES	Orientation was changed with a clockwise rotation of less than 90 degrees.

proj2_5		r b "" r r b b r r b y y y "" r b	NO	Orientation was changed with a counterclockwise rotation of less than 90 degrees. -Unable to detect green.
rot_1		r g y y b y y y g g y r g r r r	YES	Orientation was changed with a clockwise rotation of less than 90 degrees.
rot_2		b y b y g b b y r r b b g y r y	YES	Orientation has changed. Flipped around the x-axis with counterclockwise rotation of less than 90 degrees.
rot_3		r b b r r b r b y b y b r b b b	YES	Orientation was changed with a clockwise rotation of less than 90 degrees.
rot_4		g g y r y y b b b g g y g y y b	YES	Orientation has changed. Flipped around the x-axis with counterclockwise rotation of less than 90 degrees.
rot_5		b r g y y r b r y r y r y b y g	YES	

A.2 Matlab Code

```
image1 = 'org_1.png'; image2 = 'proj2_5.png';
figure(1)
imshowpair(loadImage(image1),loadImage(image2),"montage")
colormat = colourMatrix(image2);
function [output] = colourMatrix(filename)
%Uses circle coordinate to undistort new image
org_circ_coord = findCircle('org_1.png');
correctedImage = correctImage(org_circ_coord, filename);
figure(2)
imshowpair(loadImage(filename),correctedImage,"montage")
output = findColours(correctedImage);
end
function [croppedImage] = correctImage(circleCoordinates,img)
% Function un-distorts the remaining images using the
% circle coordinates as an anchor
b= loadImage(img); a = loadImage('org_1.png');
centroids = findCircle(img);
%Undistort
%centroids: Moving points are circle coordinates of distorted images
%circleCoordinates:Fixed points are circle coordinates of undistorted images
if startsWith(img, 'org_')
   denoised = removeSmallElements(b);
   croppedImage = smoothedchannels(denoised,1.5);
end
if startsWith(img, 'noise_')
   denoised = removeSmallElements(b);
   croppedImage = smoothedchannels(denoised, 3.0);
end
if startsWith(img,'proj_')
%for warped transformation, use projective
mytform = fitgeotrans(centroids, circleCoordinates, 'projective');
%creating fixed size of output image using ortho image as a reference
Rfixed = imref2d(size(a));
%constrains registered to this size
registered = imwarp(b,mytform,OutputView=Rfixed);
%crops to size of reference image
croppedImage = imcrop(registered, [0,0,480,480]);
end
if startsWith(img, 'proj1_')
%for parallel transformation, use affine
mytform = fitgeotrans(centroids, circleCoordinates, 'affine');
%creating fixed size of output image using ortho image as a reference
Rfixed = imref2d(size(a));
%constrains registered to this size
registered = imwarp(b,mytform,OutputView=Rfixed);
%crops to size of reference image
croppedImage = imcrop(registered, [0,0,480,480]);
```

```
end
if startsWith(img,'proj2_')
%for parallel transformation, use affine
mytform = fitgeotrans(centroids, circleCoordinates, 'affine');
%creating fixed size of output image using ortho image as a reference
Rfixed = imref2d(size(a));
%constrains registered to this size
registered = imwarp(b,mytform,OutputView=Rfixed);
%crops to size of reference image
croppedImage = imcrop(registered, [0,0,480,480]);
end
if startsWith(img, 'rot_')
%for parallel transformation, use affine
mytform = fitgeotrans(centroids, circleCoordinates, 'projective');
%creating fixed size of output image using ortho image as a reference
Rfixed = imref2d(size(a));
%constrains registered to this size
registered = imwarp(b,mytform,OutputView=Rfixed);
%crops to size of reference image
croppedImage = imcrop(registered, [0,0,480,480]);
end
end
function [org_circ_coord] = findCircle(image)
%Finds the coordinates of the four black circles
img = loadImage(image);
denoised_a = removeSmallElements(img);
new_ref = smoothedchannels(denoised_a, 1.5);
ref_img_bw = im2bw(new_ref,0.1);
%inverts the mask for better identification
ref_img_bw = ~ref_img_bw;
%use of bwlabel for precision in identification of objects
[ref_label,~] = bwlabel(ref_img_bw,8);
%calculates centroids of each shape as coordinate
k = regionprops(ref_label,"Centroid");
%concatenates each coordinate value into an array
org_circ_coord = cat(1,k.Centroid);
end
function [new_rgb] = smoothedchannels(new_img, thresh)
ref = new_img;
ref_red = ref(:,:,1);
ref_green = ref(:,:,2);
ref_blue = ref(:,:,3);
ref_red_smth = imgaussfilt(ref_red, thresh);
ref_green_smth = imgaussfilt(ref_green,thresh);
ref_blue_smth = imgaussfilt(ref_blue,thresh);
%Creates new smoothed Rgb space
new_rgb = cat(3, ref_red_smth, ref_green_smth, ref_blue_smth);
end
function [new_img] = removeSmallElements(img)
```

```
%Erode
se = strel("disk",1);
ero_img = imerode(img,se);
%Dilate
dil_img = imdilate(ero_img,se);
new_img = dil_img;
end
function [col_arr] = findColours(new_img)
%Finds the colours from a double image array, and returns the result of the
%colours
hsv = rgb2hsv(new_img);
%Finding Hue
h = hsv(:,:,1);
box_size = 75;
padding = 10;
col_arr = strings(4,4);
%Moving across the square verically and horizonatally
for i=0:3
   for j= 0:3
       hue = h(75+(i*box_size)+padding*i:150+(i*box_size)+padding*i,
75+(j*box_size)+padding*j:150+j*box_size+padding*j);
       %Uses median, this way pixel values for noise are outliers
       M = median(hue, 'all') * 360;
       if M < 20 || M > 330
           col_arr(i+1,j+1) = "r";
       end
       if M > 90 && M < 135
           col_arr(i+1,j+1) = "g";
       end
       if M > 195 && M < 265
           col_arr(i+1,j+1) = "b";
       end
       if M > 20 && M < 80
           col_arr(i+1,j+1) = "y";
       end
   end
end
end
function [img] = loadImage(filename)
%Takes in a file of type uint8 and outputs an
% array of type double
img = imread(filename);
if isa(img,'uint8')
   img = double(img)/255;
else
   error("This image is a of unkown type")
end
end
```