AMME4701

Photometric Stereo

Computer Vision & Image Processing

Vishant Prasad

Assignment Task 1

Table of Contents

Section 1 Introduction

Computer vision in general is concerned with the extraction of information from a scene within images. Most commonly information that can be obtained from the images includes various types of shapes of objects, estimates of features and applied to desirable tasks within realms of surgery, surveillance, preserving historical artefacts and developing navigational aids.

As technology continues to develop with camera qualities and greater resolutions, the world of computer vision can assist both services of professional fields as well hobbyists and the public.

1.2 Motivation

The impact of varying illuminate direction on the heading of the textures or surfaces allows for classification of the texture scheme, whereby the classification depend upon the appearance of the surface rather than the topology. It would be preferrable to employ surface properties rather than image properties so intrinsic characteristic of the surface need to be recovered prior to modelling or classification of the surface itself.

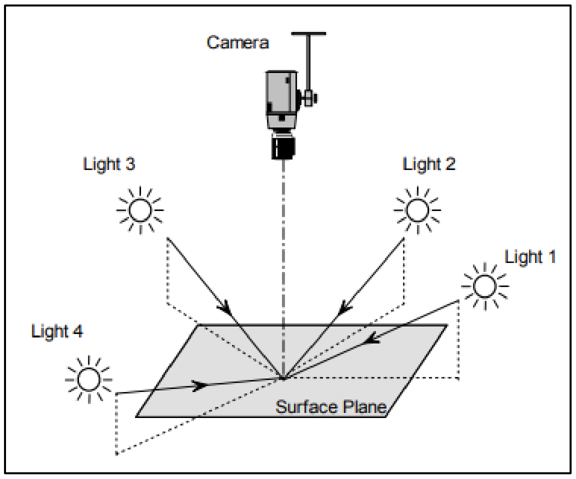
Assuming **Lambertian Reflectance Model** the image intensity of a surface at a point (x, y) can be determined from the orientation $[p(x,y), q(x,y)]$. Since there is an infinite number of surface orientations that can consequentially result in the same value of intensity – a unique surface orientation cannot be determined from a single image intensity or radiance value[1].

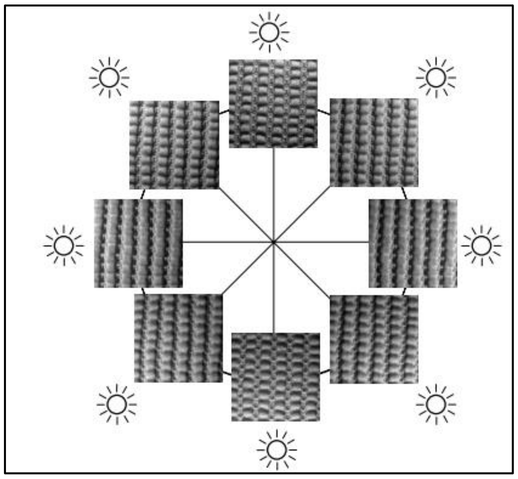
(Wu, 2003)

The intensity has **one-degree** of **freedom** and thus the surface orientation (p, q) has **two-degrees** of **freedom**. Hence, to determine the surface orientation *Photometric Stereo* is a technique that can aid in additional information from multiple images of a static object.

1.3 Photometric Stereo

Photometric Stereo is a technique within computer vision for estimating the surface normal of objects. It is undertaken by capturing several images of a static scene from the same viewpoint, while alternating the illumination direction upon the object. The significance of this practice is the acquired images will have pixels corresponding to the same object point. Applications of the photometric stereo primarily are for producing an image with enhanced contrast and removing surface noise. The object being captured must be stationary hence, allowing for the results of shape and reflectance to be collected. The models can, therefore, be developed to characterise image radiance with reference to the illumination of the environment, generating or allowing for transformations of image intensities of brightness or light.





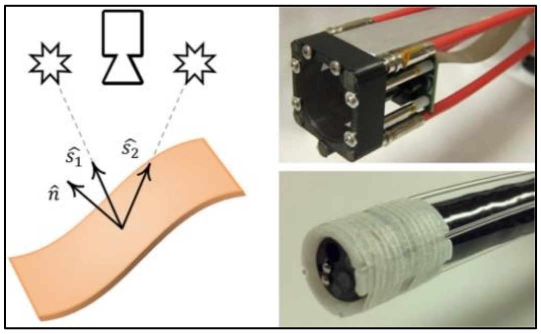
1.3.1 Advantages

Photometric stereo is a method desired due to problems with shading being able to be resolved. Using several images gains advantages as:

1) Unlike single image shape from shading algorithms, photometric stereo makes no assumption of the smoothness of the surface (p and q vary continuously over the surface)[2]. *(Shiradkar, 2014)*
2) Only additional lighting is required hence, minimising computational cost and implementation technicalities.
3) With each additional image, the reflectance map is unique, therefore, defining a unique set of orientations for each point pixel.
4) Can recover not only surface orientation (p, q) but also surface albedo.

1.3.2 Applications

Applications of photometric stereo is often evident within society in fields of health and research, primarily with 3D reconstructions with one example being in *Endoscopy* (see *figure*). For such applications, the computer assisted image processing allows for assistance in surgery, whereby renderings, estimates of shapes and reflectance is a priority.[3]



(Parot et al.)

Key Terms – *Endoscopy, Lambertian Reflectance Model, Degree of Freedom, Photometric Stereo*

Section 2 Methodology

Aim: To implement a basic shape from shading algorithm. The input to the algorithm is a set of photographs taken with known lighting directions and the output of the algorithm is the albedo, normal directions, height map or 3D reconstruction and outlier detection and rejection algorithm or modification.

The methodology of this project has been divided into two primary sections, that is the **implemented algorithms** whose results are outputted successfully and operational fully with the program designed and the **incomplete implementations** being the functions, modifications to design and future implementations that can be made with continual progress on this project.

2.1 Implemented Algorithms

2.1.1 Surface

The reconstruction of the surface of the image uses the surface normal retrieved from the 64 different images with different illumination conditions. The calculation or methodology of the surface normal involved using surface normal with the integration method.

Primarily this involved the following steps:

1) Calculating the fx and fy information using a MATLAB division of arrays of different surface normal data.
2) Creating a switch for each of the different methods for integration strategies.
3) Method is inputted by user currently with the system design – no error detection for input has been implemented at this stage.
4) Original design involved using loops to calculate height map by summing rows and cols of the fx and fy data.
5) Adjusted the integration methods to utilize a cumulative summation function in built to MATLAB.

Col:

- Moving through fx data and summing all columns.
- Moving through fy data and summing all rows.
- Calculate the sum of each of these data arrays and hence, output to the height map data array.

Row:

- Moving through fx data and summing all rows.
- Moving through fy data and summing all columns.
- Calculate the sum of each of these data arrays and hence, output to the height map data array.

Avg:

- Moving through fx data and summing all rows.
- Moving through fy data and summing all columns.
- Calculate the sum of each of these data arrays and hence, output to the row height map data array.
- Moving through fx data and summing all rows.
- Moving through fy data and summing all columns.
- Calculate the sum of each of these data arrays and hence, output to the column height map data array.
- Calculate the average of these two heigh map data arrays and output to a new heigh map data array.

2.1.4 Albedo

The albedo image has been calculated using methods of computing the expected image intensity in an 8-bit image of the size defined. The following steps are following to achieve an output for the albedo image:

1) Assign albedo data array to desired data array.
2) Loop through the image and reshape the intensity information of the image to allow for calculation of scale factor (g).
3) Calculate scale factor using light directions divided by the intensity.
4) Calculate surface normal using scale factor information.
5) Calculate albedo using square root summation of the scale factor to the power of two.

The pseudocode for this has been defined below:

```
albedo = double(zeros(h,w));

for i = 1:h
    for j = 1:w

        I = reshape(im(i,j,:),64,1);
        g = light_dirs\I;

        for k = 1:3

            I = reshape(im(i,j,:),64,1);
            g = light_dirs\I;

            normal(i,j,k) = g(k,1);

            albedo(i,j) = sqrt(sum(g.^2));
        end
    end
end
```

The pseudocode of the implementation of the implementation and output of the albedo data array is as follows:

```
G = zeros(height,width,3);
P = reshape(imArray,height*width,channel);

P = p';
G = lightDirs\p;
G = g';

albedoImage = zeros(height,width);
g = reshape(G,height,width,3);
surfaceNormals = zeros(height,width,3);

for I = 1:height
    for j = 1:width

              vectors = G(i,j,:).*G(i,j,:);
              dot_p = sum(vectors);
              albedoImage(i,j) = sqrt(dot_p);
```

```
            surfaceNormals(i,j,:) = G(i,j,:)./albedoImage(i,j);
    end
end
```

2.1.5 Height Map

The height map itself is calculated using the normal by integration methods which is denoted more specifically in the surface section above (*Section 2.1.1*). Ideally each integration method utilizes the normal and is plotted using surface plot within MATLAB. The following section below highlights a reference to where some of the specifications have been found to gain a comprehensible plot:

```
function display_face_model(albedo_image, height_map)

    % display_face_model - Shows a 3D surface model of a heightmap, coloured by
    % the matrix albedo_image
    %
    % albedo_image: h-by-w matrix of albedo values, values between 0 and 1
    % height_map: h-by-w matrix of height values (any range)

    [h,w] = size(height_map);
    [X,Y] = meshgrid(1:w, 1:h);

    figure
    mesh(X, Y, height_map, albedo_image);
    axis equal;

    xlabel('X')
    ylabel('Y')
    zlabel('Z')
    title('Height Map')
    set(gca, 'XTick', []);
    set(gca, 'YTick', []);
    set(gca, 'ZTick', []);
    set(gca, 'XDir', 'reverse')

    colormap(gray)
 end
```

2.1.6 Photometric Stereo

To summaries the photometric stereo implementation, the function utilises the past three functions and algorithms by firstly running the calculation of the albedo and surface normal of the photograph, captured using the point light sources from different directions.

Assumptions:

- The surface is assumed to be perfectly Lambertian so that the measured intensity is proportional to the albedo times the dot product between the surface normal and lighting direction.
- The lights are assumed to be of unit intensity.

Primarily this involved the following steps:

1) Read the light sources using the light directions data.

2) Complete calculation of the scale factor and each image.
3) Calculate albedo image.
4) Calculate surface normal data using albedo image data.

2.1.7 Outlier Detection

The residual is calculated using the intensity measured by the original image containing the brightness subtracted by the rendered values brightness. The predicted values has been calculated using the albedo image methodology. Once the residual has been collected, a greyscale conversion is performed to allow for processing and calculation of intensity. Like calculating the intensity values for the albedo and scale factor, an intensity calculation is performed using the residual greyscale image using the same methodology as the photometric section above (*Section 2.1.6*).

2.1.8 Outlier Rejection

Now that the comparison is made between each pixel brightness and the residual – through the comparison algorithm, the threshold is therefore set and only pixel brightness below this threshold is compiled. A new normal then can be calculated using only these acceptable intensities, ported into the already standing surface algorithm.

2.2 Incomplete Implementations

2.2.1 Preparing Data

The data can be pre-processed to subtract the effect of ambient light from each of the images. This is an implementation that has been omitted for the testing and most of the results since the dataset had minimal ambient light as default and since with this dataset no ambient image information has been provided. Through research the following section highlights a possible implementation of dealing with ambient interference. This can be implemented given the ambient image data and will output the illumination of the image with less ambient light or interference, with more of a single point light source result.

Primarily this involved the following steps:

1) Subtract the ambient image array from each of the images in the original loaded array (for each image).
2) Make sure no pixel is less than zero.
3) Rescale the values in the dataset arrays to be between 0 and 1.

2.2.2 Function Formatting

The current system has no optimization to the coding design and only the pseudocode of some of the stemming functions have been created such as photometric stereo, preparing data, sections of the main running program script, display output and the surface heightmap.

Furthermore, the program as this moment has various repetitive loops mainly with the outlier detection and rejection system which can be isolated into separate functions.

Furthermore, pseudocode for the main script has been optimised to read datasets and files in full and had been planned to perform all calculations, displays and plots into a computer file for the user to inspect or use as a function to save the plots.

```
dataDir     = fullfile('D:','Vishant','School','University','Year 5', 'S2',
'AMME4710','Assignment 1','week2_facedata');
```

```
subjectName = 'yaleB05';%debug, yaleB01, yaleB02, yaleB05, yaleB07
numImages   = 128;       % Total images for each surface
writeOutput = true;      % If true then the output will be written to an output file
outputDir   = fullfile('D:','Vishant','School','University','Year 5', 'S2',
'AMME4710','Assignment 1','week2_facedata','output');
imageDir    = fullfile(dataDir, 'photometricStereo', subjectName);
integrationMethod = 'column';
```

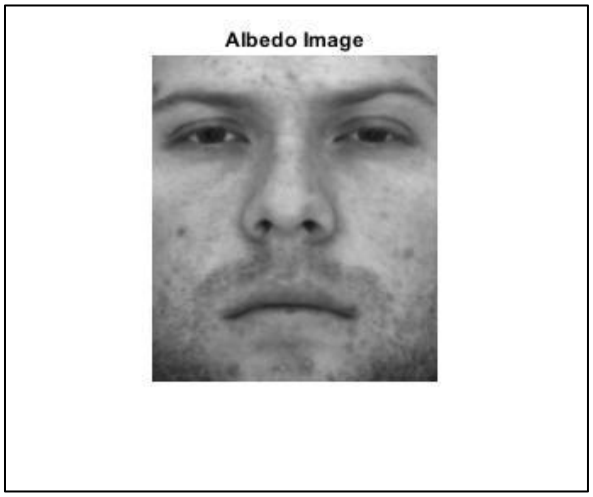
Finally, the user input for the integration method but this is not optimised as the user input has no error checking for invalid inputs.

Section 3 Results & Discussion

3.1 Results

3.1.1 Albedo

The albedo image output gives us insight into what we can expect from the surface normal and 3D reconstruction, being some inaccuracies around the lips due to flatness and some higher intensities in places that are collectively suppressed.

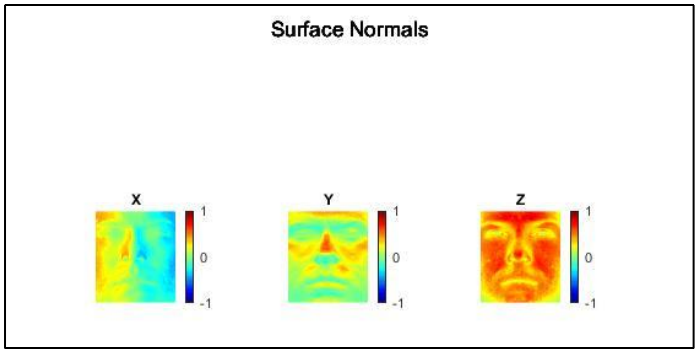


3.1.2 Rendered Montage

The rendered montage of each of the images in the dataset can be seen within the appendix section *figure*. As expected we can note that higher intensity light is being supressed around the forehead on some images, various shadows are existence due to the various lighting angles and flatness of places that are shadowed such as the lips. These consequences can be reasoned by the distance of light from the object when performing photometric stereo and the inter-reflectance between adjacent surfaces when assuming Lambertian Surface Reflection Model.

3.1.3 Surface Normal

To envision the following 3D height map, the surface normal allows an insight into the light intensities upon each axis. It can be concluded that the outputted surface normal plots correlate with the resulting 3D reconstruction. We can see the lips losing accuracy due to the x-axis plot, but the lips gains definition which is accurate in the z-axis.

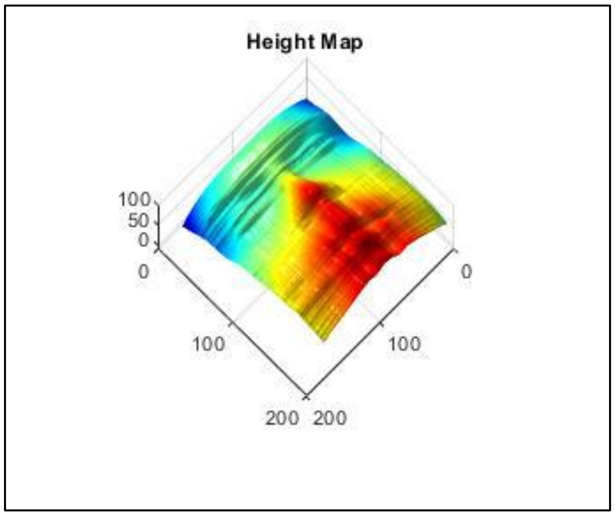


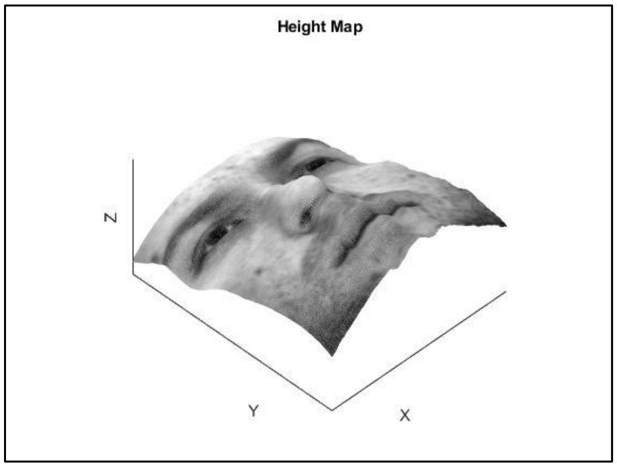
3.1.4 3D Height Map

The 3D reconstruction has been most successful with the each of the integration methods more comprehensibly denoted int the appendix section under *figures*. As expected there is displacement upon all three-axis accurate to the expected facial features and the colour mapped images allow for an insight into the light distribution and illumination upon each of the faces.

It can be noticed that the higher intensity light is being suppressed such as in areas of the forehead. The various shadows near the nose influences the reconstruction of the lips. The definition of the lips is varied correlating with the more flat features show by the rendered plot.

Through these results we can see these outputs since we are assuming the Lambertian Surface Reflection model and therefore, distance of the light from the object plays an effect in the output as well as the inter-reflectance between adjacent surfaces.



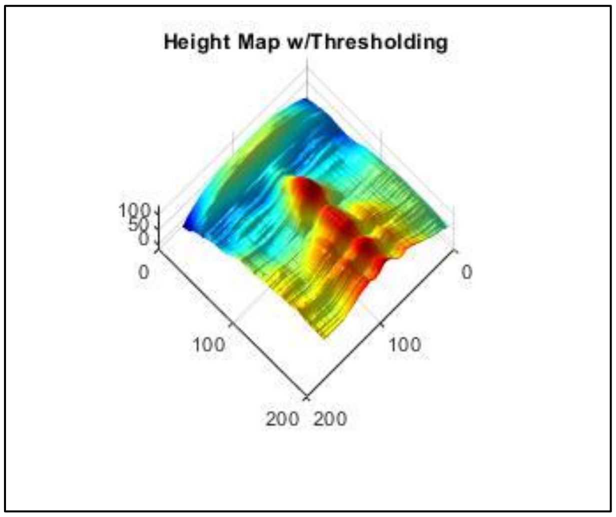


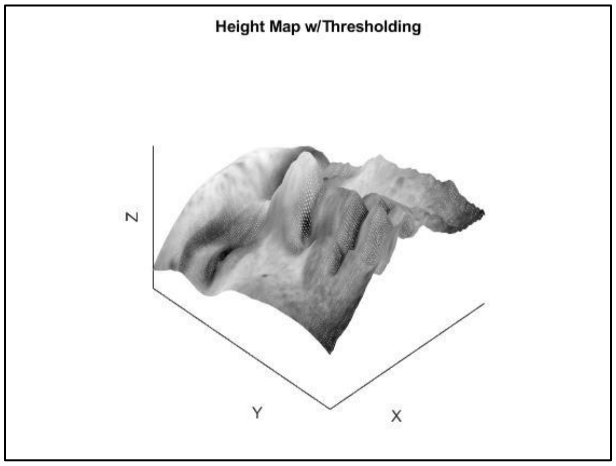
3.1.5 Residual Montage

The following depicts the output of a montage of the residual values plotted onto a single figure. Thresholding has had an effect of amplification of the intensities that are detected and suppression of intensities in other areas. We can more clearly pick out which pixels are amplified when overlayed upon the original albedo montage as seen in the appendix *figure* below. Therefore, the output is more comprehensible with this overlayed comparison denoting the higher intensity and brightness areas being those which are omitted by the threshold.

3.1.6 3D Height Map with Thresholding

The following depicts the output of the 3D height map plot with the outlier detection and rejection algorithm employed. Thresholding has had an effect of amplification of the intensities that are detected. Therefore, outliers must be successfully removed since on sections of the bridge of the nose and the lips where intensity is of high values have been omitted. The resulting plot, therefore, is a more rough output with more jagged facial aspects as expected.





3.2 Discussion

With all the tests performed we can see with the 3D reconstruction of data set two, there is mostly expected outputs across all methods. In contrast, data set five depicts an unusual occurrence around the neck region which can be seen in the appendix section, *appendix B figure*. This would be due to the face of not occupying the entirety of the frame and hence, regions around the face and neck especially display lighting upon a region that is offset drastically than other features on the face. The reflectance of many of the illuminations, therefore, cause a confusion to the algorithm with the neck being mistaken as closer to the camera since it is intense with lighting upon some images or angles of light illumination.

The performance of each integration method is very clearly illuminated through the colour mapped plots also in appendix section, *appendix B figure*. We can notice that this is performing as expected with the pixels and plotting being patterned or textured either horizontal, vertical or both for row, column, or average integration methods respectively.

The dataset one and seven also show interesting 3D reconstructions upon all methods with some of the height maps being slightly flatter. This is due to some of the interference we can see by the rendered montage plot. Furthermore, this extends to the thresholding, causing more emphasised plots of the facial regions of lower light intensity.

3.3 Conclusion

A photometric stereo solution has been achieved with a range of differing datasets. The photometric images, furthermore, has been used to recover both the normal surface orientation and albedo information given the assumptions stated within the report and methodologies employed. Overall, the outcome has been as expected and defined however, functionality of the program can be optimised to allow for a more general and suitable response from a range of different datasets.

4.1 References

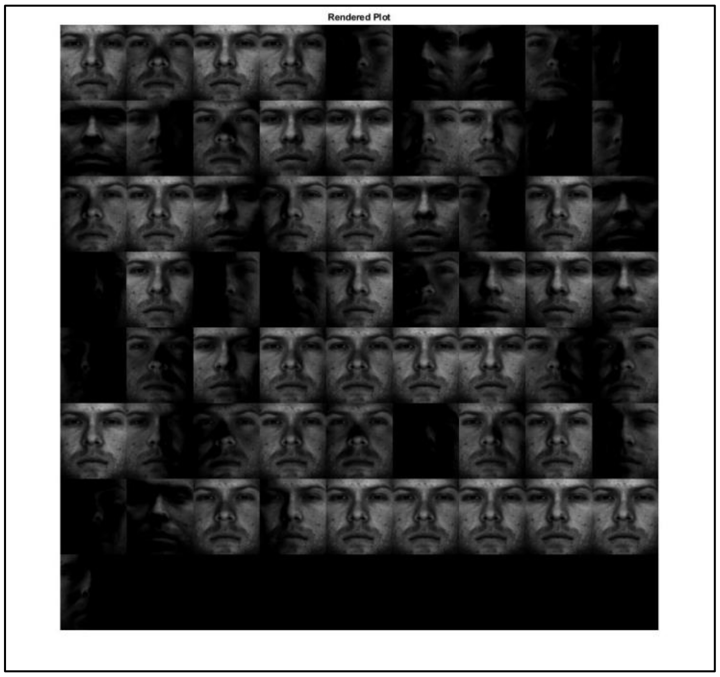
[1] Parot, V. et al., Photometric Stereo Endoscopy. Journal of Biomedical Optics. Available at: https://www.spiedigitallibrary.org/journals/Journal-of-Biomedical-Optics/volume-18/issue-7/076017/Photometric-stereo-endoscopy/10.1117/1.JBO.18.7.076017.full?SSO=1 [Accessed September 1, 2021].

[2] Wu, J., 2003. Chapter 4 - Photometric Stereo . Rotation Invariant Classification of 3D Surface Texture Using Photometric Stereo, pp.63–79. Available at: http://www.macs.hw.ac.uk/texturelab/files/publications/phds_mscs/JW/chapter4.pdf [Accessed September 1, 2021].

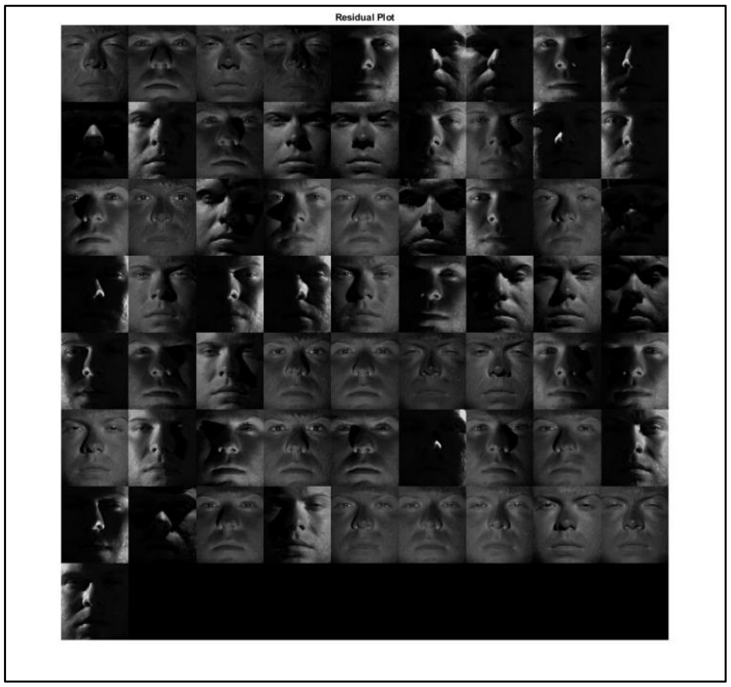
[3] Parot, V. et al., Photometric Stereo Endoscopy. Journal of Biomedical Optics. Available at: https://www.spiedigitallibrary.org/journals/Journal-of-Biomedical-Optics/volume-18/issue-7/076017/Photometric-stereo-endoscopy/10.1117/1.JBO.18.7.076017.full?SSO=1 [Accessed September 1, 2021].

5.1 Appendix

5.1.1 Appendix A

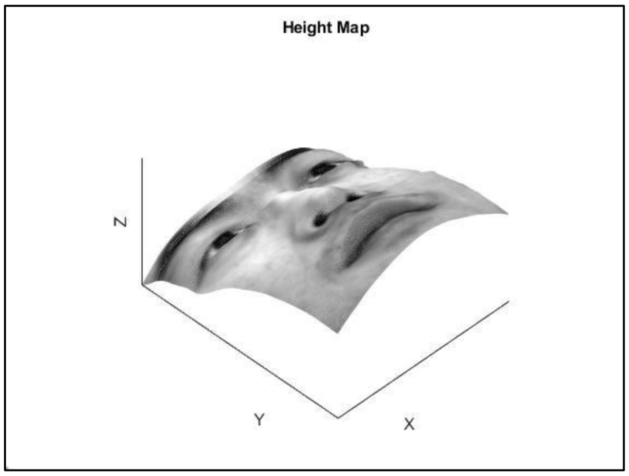


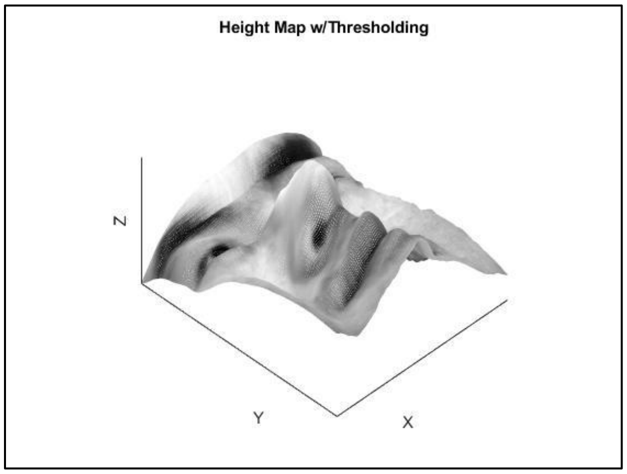
Rendered Plot



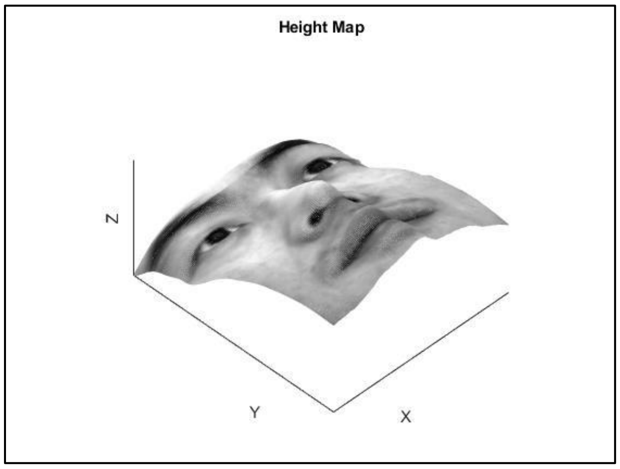
Residual Plot

5.1.2 Appendix B

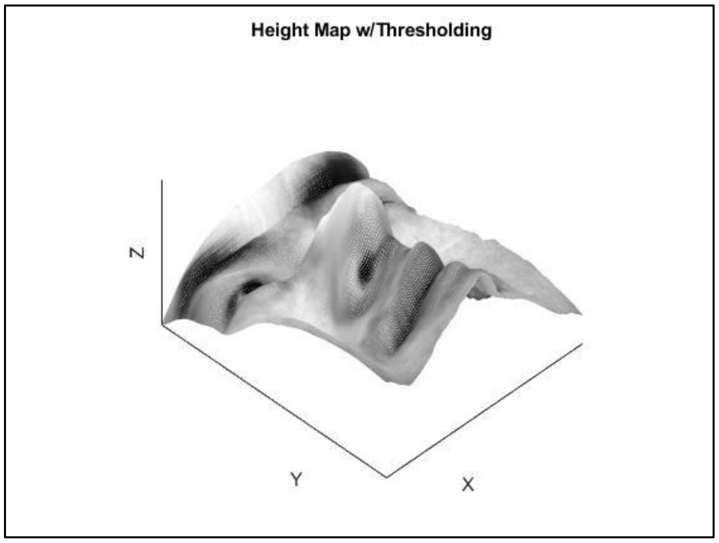




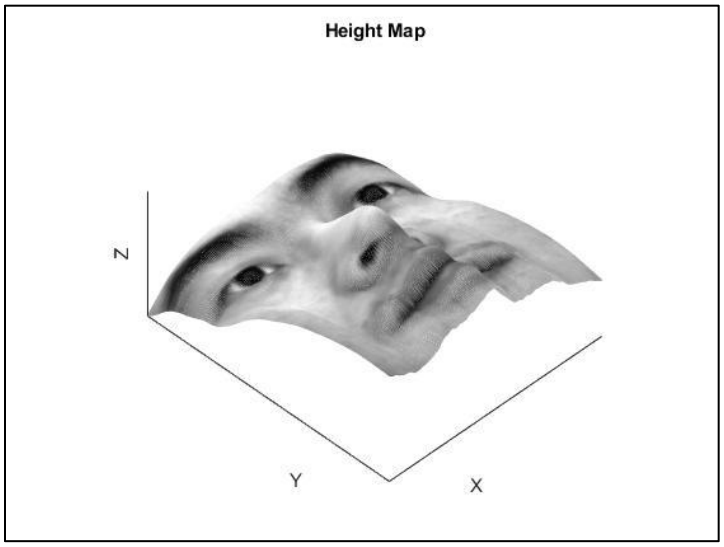
Other Dataset Performance



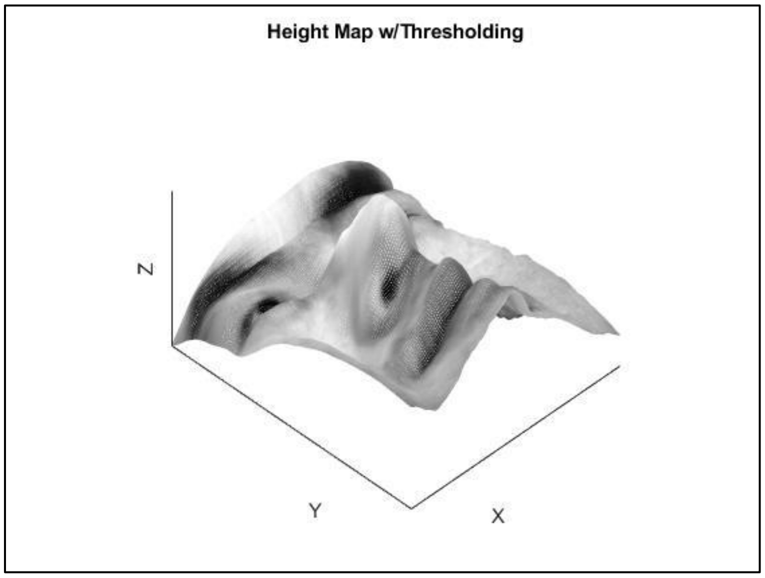
Average Integration Method



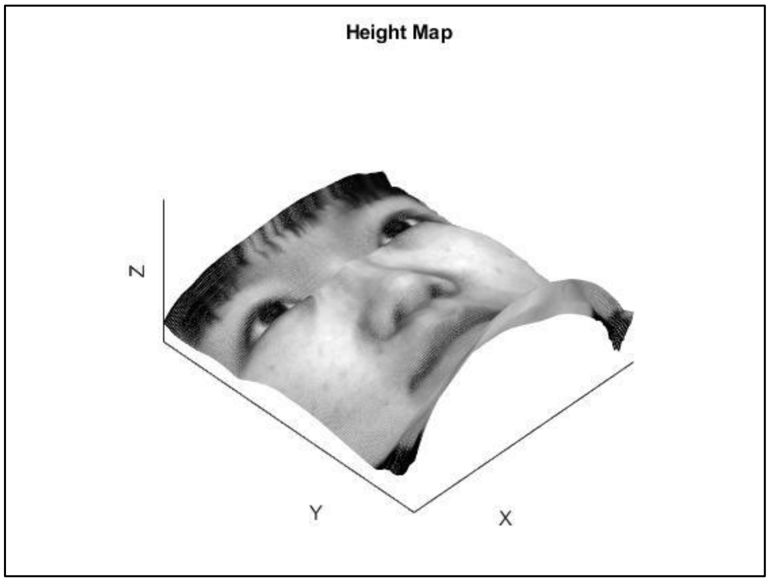
Average Integration Method with Thresholding



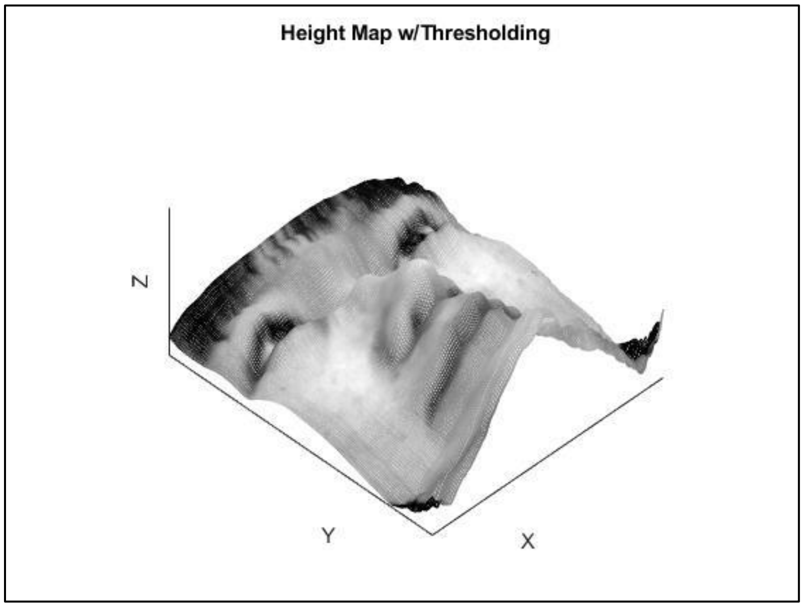
Column Integration Method



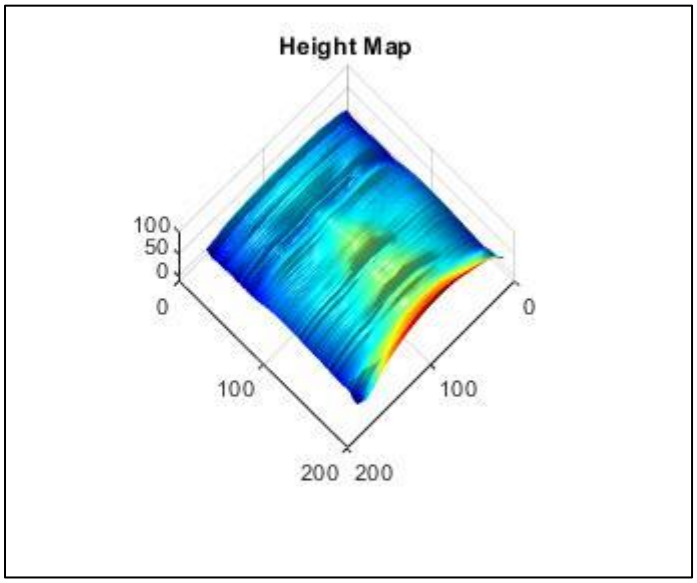
Column Integration Method with Thresholding



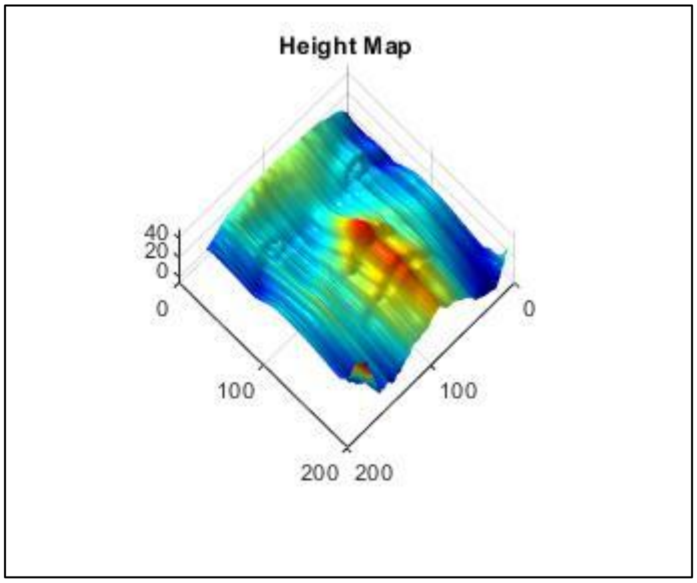
Row Integration Method with Dataset 5



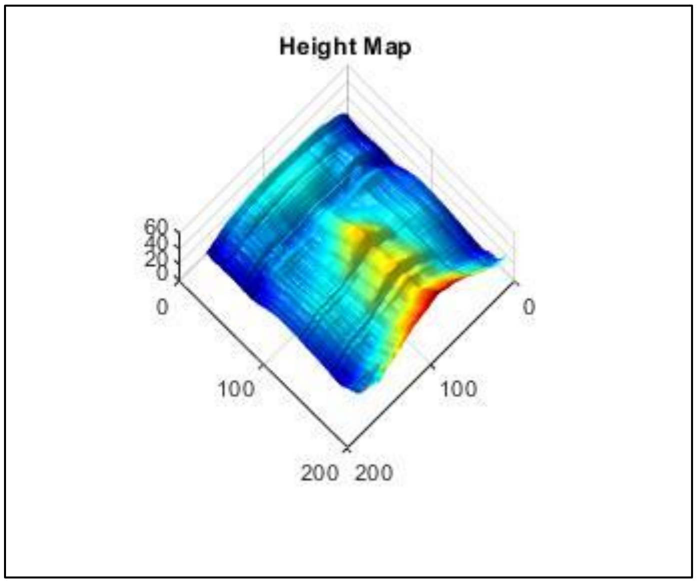
Row Integration Method with Thresholding Dataset 5



Row Integration Method



Column Integration Method



Average Integration Method

6.1 Code

6.1.1 Question 1 – Complete Design

```
clear
clc
clf

% load dataset
load('facedata_yaleB05.mat');

% divide into dimensions
[h,w,N] = size(im_array);

% removing ambient light omitted
% ch = size(im_array,3);
% for i = 1:ch
%     im_array(:,:,i) = im_array(:,:,i) - ambientImage(:,:);
%     im_array(im_array(:,:) < 0) = 0;
%
%     im_array(:,:,i) = im_array(:,:,i)./max(max(im_array(:,:,i)));
% end

% convert to grey
im = mat2gray(im_array);

% define variables for scale factor
g = zeros(h,w,N);
p = reshape(im_array,h*w,N);

p = p';

albedo = double(zeros(h,w));
normal = double(zeros(h,w,3));

% ask user for integration method
prompt = 'Choose method of integration (row, col, avg): ';
method = input(prompt,'s');

% perform surface normal calculation
for i = 1:h
    for j = 1:w
        I = reshape(im(i,j,:),64,1);
        g = light_dirs\I;

        for k = 1:3

            % calculate scale factor
            I = reshape(im(i,j,:),64,1);
            g = light_dirs\I;

            normal(i,j,k) = g(k,1);

            % output albedo image
            albedo(i,j) = sqrt(sum(g.^2));
        end
    end
end

% perform calculation to gain predicted intensity values
for im = 1:N
    for i = 1:h
        for j = 1:w
           I_pred(i,j) = albedo(i,j)*reshape(normal(i,j,:),1,3)*light_dirs(im,:)';

           % collect residual data into array
           rendered(i,j,im) = uint8(255*I_pred(i,j));
           residual(i,j,im) = im_array(i,j,im) - rendered(i,j,im);
        end
    end
end

% convert residual to greyscale
im_2 = mat2gray(residual);

% calculate new intensity values using residual
for i = 1:h
    for j = 1:w
        I_r = reshape(im_2(i,j,:),64,1);
        g_r = light_dirs\I_r;

        for k = 1:3

            I_r = reshape(im_2(i,j,:),64,1);
            g_r = light_dirs\I_r;
        end
    end
end

% compare the intensities for thresholding
j = 1;
for i = 1:N

    if I_r(i) < I(i)

        I_o(j) = I(i);
        j = j + 1;
    else

        I_o(j) = 0;
        j = j + 1;
    end
end

% calculate the new surface normal with new thresheld intensity values
for i = 1:h
    for j = 1:w
        I_o = reshape(im_2(i,j,:),64,1);
        g_o = light_dirs\I_o;
        for k = 1:3

            I_o = reshape(im_2(i,j,:),64,1);
            g_o = light_dirs\I_o;
            normal_r(i,j,k) = g_o(k,1);
        end
    end
end

% plot the rendered montage
figure(5);
montage(reshape(rendered,h,w,1,N));
title('Rendered Plot');

% plote the residual montage
figure(6);
montage(reshape(residual,h,w,1,N));
title('Residual Plot');

% define dimensions of the surface normal data array
h = size(normal,1);
w = size(normal,2);
n = size(normal,3);

h_r = size(normal_r,1);
w_r = size(normal_r,2);
n_r = size(normal_r,3);

fx = zeros(h,w);
fy = zeros(h,w);

fx_r = zeros(h_r,w_r);
fy_r = zeros(h_r,w_r);

% perform integration of the surface normals
for i = 1:h
    for j = 1:w

        fx(i,j) = normal(i,j,1)./normal(i,j,3);
        fy(i,j) = normal(i,j,2)./normal(i,j,3);

        fx_r(i,j) = normal_r(i,j,1)./normal_r(i,j,3);
        fy_r(i,j) = normal_r(i,j,2)./normal_r(i,j,3);
    end
end

% generate a switch for each of the desired integration methods
switch method

    % integrate in the vertical direction first along the first column,
    % then across each row
    case 'col'
%         heightMap = cumsum(fx,2)+cumsum(fy,1);
```

```
        heightMap_o = cumsum(fx_r,2)+cumsum(fy_r,1);
        for k = 1:h
            for m = 1:w

                heightMap(k,m) = sum(fx(1,1:m))+sum(fy(1:k,m));
            end
        end

    % integrate in the horizontal direction along the top row first,
    % then down each column
    case 'row'
%         heightMap = cumsum(fy,1)+cumsum(fx,2);
        heightMap_o = cumsum(fy_r,1)+cumsum(fx_r,2);
        for k = 1:h
            for m = 1:w

                heightMap(k,m) = sum(fy(1:k,1))+sum(fx(k,1:m));
            end
        end

    % the average of height computer by the previous two methods
    case 'avg'
%         heightMap_r = cumsum(fy,1)+cumsum(fx,2);
%         heightMap_c = cumsum(fx,2)+cumsum(fy,1);

        heightMap_or = cumsum(fy_r,1)+cumsum(fx_r,2);
        heightMap_oc = cumsum(fx_r,2)+cumsum(fy_r,1);
        for k = 1:h
            for m_1 = 1:w

                heightMap_c(k,m_1) = sum(fy(1:k,1))+sum(fx(k,1:m_1));
            end
        end

        for j = 1:h
            for m_2 = 1:w

                heightMap_r(j,m_2) = sum(fx(1,1:m_2))+sum(fy(1:j,m_2));
            end
        end

        % calculate the average of the collected row and col heightmap
        % values
        heightMap = (heightMap_r+heightMap_c)/2;
        heightMap_o = (heightMap_or+heightMap_oc)/2;
end

% plot the surface normals by axis
figure(1);
subplot(1,3,1);
imagesc(normal(:,:,1), [-1 1]); colorbar; axis equal; axis tight; axis off;
title('X')

subplot(1,3,2);
imagesc(normal(:,:,2), [-1 1]); colorbar; axis equal; axis tight; axis off;
```

```
title('Y')

subplot(1,3,3);
imagesc(normal(:,:,3), [-1 1]); colorbar; axis equal; axis tight; axis off;
title('Z')

% colormap gray;
colormap jet;

% setting the 3D plot configuration
sgtitle('Surface Normals');

[hgt, wid] = size(heightMap);
[X,Y] = meshgrid(1:wid, 1:hgt);
B = rot90(h,2);
A = rot90(albedo,2);

[hgt_r, wid_r] = size(heightMap_o);
[X_r,Y_r] = meshgrid(1:wid_r, 1:hgt_r);
B_r = rot90(h_r,2);
A_r = rot90(albedo,2);

% plot the albedo image
figure(2)
imshow(albedo);
title('Albedo Image');

figure(3);
imshow(heightMap, [0 max(max(heightMap))]);
surfl(heightMap, 'light');
shading interp;
title('Height Map')

view([45, 45, 300]);
colormap jet;
% colormap gray;

% Plotting the original height map in greyscale
figure(4);
mesh(X, Y, heightMap, albedo);
axis equal;

xlabel('X')
ylabel('Y')
zlabel('Z')
title('Height Map')
set(gca, 'XTick', []);
set(gca, 'YTick', []);
set(gca, 'ZTick', []);
set(gca, 'XDir', 'reverse')

view([45, 45, 60]);
colormap gray;
```

```
% Plotting the original height map with colourspace to see lighting/
% intesity effects
figure(7);
imshow(heightMap_o, [0 max(max(heightMap_o))]);
surfl(heightMap_o, 'light');
shading interp;
title('Height Map w/Thresholding')

view([45, 45, 300]);
colormap jet;
% colormap gray;

% Plotting the Height Map or 3D reconstruction with outlier detection &
% rejection
figure(8);
mesh(X_r, Y_r, heightMap_o, albedo);
axis equal;

xlabel('X')
ylabel('Y')
zlabel('Z')
title('Height Map w/Thresholding')
set(gca, 'XTick', []);
set(gca, 'YTick', []);
set(gca, 'ZTick', []);
set(gca, 'XDir', 'reverse')

view([45, 45, 60]);
colormap gray;
```

6.2.1 Question 1 – Incomplete Optimised Design

Main

```
% More optimised main file to run the photometric stereo and 3D reconstruction
% Does not include the outlier detection and rejection
dataDir     = fullfile('D:','Vishant','School','University','Year 5', 'S2',
'AMME4710','Assignment 1','week2_facedata');
subjectName = 'yaleB05';       % Set or test for, yaleB01, yaleB02, yaleB05, yaleB07
numImages   = 64;              % Total images for each surface
outputDir   = fullfile('D:','Vishant','School','University','Year 5', 'S2',
'AMME4710','Assignment 1','week2_facedata','output');
imageDir    = fullfile(dataDir, 'photometricStereo', subjectName);

% Requesting use input for the integration method
prompt = 'Choose method of integration (row, col, avg): ';
method = input(prompt,'s');

% Load the ambient image information into a new data array
[ambientImage, imArray, lightDirs] = loadFaceImages(imageDir, subjectName,
numImages);

% Prepare the data for algorithm
imArray = prepareData(imArray, ambientImage);

% Break up scene into shape and albedo
```

```
[albedoImage, surfaceNormals] = photometricStereo(imArray, lightDirs);

% Compute height from normals by integration
heightMap = getSurface(surfaceNormals, integrationMethod);

% Display the output
displayOutput(albedoImage, heightMap);
plotSurfaceNormals(surfaceNormals);
```

Surface

```
function  heightMap = getSurface(surfaceNormals, method)

    % define dimensions of the surface normals data array
    h=size(surfaceNormals,1);
    w=size(surfaceNormals,2);
    n=size(surfaceNormals,3);

    % prepare arrays for integration methods
    fx=zeros(h,w);
    fy=zeros(h,w);


    for i=1:h
        for j=1:w

            % perform integrations and place values into prepared integration
            % data arrays
            fx(i,j)=surfaceNormals(i,j,1)./surfaceNormals(i,j,3);
            fy(i,j)=surfaceNormals(i,j,2)./surfaceNormals(i,j,3);
        end
    end

    % create a switch for each of the desired methods selected by the user
    switch method

        % integrate in the vertical direction first along the first column,
        % then across each row
        case 'col'

            % Optimisable using either the loop functionality or the
            % cumlative summing function.
            % heightMap = cumsum(fx,2)+cumsum(fy,1);
            for k = 1:h
                for m = 1:w

                    heightMap(k,m) = sum(fx(1,1:m))+sum(fy(1:k,m));
                end
            end
        % integrate in the horizontal direction along the top row first,
        % then down each column
        case 'row'

            % Optimisable using either the loop functionality or the
            % cumlative summing function.
            % heightMap = cumsum(fy,1)+cumsum(fx,2);
            for k = 1:h
                for m = 1:w

                    heightMap(k,m) = sum(fy(1:k,1))+sum(fx(k,1:m));
                end
            end
        % the average of height computer by the previous two methods

        case 'avg'
            % Optimisable using either the loop functionality or the
            % cumlative summing function.

            % heightMap_r = cumsum(fy,1)+cumsum(fx,2);
            % heightMap_c = cumsum(fx,2)+cumsum(fy,1);
            for k = 1:h
                for m_1 = 1:w

                    heightMap_c(k,m_1) = sum(fy(1:k,1))+sum(fx(k,1:m_1));
                end
            end

            for j = 1:h
                for m_2 = 1:w

                    heightMap_r(j,m_2) = sum(fx(1,1:m_2))+sum(fy(1:j,m_2));
                end
            end

            % calculate the average of the collected row and col heightmap
            % values
            heightMap = (heightMap_r+heightMap_c)/2;
end
```

Prepare Data

```
% Prepare the data by removing the ambient light from the image data for optimal
processing

function prepareData(im_array, ambientImage)

    ch = size(im_array,3);

    % Subtract the ambientImage from each image in the dataset
    for i = 1:ch

        im_array(:,:,i) = im_array(:,:,i) - ambientImage(:,:);

        % Make sure no pixel is less than zero
        im_array(im_array(:,:) < 0) = 0;

        % Rescale the values in the dataset to be between 0 and 1
        im_array(:,:,i) = im_array(:,:,i)./max(max(im_array(:,:,i)));
    end
end
```

Display Height Map

```
% Function to display the 3D reconstruction on a plot

function displayOutput(albedo, height)

    % NOTE: h x w is the size of the input images
    % albedo: h x w matrix of albedo
    % height: h x w matrix of surface heights

    % Some cosmetic transformations to make 3D model look better
    [hgt, wid] = size(height);
    [X,Y] = meshgrid(1:wid, 1:hgt);
    H = flipud(fliplr(height));
    A = flipud(fliplr(albedo));

    figure, imshow(albedo);
    title('Albedo');

    figure;
    mesh(H, X, Y, A);
    axis equal;
    xlabel('Z')
    ylabel('X')
    zlabel('Y')
    title('Height Map')

    % Set viewing direction
    view(-60,20)
    colormap(gray)
    set(gca, 'XDir', 'reverse')
    set(gca, 'XTick', []);
    set(gca, 'YTick', []);
    set(gca, 'ZTick', []);
end
```

AMME4701

Lego Block Color-Based Tracking

Computer Vision & Image Processing

Vishant Prasad
Assignment Task 1

Table of Contents

Section 1 Introduction

This research paper introduces a method of using color thresholds to identify certain objects and classifications to given objects through using a combination of RGB, Binary and HSV. The methodologies including image color detect the images 3D colour space for thresholding. The color detection algorithm is executed utilizing MATLAB. This color detection algorithms that are covered have applications extended onto security, spy robots, tracking and color-based object isolation and intrusion detection.

1.2 Motivation

From papers proposed through history, such as N.Otsu (1979)[1] the threshold mechanically has been derived from the point of view of discriminant analysis. Therefore, evaluating how optimal threshold is associate with maximising the discriminant measure of the grey levels. G.Wyszecki (1982)[3] proposed research regarding colour and strategies showing that RGB used to acknowledge the shading within the image. He proposed that RGB shading model that joins red inexperienced associated with blue lights. Furthermore, in Eddins (2004)[10] , his paper uses MATLAB digital image process for ways to detect shapes and color of objects, whereby having conversion of RGB image to grey scale images and so the black and white (binary) image has additional information.

1.3 Colour Based Tracking

In Abadpour (2005)[4], his paper concerned with colour image processing explains colour recognition through comparisons of each pixel in an image. Within this paper it was clear that the colour dominant in an image would cause the given object to have interference and inaccuracies for identification of other objects. Furthermore, Senthamaraikannan D (2014)[5] in his paper denotes color recognition to have applications and process in real-time as well as extracting the primary colours of the image to allow for vision-based computer algorithms and interaction programmable for users. Therefore, vision primarily is focused on segmentation which could be an extension of the research involved in this paper. Difficulties faced also included cluttered backgrounds, unknown lighting conditions and multiple moving objects which could be addressed by segmentation.

Key Terms – *MATLAB, RGB, Binary, HSV*

Section 2 Methodology

Aim: The primary objectives of this research is to read an input image, separate into the RGB spectrum of colour for the image, compute and plot the histogram information through applications and tools in MATLAB for further processing and conversion to grey-scale. The images consist of LEGO bricks, with the goal being for identification of the bricks themselves and classifying the colour of the bricks.

2.1 Implemented Algorithms

The following can be a summary of all the steps involved and implemented currently into the program:

1) Image Acquisition
2) Store Image
3) Define the Color Desired
4) Convert the Image into Binary Image
5) Label the Colour & Identify the Component
6) Find the Centroid for a Bounding Box
7) Colour Detected

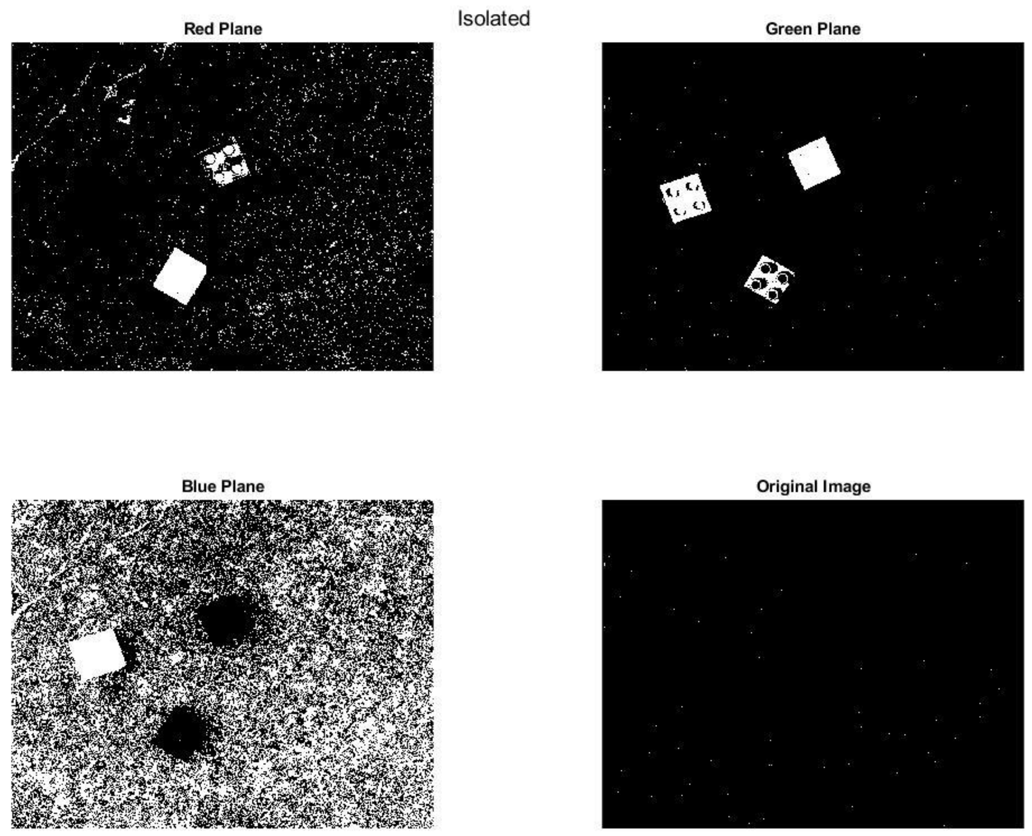
2.1.1 Reading the Input Image

The process of reading the image has not been made most optimal, as at this stage concentration on program design and optimality has not been prioritised rather the performance and accuracy of the colour identification and reduction of interference from the background, lighting, and noise. The resolution of the image is a three-dimension matrix of size $H \times W \times 3$ where each dimension of the RGB is an RGB format representing a colour picture.

2.1.2 Generating Colour Bands

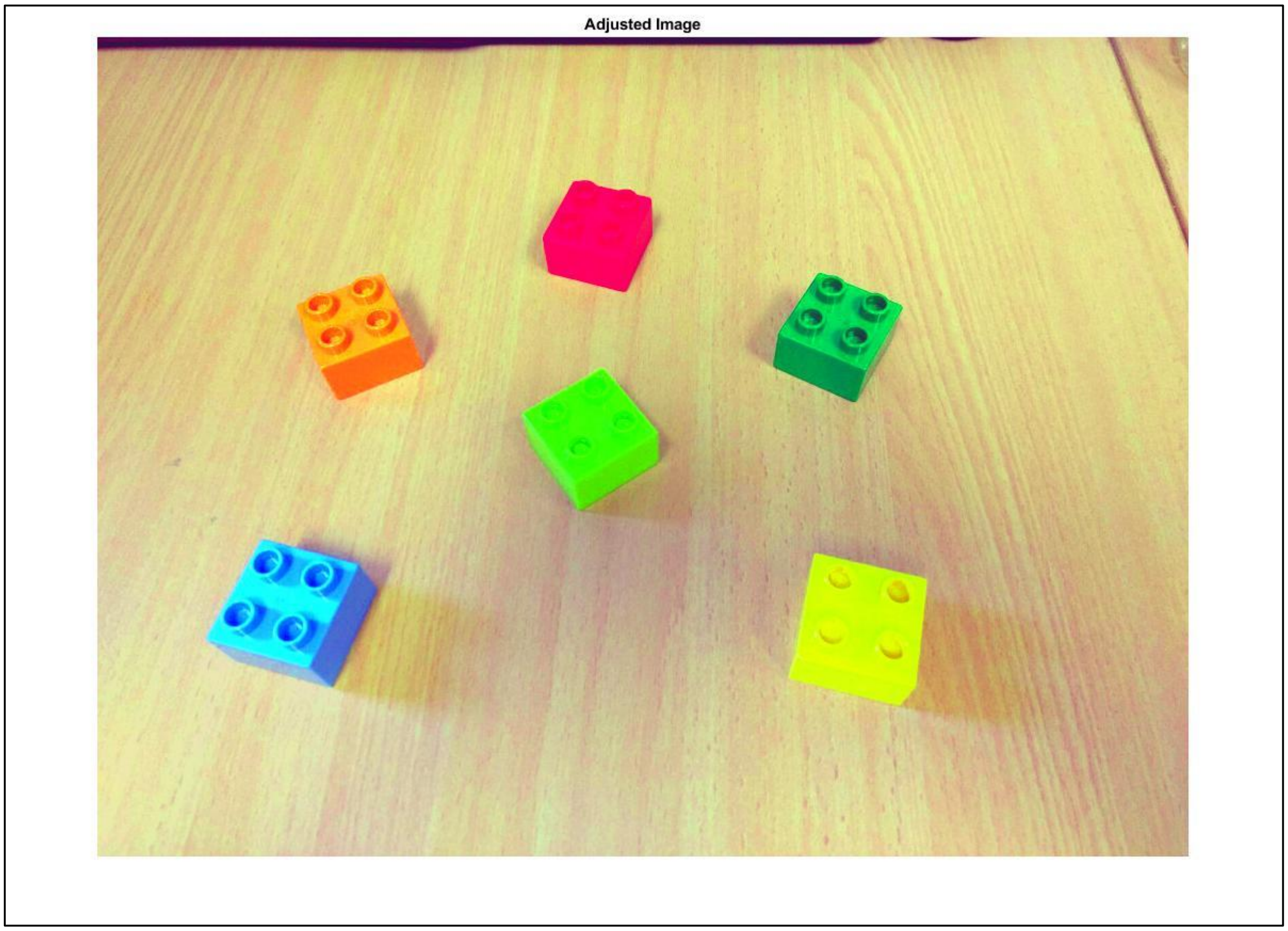
The images consist of pixels that corresponds to specific combinations of the colour and therefore, the primary functions that are created utilize Colour Thresholding within the HSV or RGB realm. This is selected based on the colour's separation and region on a 3D plot when analysed.

When thresholding is accomplished the binarized image data can be saved and therefore, for each image the red, green, and blue colour bands can be generated as seen in *figure* below. Therefore, using these generated binarized colour bands we can detect the primary images in each of the images.



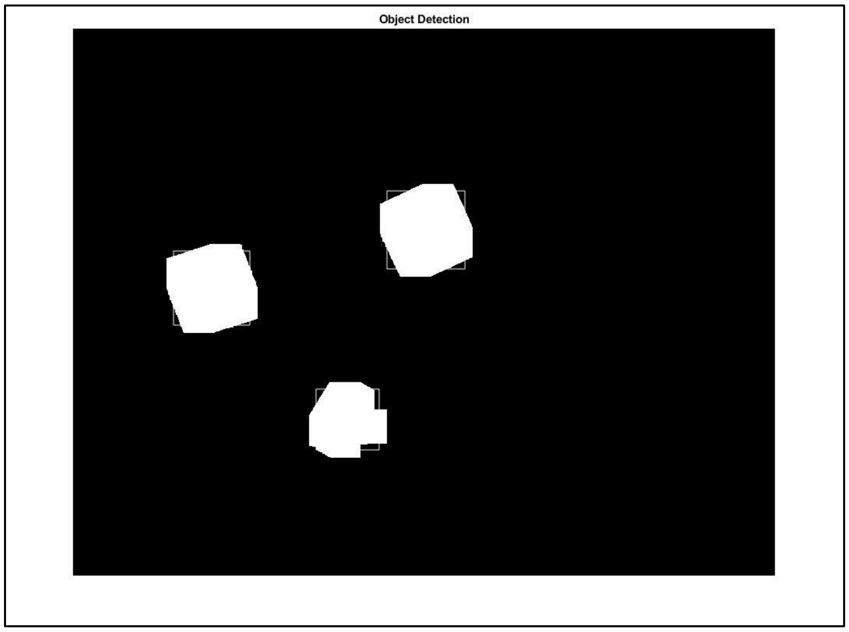
2.1.3 Removing Noise

We can also eliminate factors of noise by using a cropped image eliminating possible edges that have other high intense colours. Furthermore, using a stretching of the greyscale or RGB image means the colours are made more intense and easier to distinguish, possibly even limited a maximum colour value therefore, making most of the brick's colours similar and reducing effects of lighting upon each block. The decorrelation can also be applied to stretch the multispectral image. Therefore, the mean and variance in each band are same as in the original image. The primary purpose is like the previous function of stretching limits whereby a visual enhancement is made to the differences in the image.



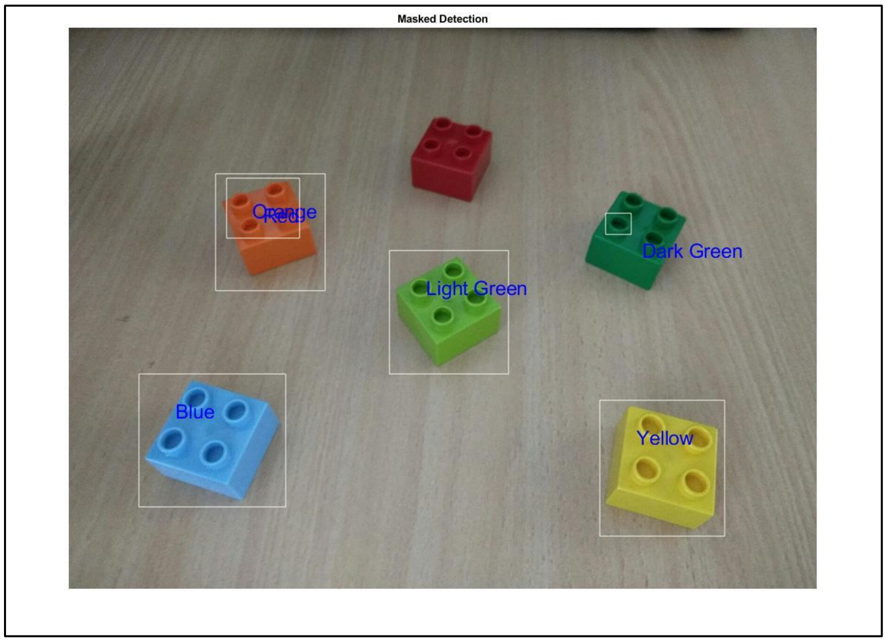
2.1.3 Object Detection

Further object detection is accomplished by removing the holes and small noise from the binary image and filling all inconsistencies on the bricks themselves. The bounding boxes are placed through calculation of the centroids (x, y) positions via creation of blobs for each of the detected colours and hence, these can be labelled accordingly to the colour being detected.



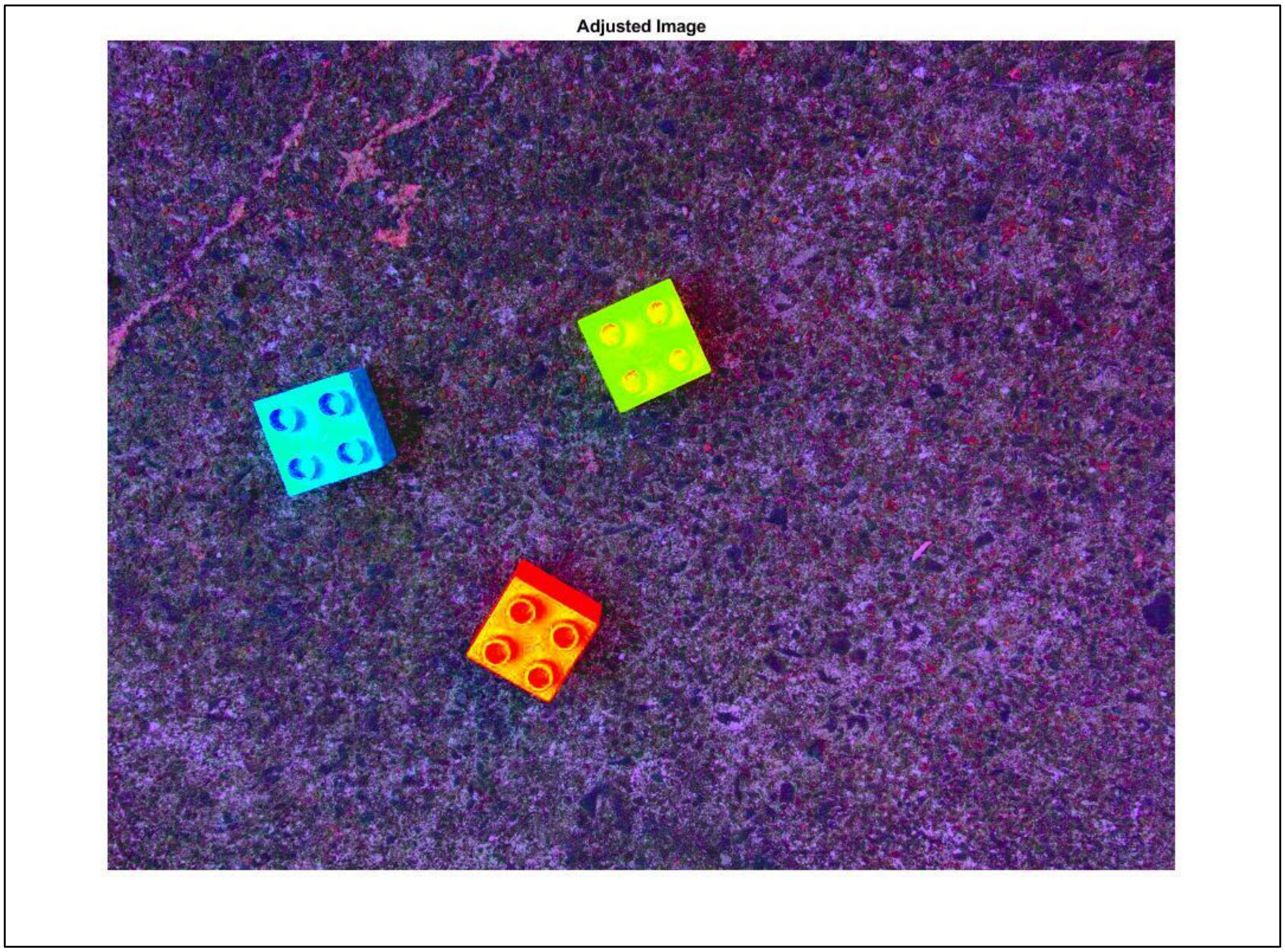
2.1.4 Labelling Colour

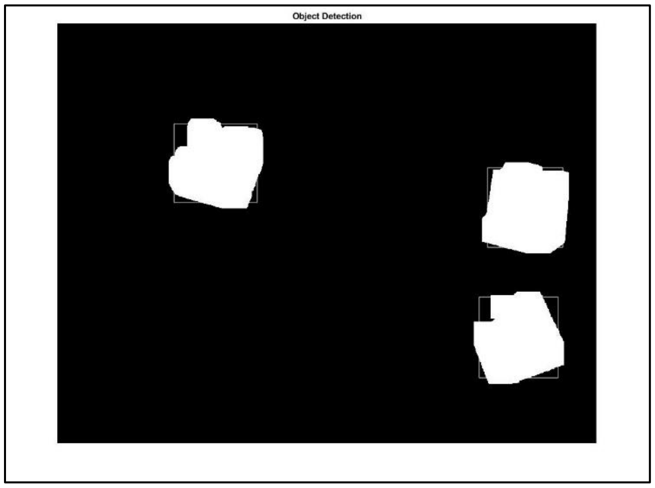
Once each of the objects have been detected the dilation and filling then make the brick more prominent and therefore, can be labelled with the overlay being shown over the original image. However, through the design currently employed in the program there is limitations in accuracy and therefore, there is still noise with the colours as each new image poses great contrast especially with outdoor settings, indoor settings, dark settings, changes in backgrounds, etc.

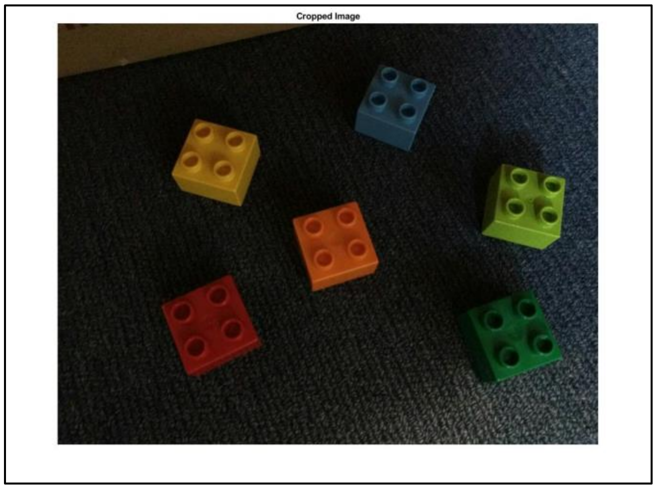


Section 3 Results & Discussion

There are issues with the stretching and adjustment of brightness's and limitations of the image as some of the outputs can results in the background also being amplified with its colours, therefore, there requires to be more tuning and possibly eliminating some of the functions used to only be applied in a RGB colour space.

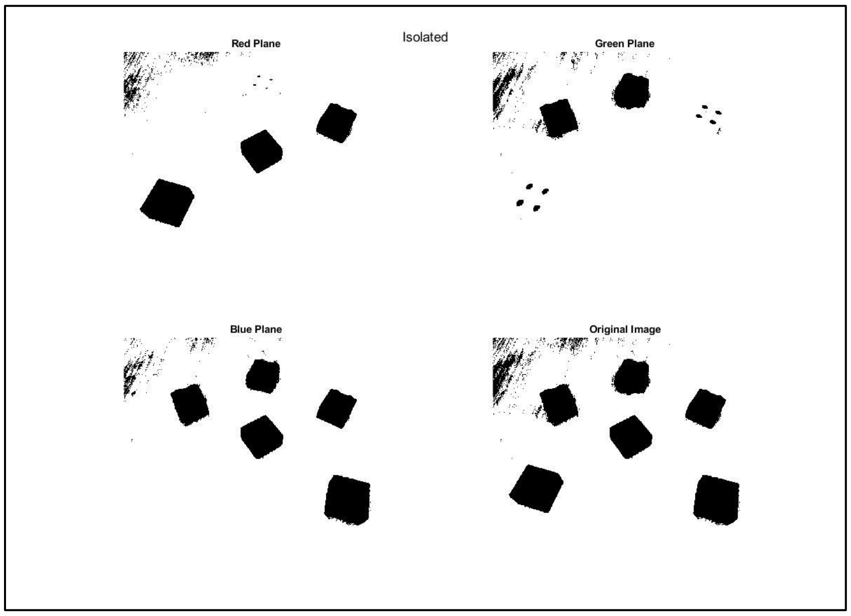






We can more clearly see the detection system not properly detecting all figures as not lighting has changed however this algorithm utilizes great eliminate of the background.

Within an outdoor setting or images, the lighting therefore, has an inverse effect on detection as the background becomes amplified. Therefore, the algorithm must be re-evaluated for situations as such since the background would be illuminated or binarized in the opposite shade.



With the joined image sets we see many different colours being sources of interference and the algorithm in place is not sufficient for such accurate isolation and identification of the bricks.

Object Detection

In summation, most difficulties and inconsistencies in results are due to the changes of lighting, background and other colours or objects being introduced. Attempts to supress interferences using dilation, fill, stretching of limitations, brightness and increasing contrast threshold has provided minimal improvements. Therefore, this would require image segmentation possibly to deal with situations of lighting and background changes or interference. For the additional images, it would be recommended that new functionalities be added that very specifically look for congested pixels of similar colour in a small area or region, allowing for the differentiation from other smaller different objects to the structure of the LEGO bricks themselves.

4.1 References

[1] Otsu N, “A threshold selection method from gray-level histograms”, IEEE transactions on systems, man, and cybernetics, Vol.9, No.1, (1979), pp.62-66

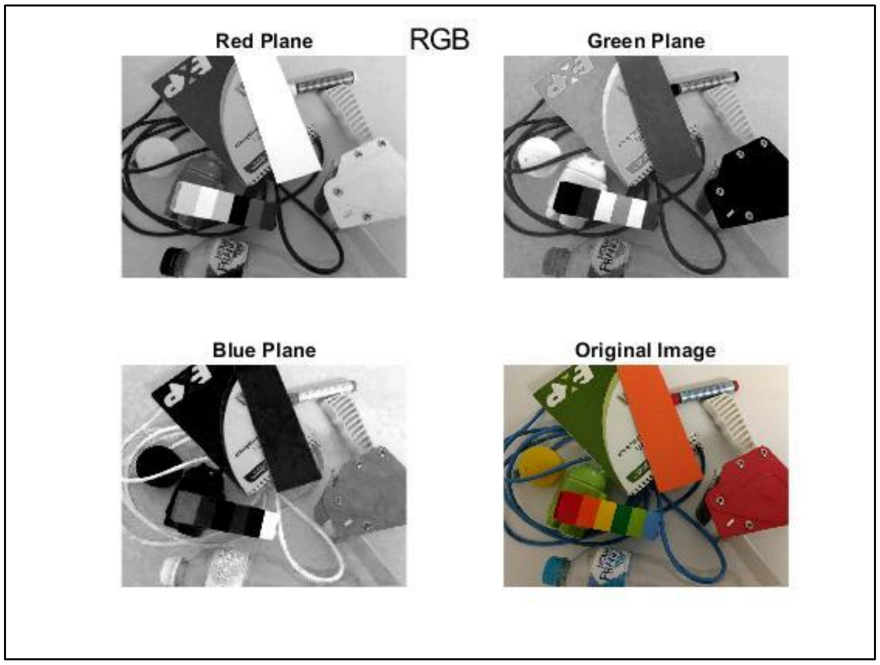
[2] Gonzalez RC, Woods RE & Eddins S.L, Digital Image Processing Using MATLAB, Pearson Education, Inc, (2004).

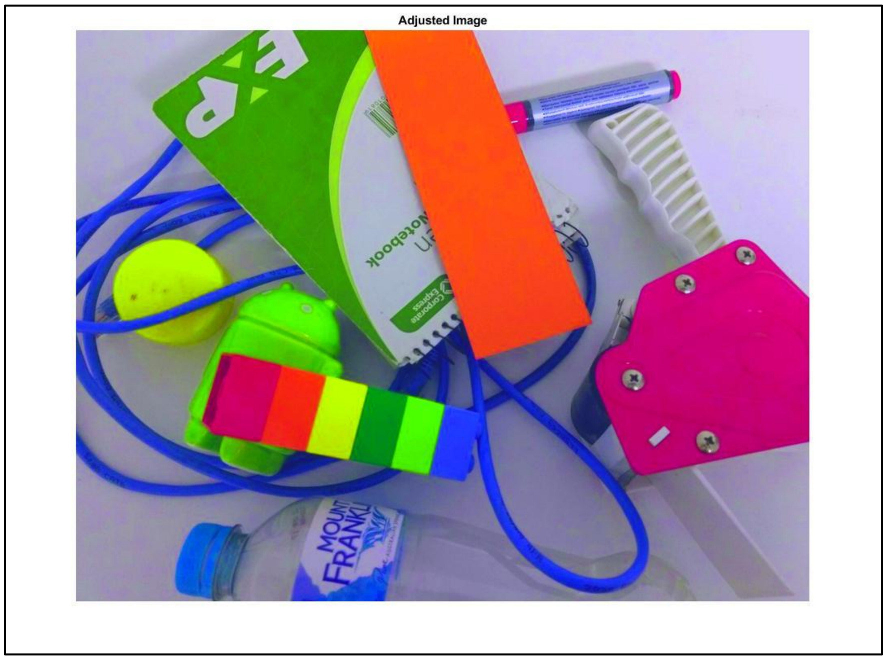
[3] Wyszecki G & Stiles WS, Color science, New York: Wiley, (1982).

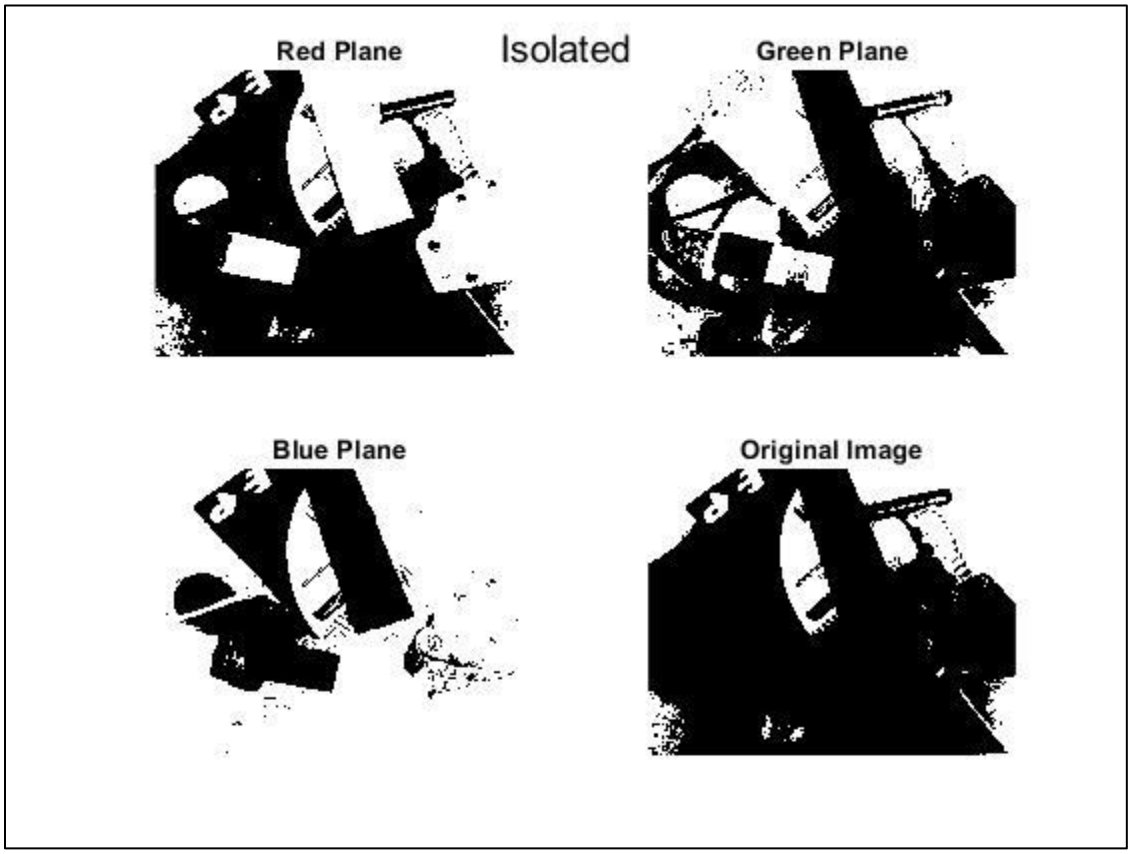
[4] Abadpour A, Color Image Processing Using Principal, (2005).

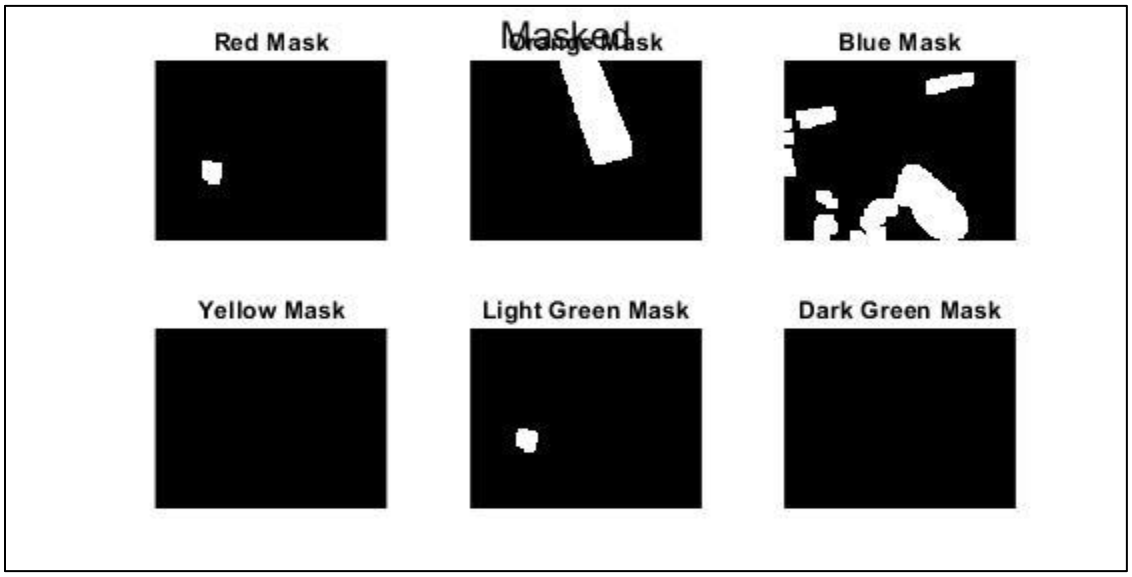
[5] Senthamaraikannan D, Shriram S & William J, “Real time color recognition”, International Journal of Innovative Research In Electrical, Electronics, Instrumentation And Control Engineering, Vol.2, No.3,(2014).

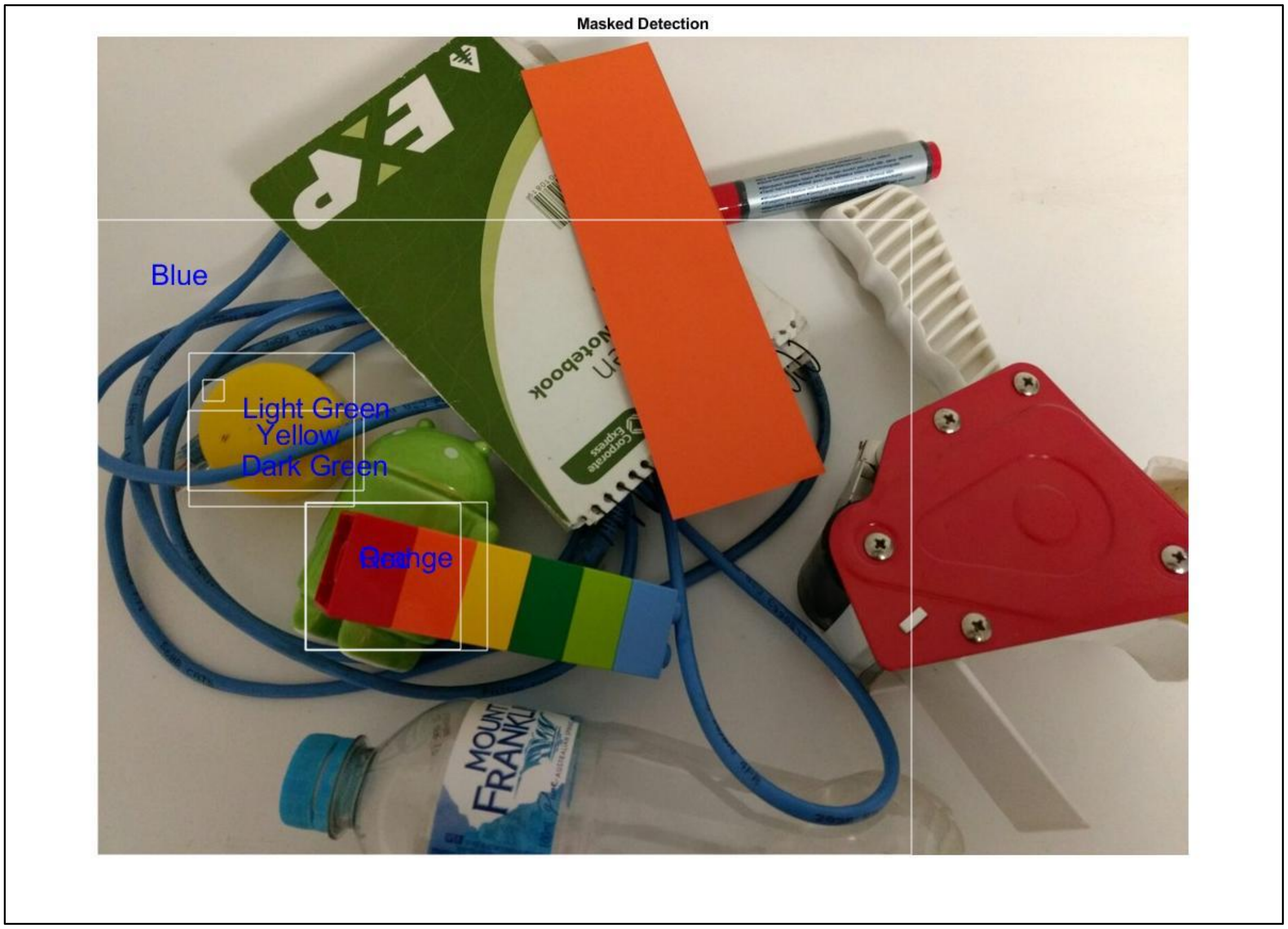
5.1 Appendix

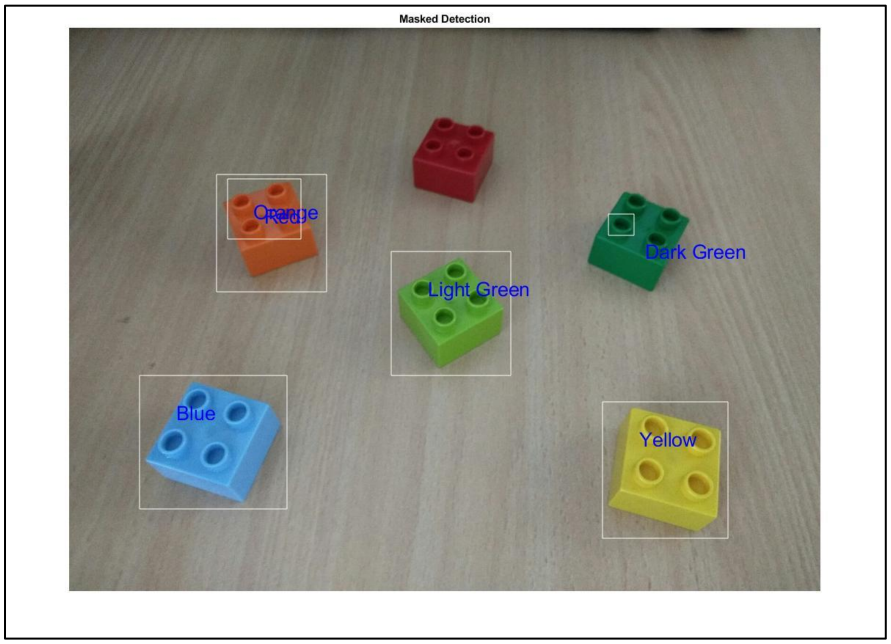

RGB
Red Plane
Green Plane
Blue Plane
Original Image

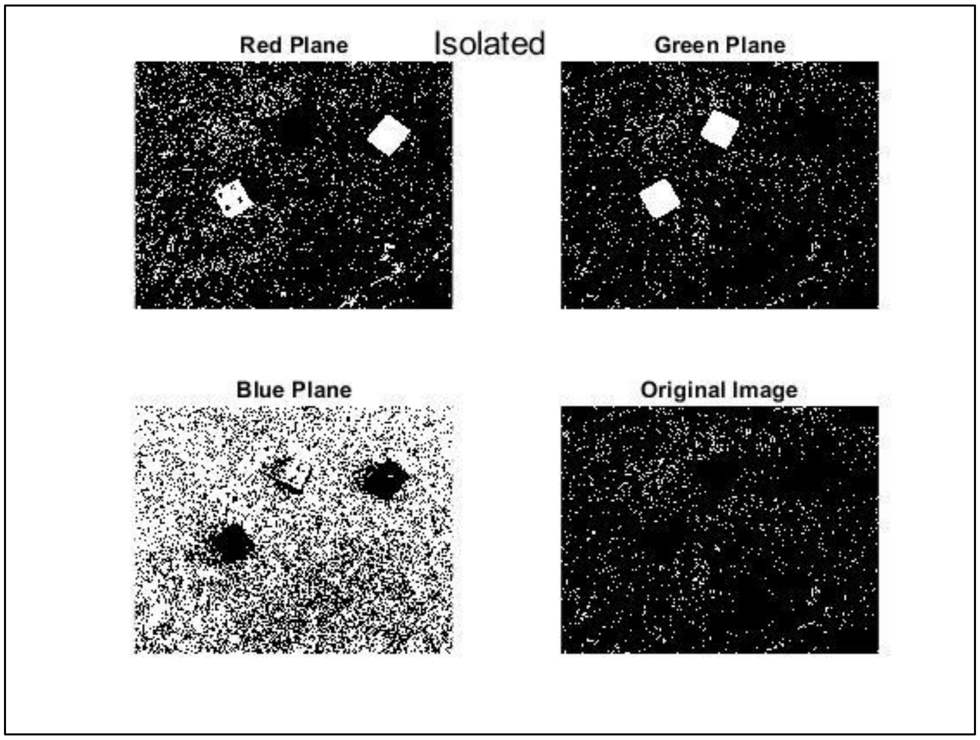

Adjusted Image

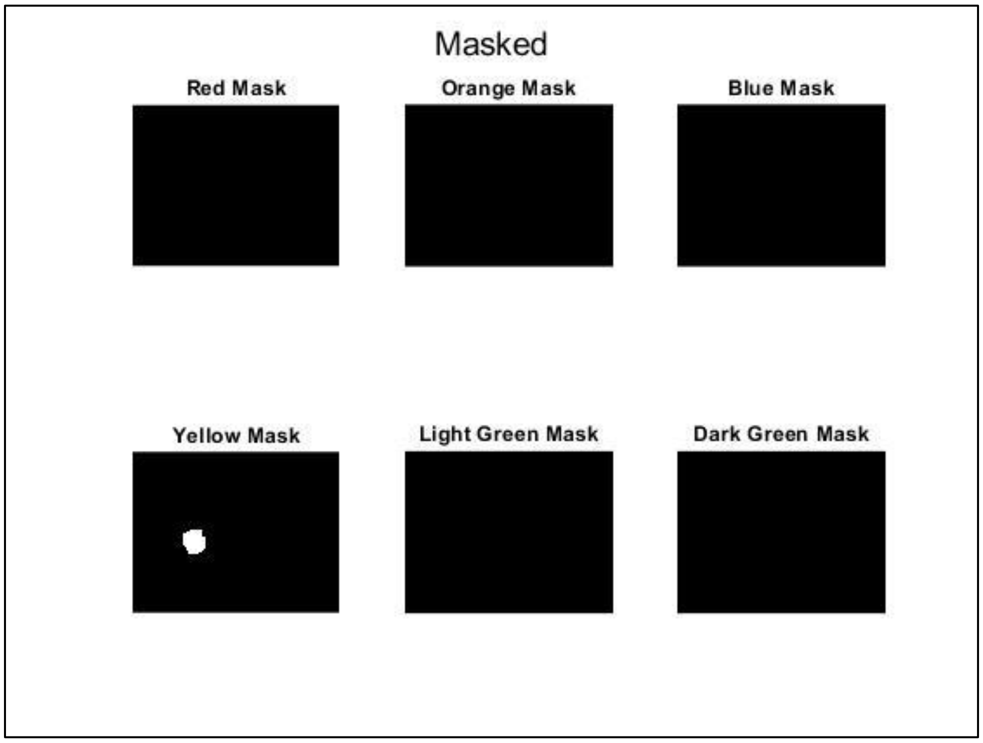

Isolated
Red Plane
Green Plane
Blue Plane
Original Image

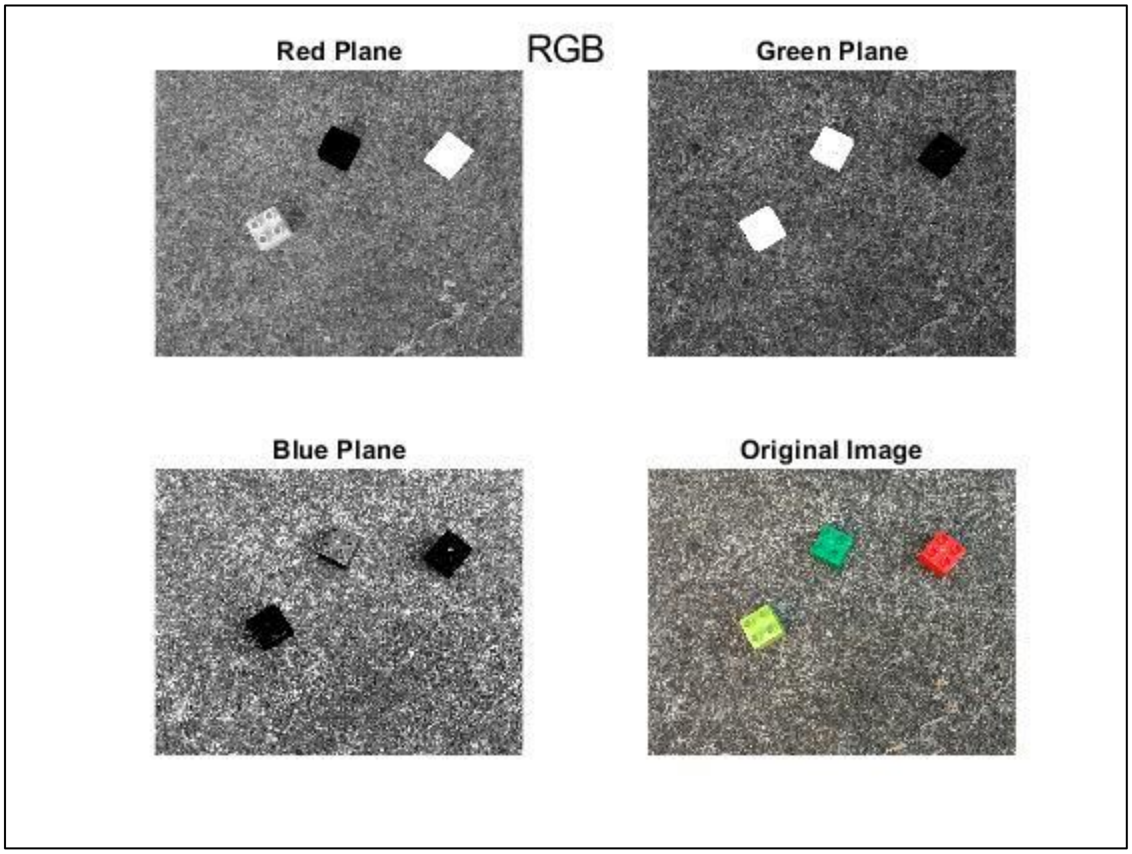

Masked
Red Mask
Orange Mask
Blue Mask
Yellow Mask
Light Green Mask
Dark Green Mask


Masked Detection
Blue
Light Green
Yellow
Dark Green
Orange
Red


Masked Detection
Orange
Red
Dark Green
Light Green
Blue
Yellow


Isolated
Red Plane
Green Plane
Blue Plane
Original Image


Masked
Red Mask
Orange Mask
Blue Mask
Yellow Mask
Light Green Mask
Dark Green Mask


Red Plane
RGB
Green Plane
Blue Plane
Original Image

6.1 Code

6.1.1 Question 1 – Complete Design

```
clear;
clc;
clear all;

% set flags for each colour we desire to be detected
flag_b = 0;
flag_o = 0;
flag_y = 0;
flag_lg = 0;
flag_dg = 0;
flag_r = 0;

% setting limiations to mimimum required pixels to try to eliminate some
% noise
count_b = 0;
count_o = 0;
count_y = 0;
count_lg = 0;
count_dg = 0;
count_r = 0;

% load image
im = imread('legobricks002.jpg');

% adjustments made to the image through the stretching of the RGB and
% greyscale space
im2 = imadjust(im,stretchlim(im));
im3 = decorrstretch(im2, 'Tol', 0.01);

figure(1);
imshow(im3);
title('Adjusted Image');

% setup labelling structure
s = {'Blue','Orange','Yellow','Light Green','Dark Green','Red'};

% seperate the image into its colour regions
[h,w,N] = size(im3);
red = im3(:,:,1);
blue = im3(:,:,2);
green = im3(:,:,3);

% run each of the colour thresholding functions to isolate each of the lego
% bricks
[BW_b,masked_b] = createMaskBlue(im3);
[BW_o,masked_o] = createMaskOrange(im3);
[BW_y,masked_y] = createMaskYellow(im3);
[BW_r,masked_r] = createMaskRed(im3);
[BW_dg,masked_dg] = createMaskDarkGreen(im3);
[BW_lg,masked_lg] = createMaskLightGreen(im3);
```

```
% function for detecting each of the colours in the image and which colours
% are present
for c = 1:w
    for r = 1:h

        if BW_b(r,c) > 0
            count_b = count_b + 1;
            if count_b > 100

                flag_b = 1;
            end
        end

        if BW_o(r,c) > 0
            count_o = count_o + 1;
            if count_o > 250

                flag_o = 1;
            end
        end

        if BW_y(r,c) > 0
            count_y = count_y + 1;
            if count_y > 250

                flag_y = 1;
            end
        end

        if BW_r(r,c) > 0
            count_r = count_r + 1;
            if count_r > 250

                flag_r = 1;
            end
        end

        if BW_dg(r,c) > 0
            count_dg = count_dg + 1;
            if count_dg > 250

                flag_dg = 1;
            end
        end

        if BW_lg(r,c) > 0
            count_lg = count_lg + 1;
            if count_lg > 250

                flag_lg = 1;
            end
        end

    end
```

```
end

% morphological computation
se = strel('cube',20);

% further optimisation of the binary image with dilation and fill for each
% of the masked and detected colours
outb = imdilate(BW_b,se);
outB = imfill(outb,'holes');
out_B = imopen(outB,se);
blobs_b = regionprops(out_B,'BoundingBox');

outo = imdilate(BW_o,se);
outO = imfill(outo,'holes');
out_O = imopen(outO,se);
blobs_o = regionprops(out_O,'BoundingBox');

outy = imdilate(BW_y,se);
outY = imfill(outy,'holes');
out_Y = imopen(outY,se);
blobs_y = regionprops(out_Y,'BoundingBox');

outr = imdilate(BW_r,se);
outR = imfill(outr,'holes');
out_R = imopen(outR,se);
blobs_r = regionprops(out_R,'BoundingBox');

outdg = imdilate(BW_dg,se);
outDG = imfill(outdg,'holes');
out_DG = imopen(outDG,se);
blobs_dg = regionprops(out_DG,'BoundingBox');

outlg = imdilate(BW_lg,se);
outLG = imfill(outlg,'holes');
out_LG = imfill(outLG,'holes');
blobs_lg = regionprops(out_LG,'BoundingBox');

% detect the centroid point x,y for each of the bounding boxes created for
% each of the lego bricks
if flag_b > 0

    points = blobs_b.BoundingBox;
    xCentroids_b = points(1:2:end);
    yCentroids_b = points(2:2:end);
end

if flag_o > 0

    points = blobs_o.BoundingBox;
    xCentroids_o = points(1:2:end);
    yCentroids_o = points(2:2:end);
end

if flag_y > 0
```

```
    points = blobs_y.BoundingBox;
    xCentroids_y = points(1:2:end);
    yCentroids_y = points(2:2:end);
end

if flag_r > 0

    points = blobs_r.BoundingBox;
    xCentroids_r = points(1:2:end);
    yCentroids_r = points(2:2:end);
end

if flag_dg > 0

    points = blobs_dg.BoundingBox;
    xCentroids_dg = points(1:2:end);
    yCentroids_dg = points(2:2:end);
end

if flag_lg > 0

    points = blobs_lg.BoundingBox;
    xCentroids_lg = points(1:2:end);
    yCentroids_lg = points(2:2:end);
end

figure(4);
imshow(im3);
title('Colour Detection');

figure(5);
imshow(im);
title('Masked Detection');

hold on;

% plot the box and the text onto the bricks themselves
if flag_b > 0

    rectangle('Position',blobs_b(1).BoundingBox,'Edgecolor','w');
    text(xCentroids_b(1)+50,yCentroids_b(1)+50,s(1),'Color','b','FontSize',20);
end

if flag_o > 0

    rectangle('Position',blobs_o(1).BoundingBox,'Edgecolor','w');
    text(xCentroids_o(1)+50,yCentroids_o(1)+50,s(2),'Color','b','FontSize',20);
end

if flag_y > 0

    rectangle('Position',blobs_y(1).BoundingBox,'Edgecolor','w');
    text(xCentroids_y(1)+50,yCentroids_y(1)+50,s(3),'Color','b','FontSize',20);
```

```
end

if flag_r > 0

    rectangle('Position',blobs_r(1).BoundingBox,'Edgecolor','w');
    text(xCentroids_r(1)+50,yCentroids_r(1)+50,s(6),'Color','b','FontSize',20);
end

if flag_dg > 0

    rectangle('Position',blobs_dg(1).BoundingBox,'Edgecolor','w');
    text(xCentroids_dg(1)+50,yCentroids_dg(1)+50,s(5),'Color','b','FontSize',20);
end

if flag_lg > 0

    rectangle('Position',blobs_lg(1).BoundingBox,'Edgecolor','w');
    text(xCentroids_lg(1)+50,yCentroids_lg(1)+50,s(4),'Color','b','FontSize',20);
end

hold off;
```