Name: Iyer Krithika Ganesh Sundaram

UNID: u1135255

Email ID: u1135255@utah.edu

Project 2

A. Image colorization

1. A histogram equalizing transformation routine

As described in the requirement, a program for calculating the histogram equalisation transform has been implemented. The input to the function:

- Input gray image.

For the given binary image, a simple histogram is calculated first. Then the probability distribution of the pixels is calculated by dividing the frequency of occurrence (from histogram) of each pixel with the total number of pixels in the image. Next the cumulative distribution function of the pixels is calculated. This cumulative distribution function is the histogram transformation which when applied to the original image will give us the equalised image.

Example 1:

For the given input image, the Normalised histogram is array is returned.

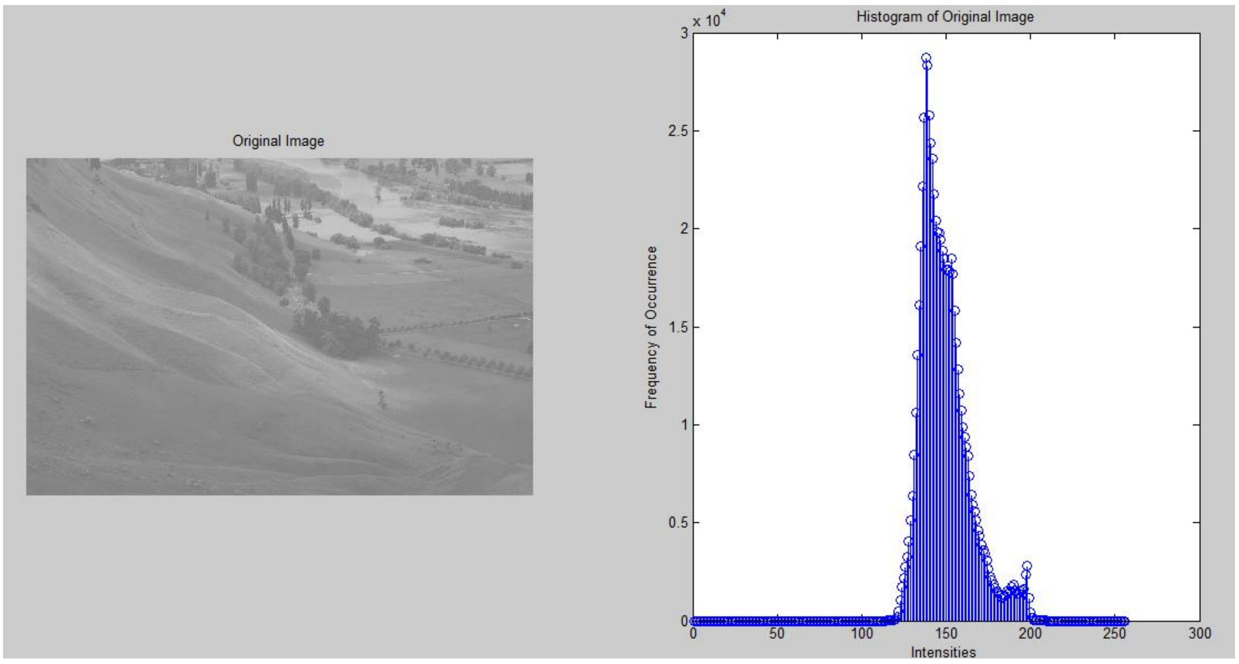


Figure 1

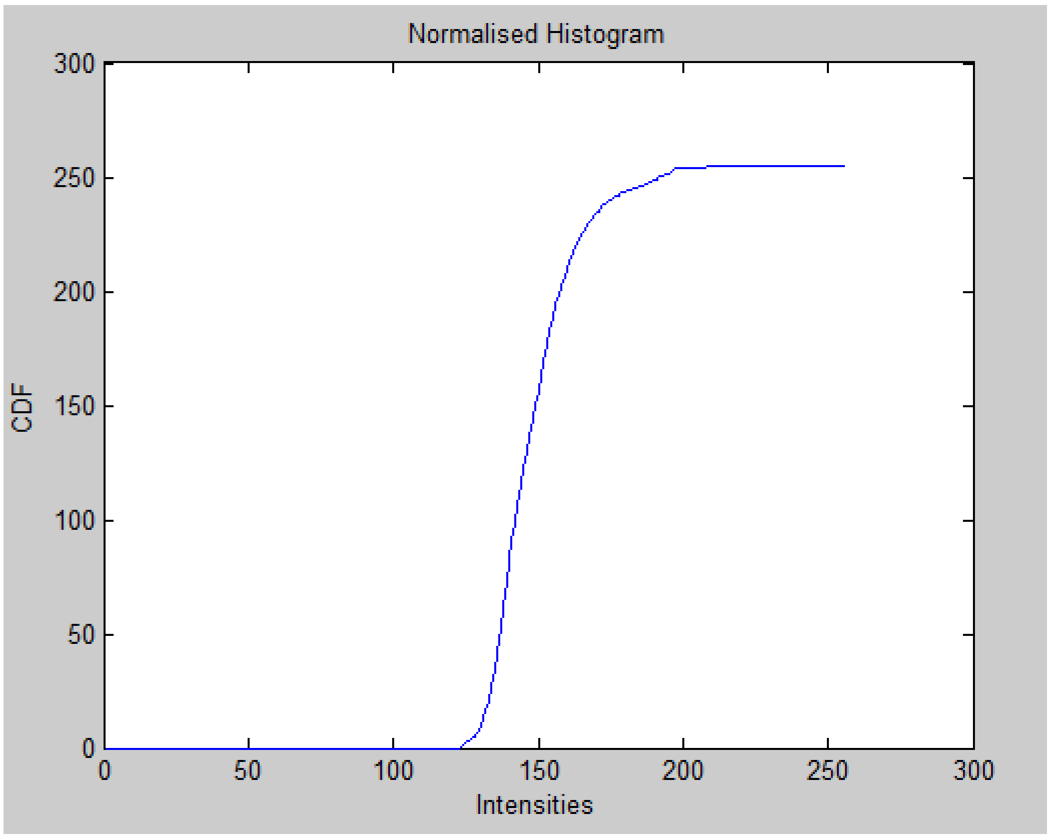


Figure 2

Example 2:

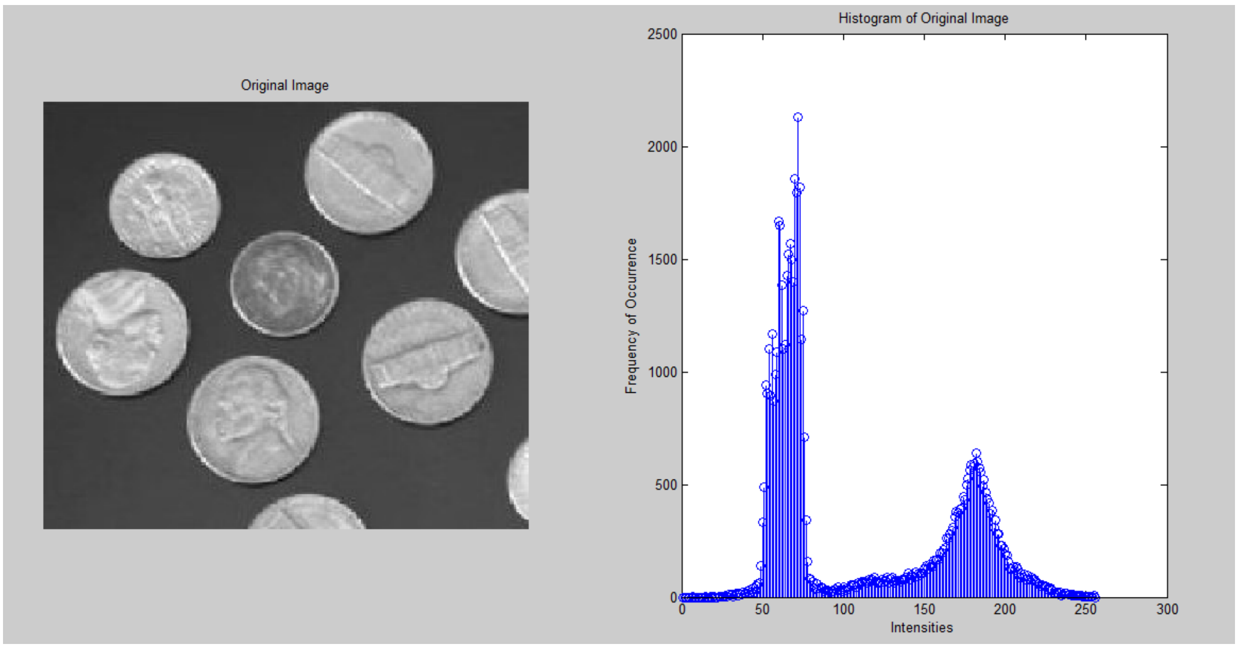


Figure 1

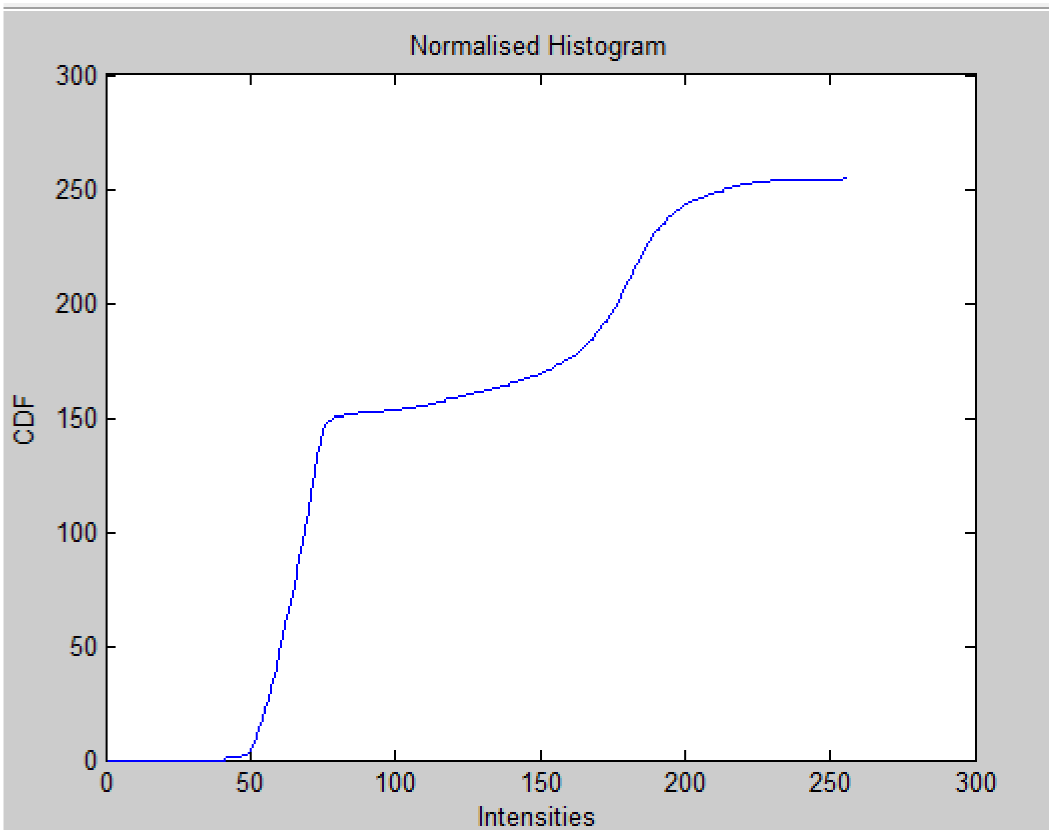


Figure 2

CODE: myhisteq.m

2. An intensity transformation routine

As described in the requirements, a program to perform intensity transformation has been implemented. For the transformation, the value of the transformation array at the index equal to the input image intensity is taken and applied to the output image.

The input to the function:

- Input gray image
- Mapping array

outputImage(r,c)=cdf(inputImage(r,c)+1);

if cdf(3) =5 then

if for some value say (r,c)=(100,123) , inputImage(100,123)=2 then

outputImage(100,123) =cdf(inputImage(100,123)+1) ---→ Her we are adding one because our indexing starts from 1 and not from 0

= cdf(2+1)

= cdf(3)

= 5 -→ Equalized value

With the intensity transformation, we are equalizing the histogram of the image. This means we are trying to adjust the contrast of the image by making sure the probability distribution of all pixels is as close to uniform as possible.

Example 1:

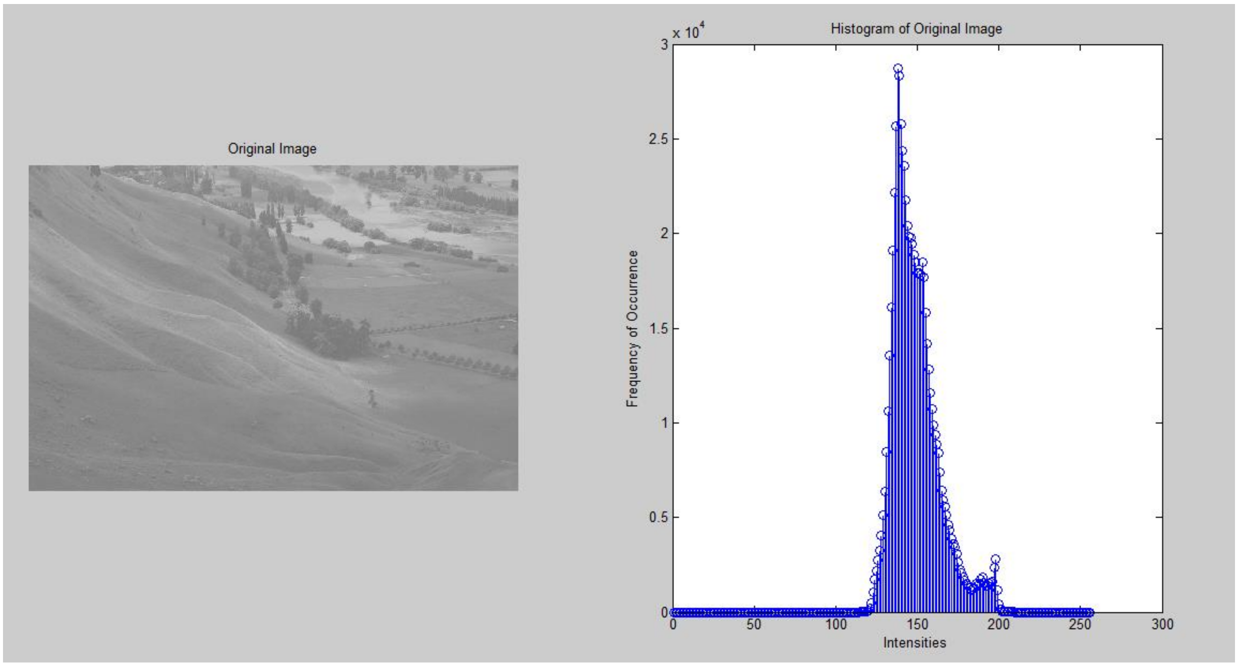


Figure 1

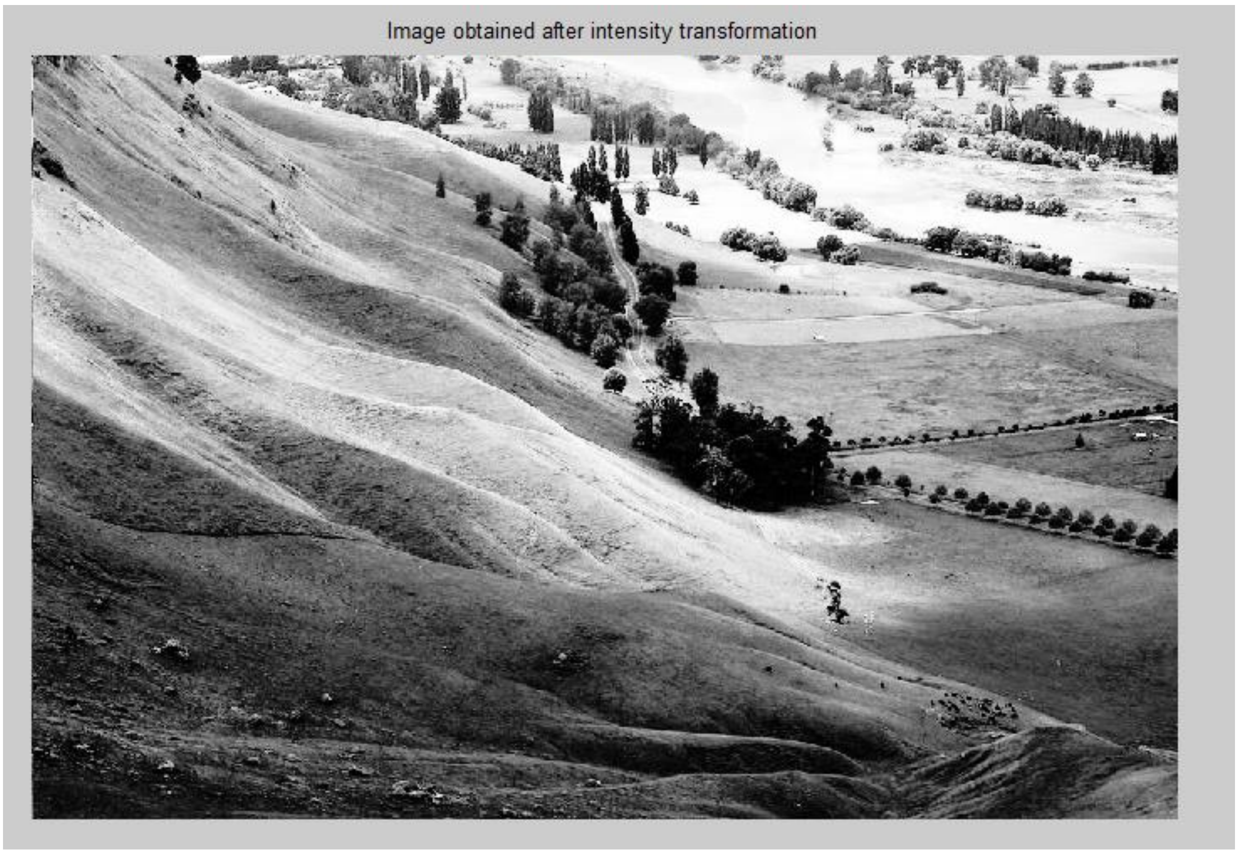


Figure 2

As we can see after applying the histogram equalization transform the image has become clear, as the intensities are spread out after equalization.

Example 2:

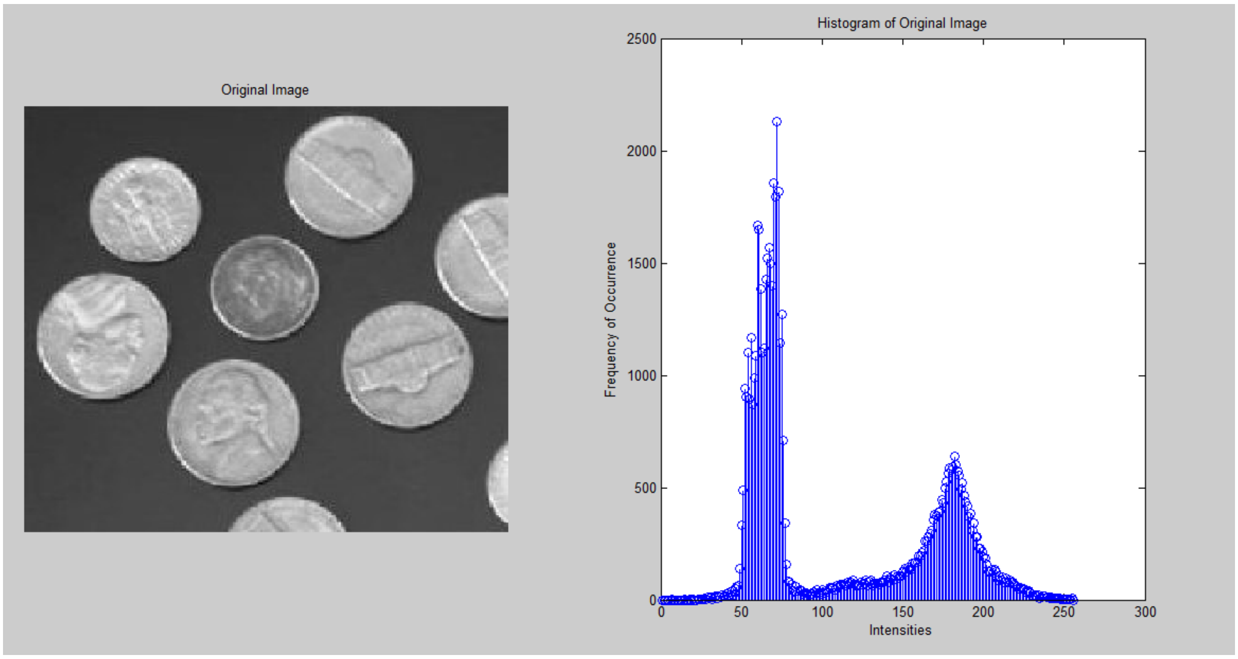


Figure 1

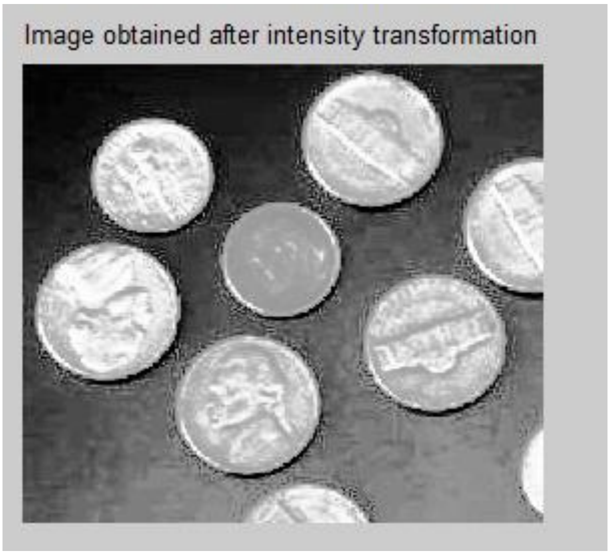


Figure 2

Example 3:

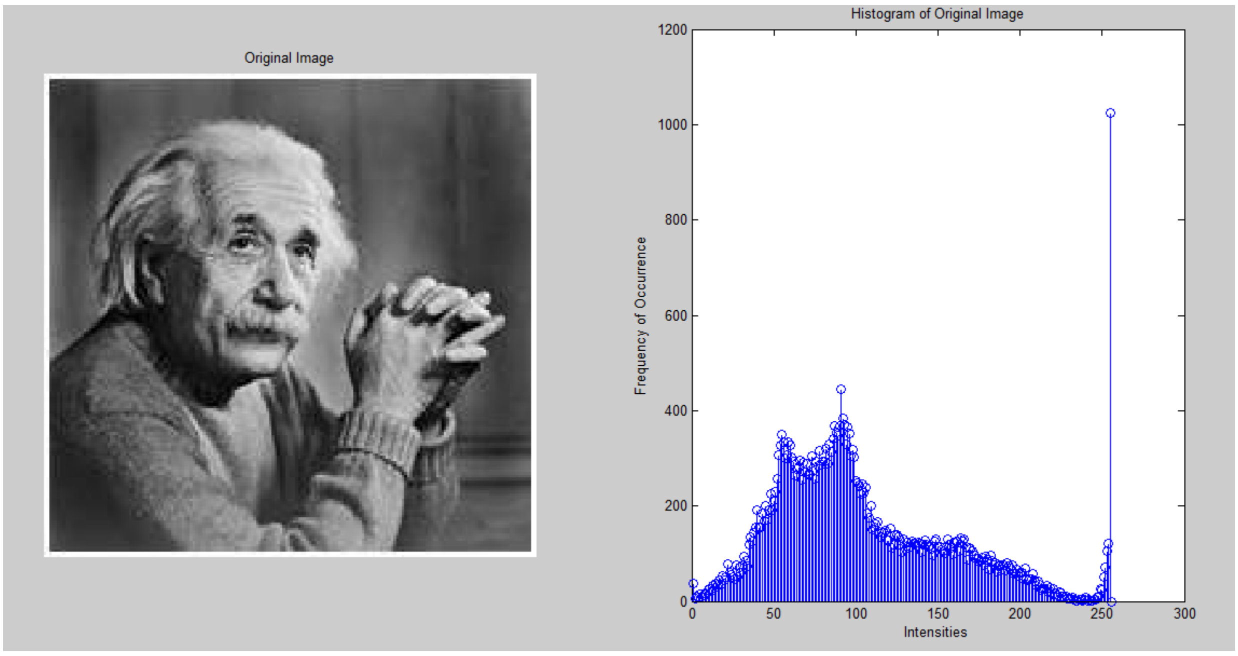


Figure 1

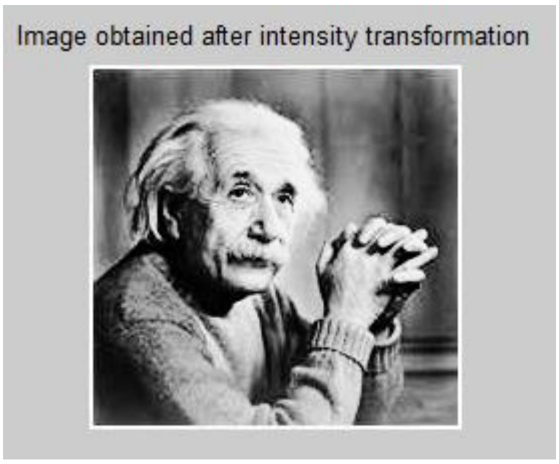


Figure 2

Example 4:

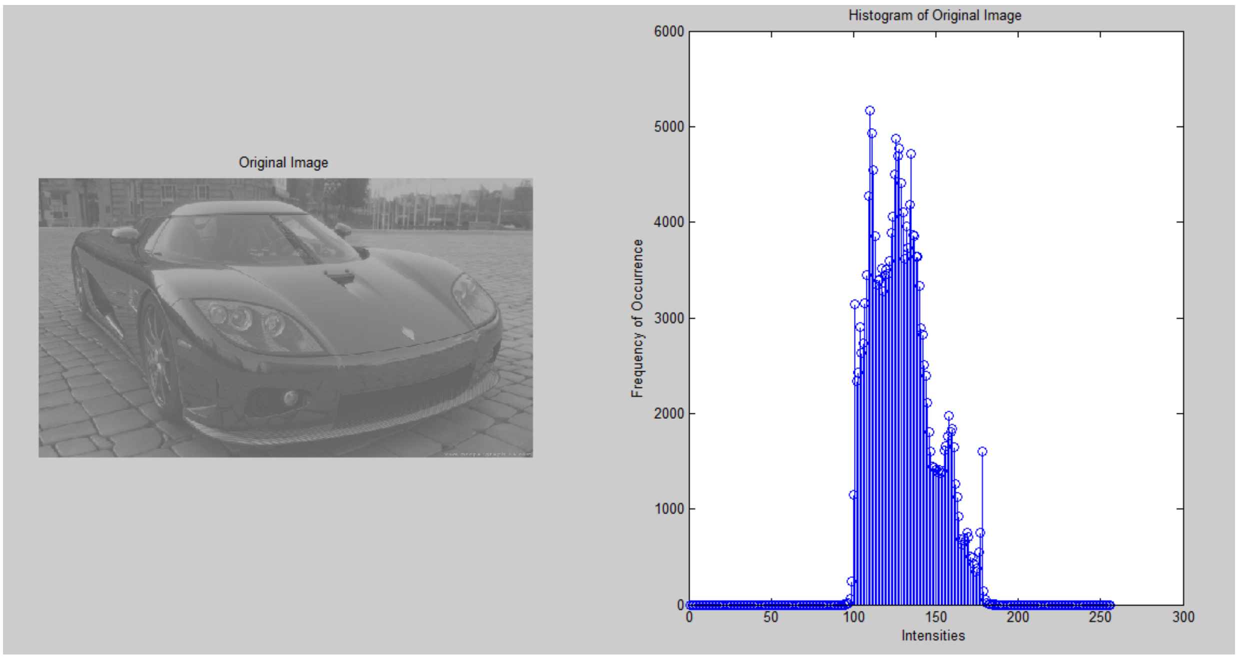


Figure 1

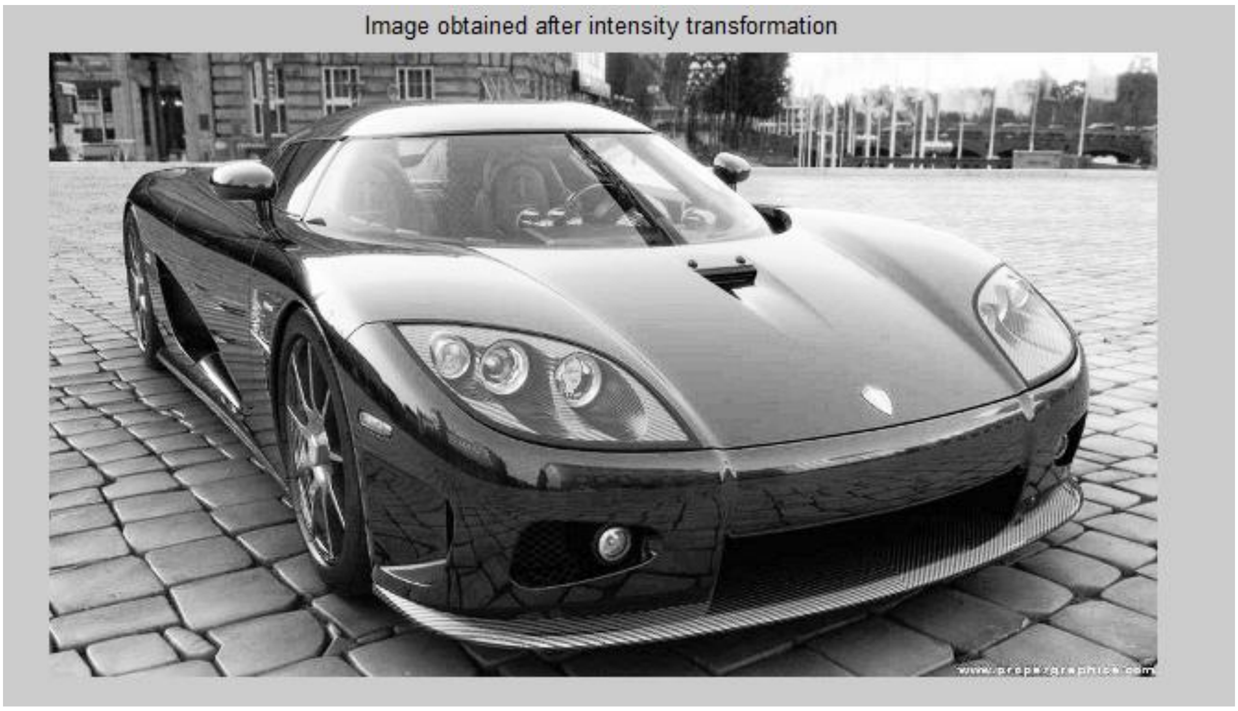


Figure 2

CODE: iTransform.m

3. An inverse histogram equalization transformation routine

For the inverse histogram equalization, the CDF (cdf1) of the image is calculated as mentioned in the description of the first function. Next, the CDF obtained is used to find the equalized image using intensity transformation. This results in equalized image. The CDF(cdf2) of equalized image is found. Now we map cdf2 to cdf1 and obtain the inverse transformation array. As a parameter of check, this inverse mapping is applied on the equalized image to check if the original image is returned.

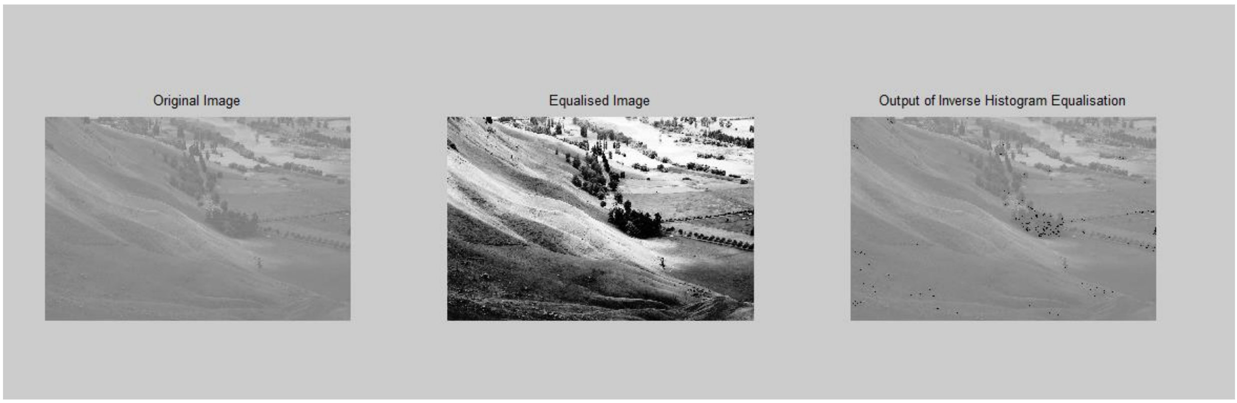
Algorithm:

i. Histogram calculation
ii. Calculate pixel probability distribution
iii. Calculate CDF-cdf1
iv. Apply the intensity transformation of cdf1 on input image
v. Calculate cdf2 for equalized image
vi. Mapping cdf2 and cdf1 by checking the difference between each value of cdf2 with all values of cdf1
vii. The index at which value of cdf1 gives the least difference is mapped with the corresponding cfd2 index

Example 1:

```
Mapping array is 0  123  124  125  126  127  127  128  128  129  129  129  129  130  130  130  131
131  131  131  131  132  132  132  132  132  132  133  133  133  133  133  133  133  134  134  134
134  134  134  134  134  135  135  135  135  135  135  135  135  135  135  136  136  136  136  136
136  136  136  136  136  137  137  137  137  137  137  137  137  137  137  138  138  138  138  138
138  138  138  138  138  139  139  139  139  139  139  139  139  139  140  140  140  140  140  140
140  140  141  141  141  141  141  141  141  141  142  142  142  142  142  142  142  142  143  143
143  143  143  143  143  144  144  144  144  144  144  144  145  145  145  145  145  145  145  146
146  146  146  146  146  146  147  147  147  147  147  147  147  148  148  148  148  148  148  149
149  149  149  149  149  149  150  150  150  150  150  150  151  151  151  151  151  151  151  152
```

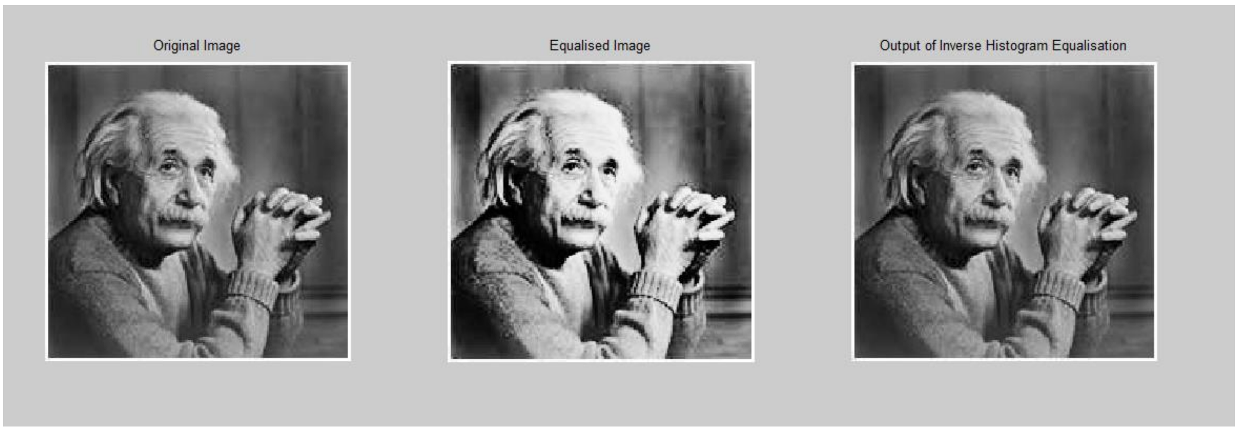
```
152  152  152  152  152  152  153  153  153  153  153  154  154  154  154  154  154  155  155  155
155  156  156  156  156  157  157  157  157  158  158  158  158  159  159  159  160  160  160  160
161  161  161  162  162  163  163  163  164  164  165  165  166  166  167  168  168  169  170  170
171  172  173  174  175  176  178  180  182  184  186  188  189  191  193  195  196  197  208
```



As we can see the image obtained after applying the inverse histogram equalization is same as the input image, which is as expected. The histogram equalization process is not purely reversibly, this can be seen with the granular black dots. This is because, while mapping we might not be able to map the values to the exact level and hence we map it to the nearest value which introduces small error.

Example 2:

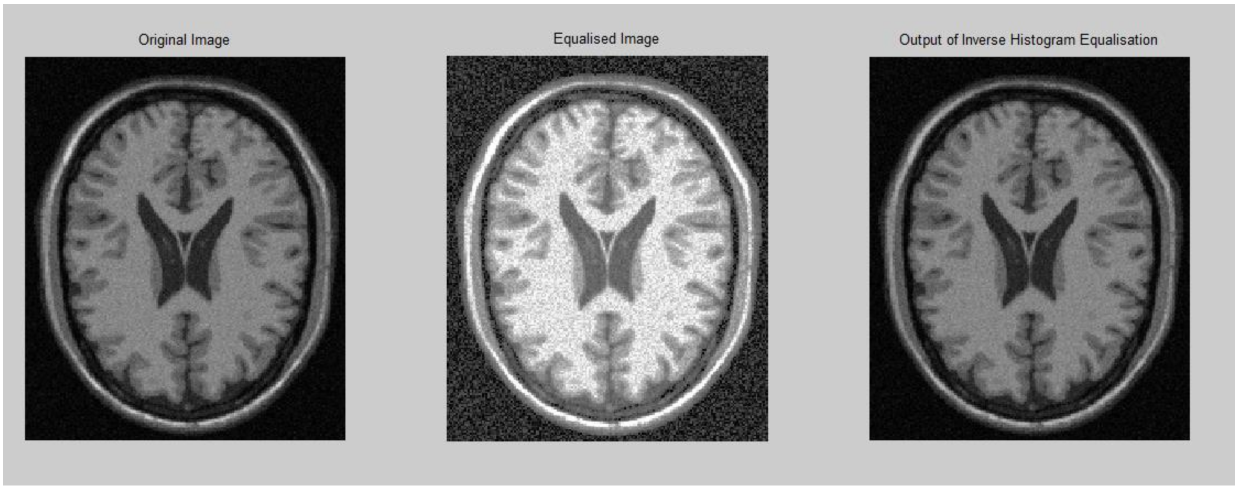
```
Mapping array is 0    9   14   17   20   22   24   27   29   31   32   34   35   36   37   38   38   39   40
41   41   42   43   44   44   45   46   46   47   48   48   49   49   50   50   51   51   52   52   53   53   53
54   54   55   55   55   56   56   57   57   57   58   58   59   59   60   60   61   61   62   62   62   63   63
64   64   65   65   66   66   67   67   68   68   69   69   70   70   71   71   71   72   72   73   73   74   74
75   75   76   76   77   77   78   78   78   79   79   80   80   81   81   82   82   82   83   83   84   84   84
85   85   85   86   86   87   87   87   88   88   88   89   89   89   90   90   90   91   91   91   92   92   93
93   93   94   94   94   95   95   96   96   97   97   97   98   99   99  100  100  101  101  102  102  103
104  104  105  105  106  107  108  108  109  110  111  112  113  114  115  116  116  117  118  119
120  121  123  123  124  125  126  128  129  130  131  132  133  134  135  137  138  139  140  141
142  143  144  146  147  148  149  150  151  153  154  155  156  157  159  160  161  162  163  164
165  166  168  169  170  171  173  174  176  177  179  181  182  184  186  187  189  191  193  195
197  199  202  204  207  210  215  220  239  250  252  253  253  253  253  253  253  253  253  254
```



Example 3:

```
Mapping array is 0    0    0    0    1    1    1    2    2    2    2    3    3    3    3    3    4    4    4    4    4    4
5    5    5    5    5    5    5    6    6    6    6    6    6    7    7    7    7    7    7    7    8    8    8    8    8    8    9
9    9    9    9    9   10   10   10   10   10   11   11   11   11   11   12   12   12   12   12   13   13   13
14   14   14   15   15   16   16   17   17   18   19   20   21   22   23   24   26   27   29   30   31   32   33
34   35   36   37   37   38   39   40   40   41   42   43   43   44   45   46   47   48   49   50   51   52   53
```

```
55  56  57  59  61  63  65  67  69  71  73  74  76  78  79  81  82  83  84  85  86  87  87
88  89  89  90  91  91  92  92  93  93  94  94  95  96  96  97  97  98  98  99  99 100
100 101 101 102 102 103 103 104 105 105 106 107 107 108 109 110 111 112 113 113
114 115 116 117 118 119 119 120 121 121 122 122 123 123 124 124 125 125 125 126
126 126 127 127 127 128 128 128 129 129 129 130 130 130 130 131 131 131 132 132
132 133 133 133 133 134 134 134 135 135 135 136 136 136 137 137 138 138 139 139
140 140 141 142 142 143 144 146 147 150 157 175 222
```



CODE: inversehistogram.m

4. Color matching

As described in the requirements, two images are taken as input. First image is the gray scale image which is to be colored and the second is the reference color image. The normalized histogram(cdf) of the gray scale image is mapped with normalized histogram of the RGB layers of the color image and 3 different mappings arrays are created. These three mapping are applied on the gray image and a new color image is obtained from them.

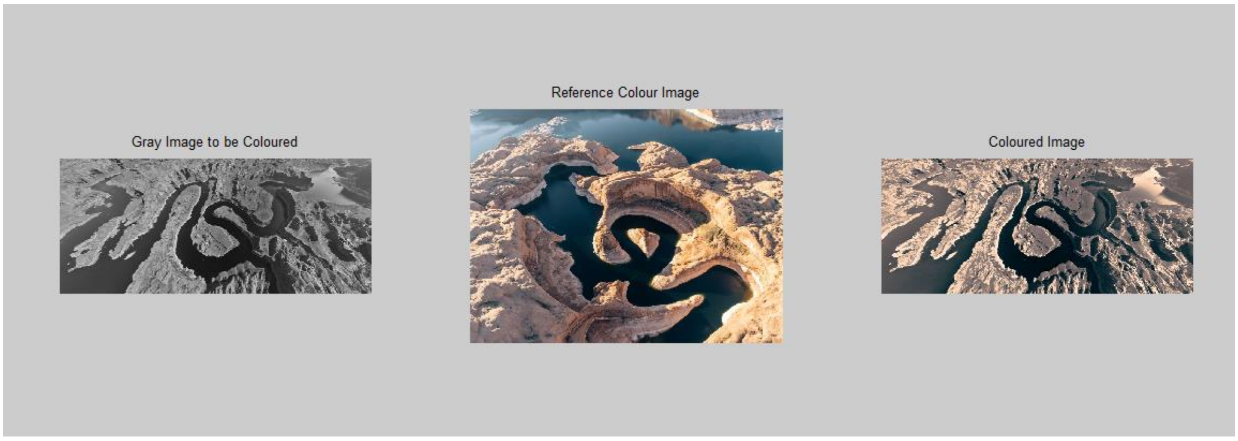
The function has following inputs:

- Input gray image
- Reference colour image

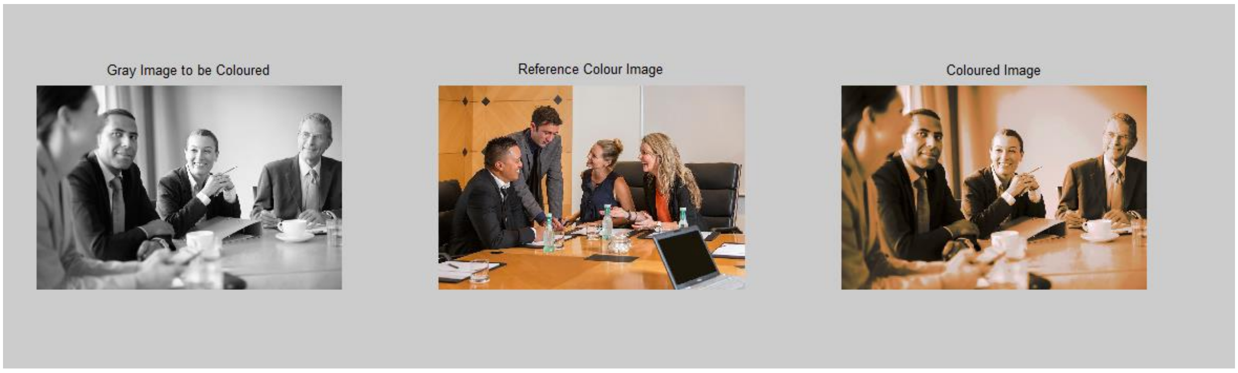
Algorithm:

i. Calculate histogram of input gray image – H1
ii. Calculate histogram of all three planes of color image – R, G,B
iii. Map H1 and R and find the intensity transformation array MapR
iv. Map H1 and G and find the intensity transformation array MapG
v. Map H1 and B and find the intensity transformation array MapB
vi. Apply MapR, MapG and MapB on the input image and obtain 3 new planes of the coloured image. For applying the mapping use the same steps as mention in intensity transformation function.
vii. Combine all three new transformed planes to obtain coloured image.

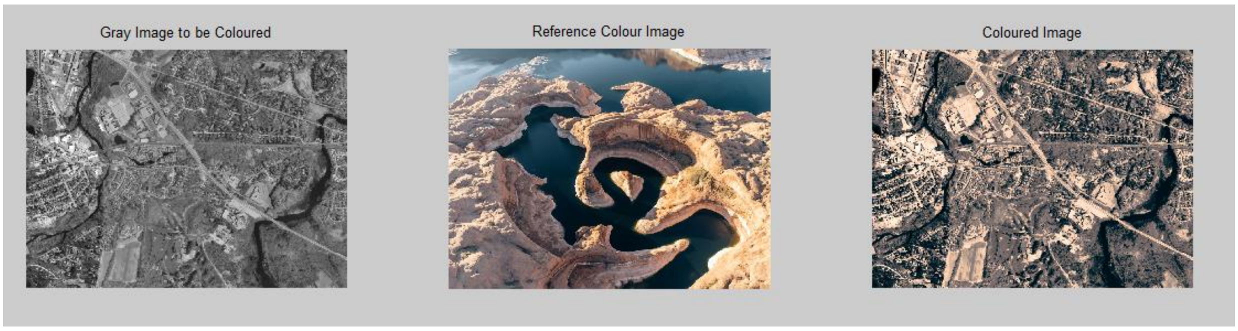
Example 1:



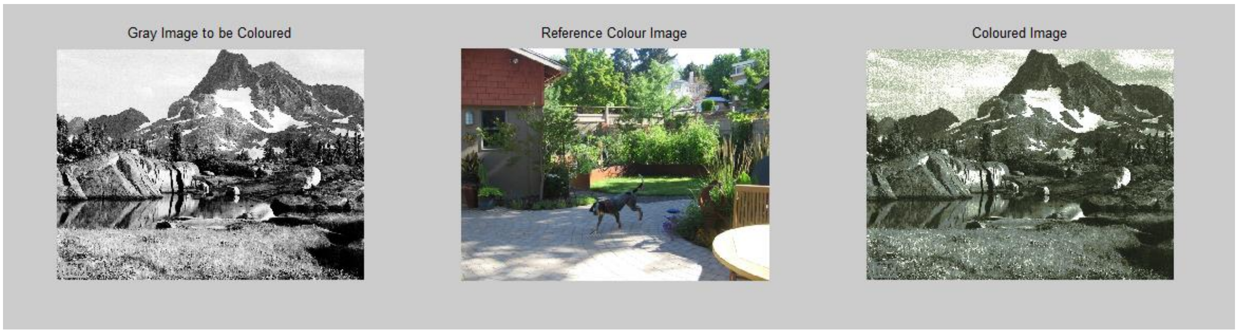
Example 2:



Example 3:



Example 4:



Here two images from different context are used.

CODE: colour.m

B. Counting cells

1. Otsu’s thresholding method

As per the requirements, a function to threshold the image using Otsu thresholding method has been implemented. Following the algorithm, the image is progressively divided into two groups and the probability distribution of these group of pixels P1 and P2 is calculated. The mean values of these two sets of pixels are calculated and iteratively it is checked to find which set of pixels give the maximum variance. The value of pixel at which maximum variance is calculated is taken as the threshold value and image is thresholded.

Example 1:

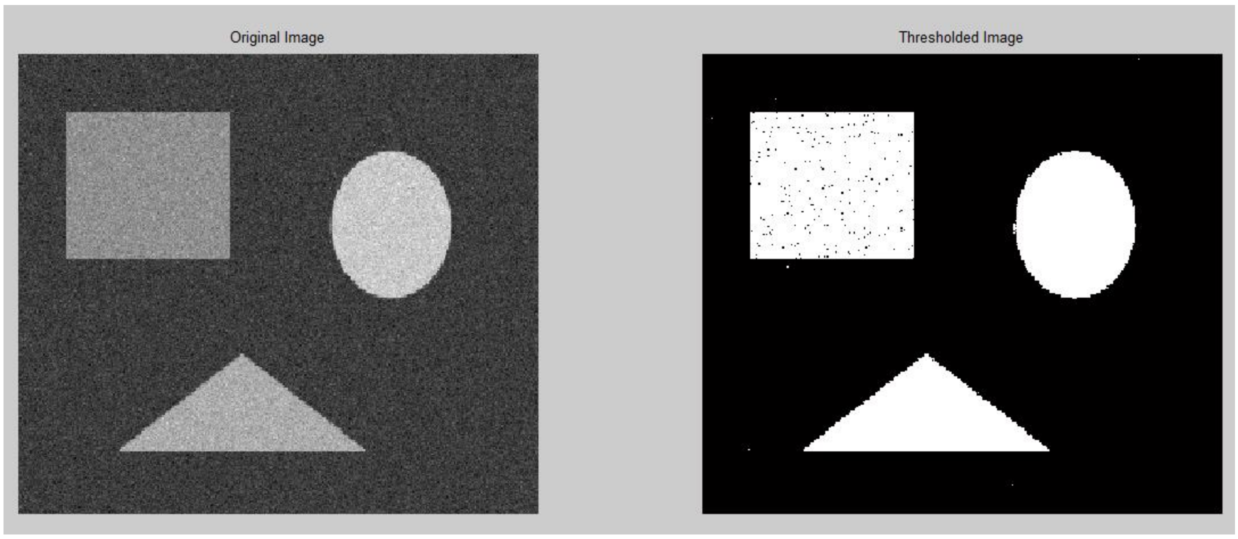


Figure 1

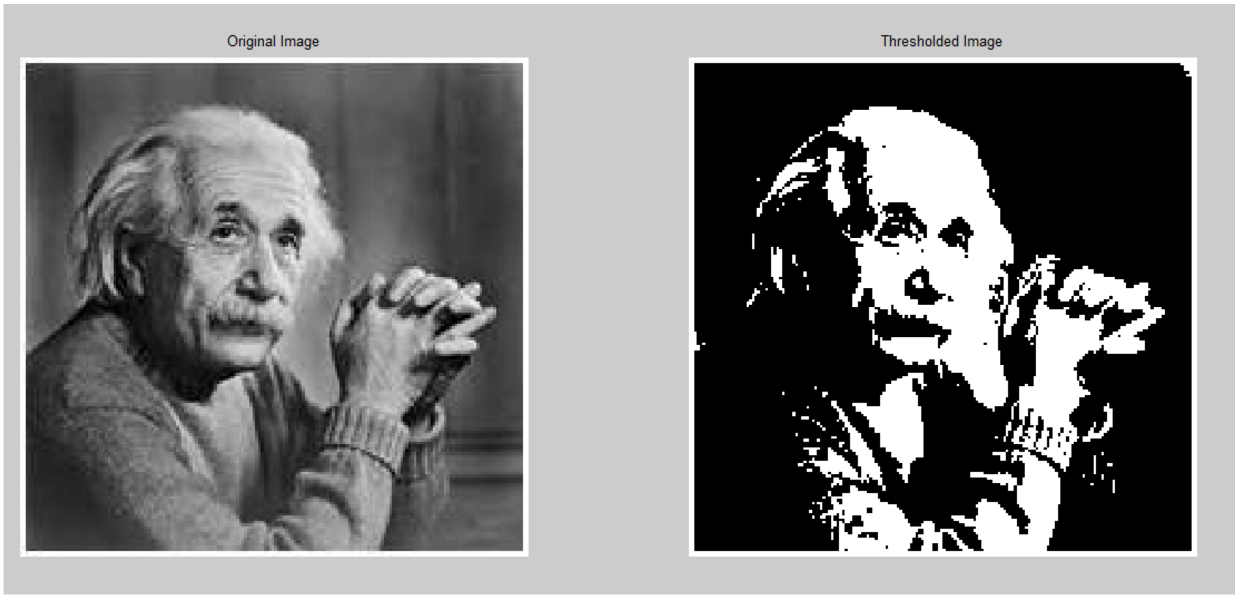


Figure 1

Example 3:

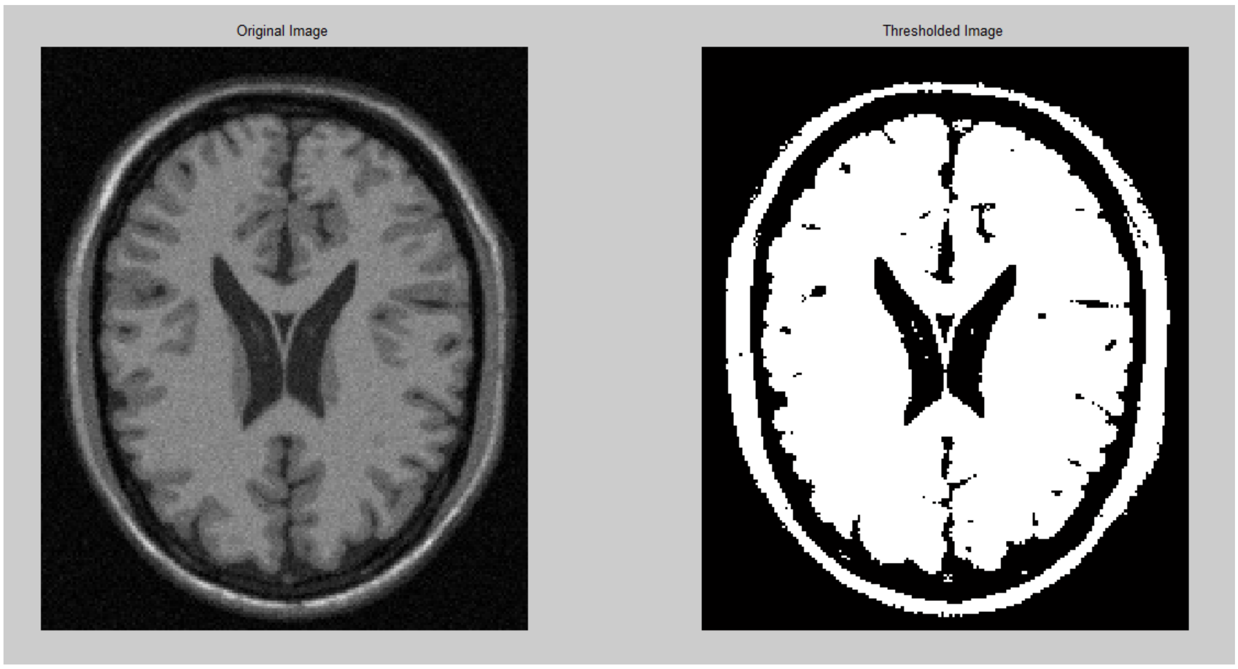


Figure 1

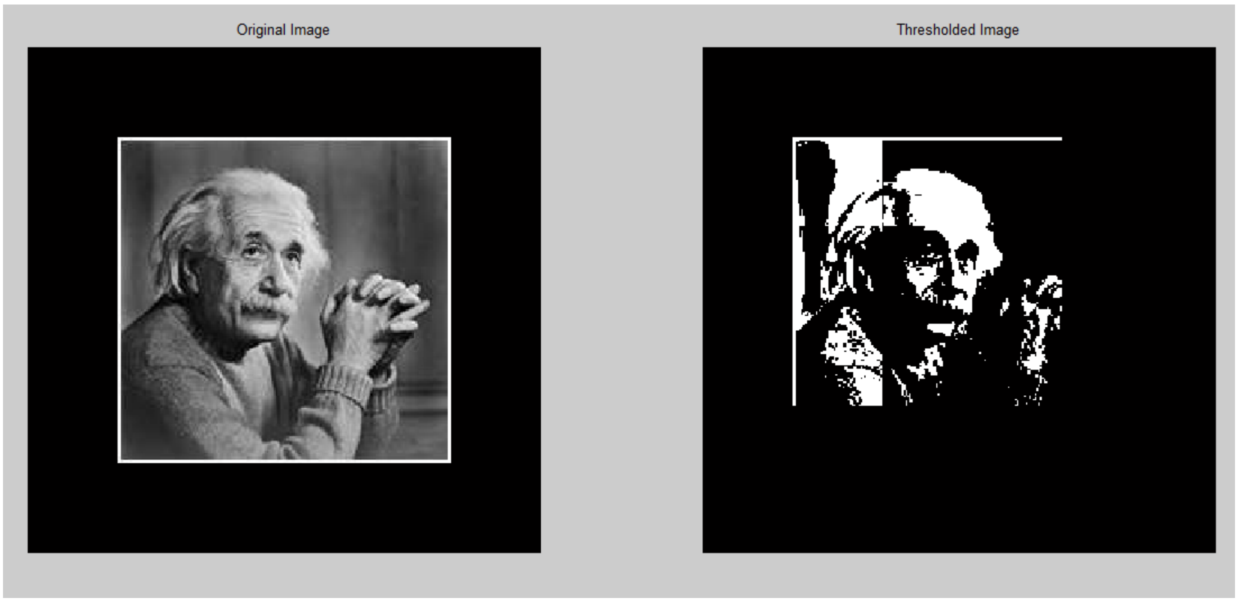
CODE: otsuThresh.m

2. Adaptive thresholding method

The image is divided into multiple blocks of size *bsize* and analysed with refernce to the input minimum variance, to find the threshold of each block. The mean of all the pixels in the block is calculated and then the variance of each block is calculated. The blocks which have variance greater than minimum variance are foreground pixels and hence are thresholded at a level mean plus one standard variance.

Example 1:

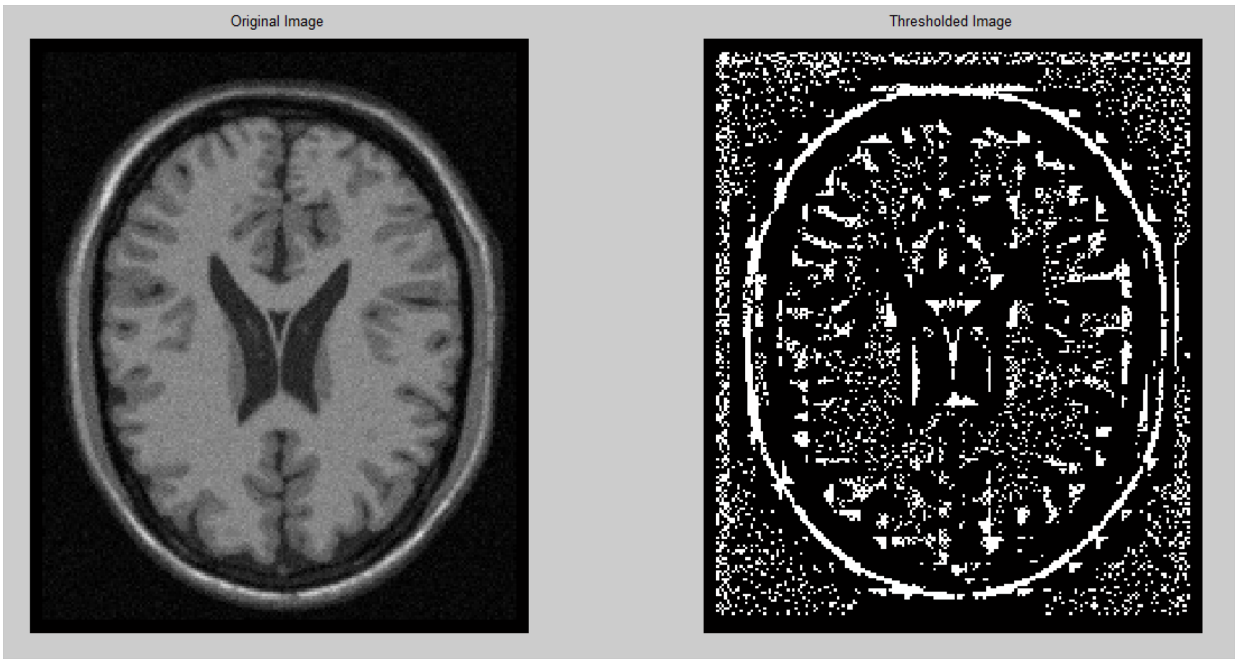
Block size: 100
Minimum variance value : 4.6



Zero padding has been done at the borders so that the image is equally divided in to the block size.

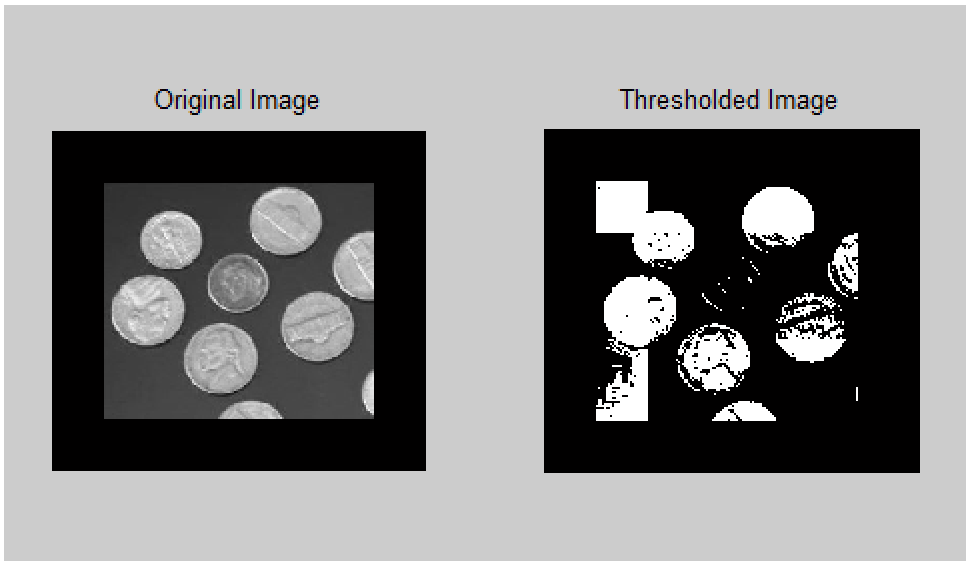
Example 2:

Block size: 10
Minimum variance value : 0.002



Example 3:

Block size: 100
Minimum variance value : 2



Some noise can be seen when we use adaptive thresholding , since it looks at the image at a block level to identify the threshold.

CODE: adaptive_Thresh.m

3. Cell Detection

In this part, Otsu’s thresholding and adaptive thresholding methods are used to identify the cells in the given image. The thresholding algorithms are followed by topological denoising to get an appropriate count.

CODE: cellDetectOtsu.m
Cell detection using Otsu’s thresholding method results:

```
>>Number of cells
```

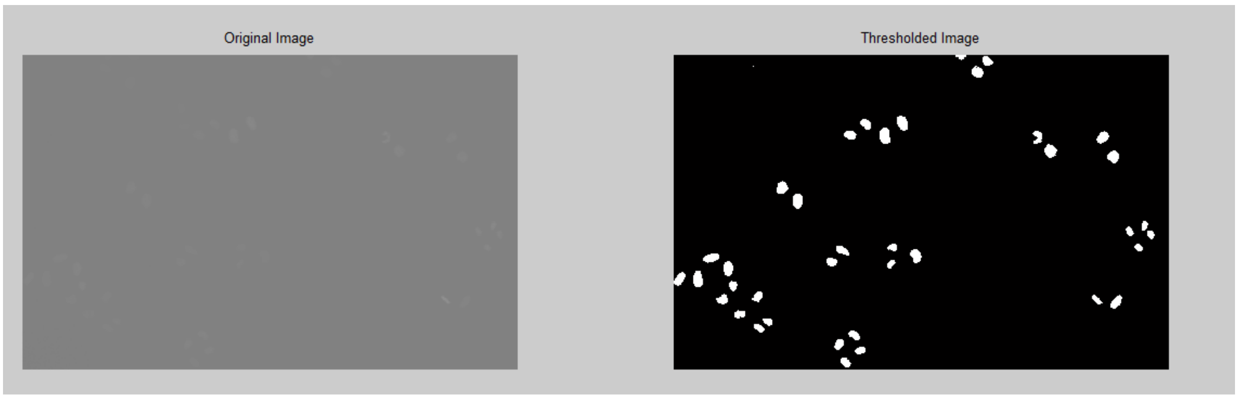


Figure 1

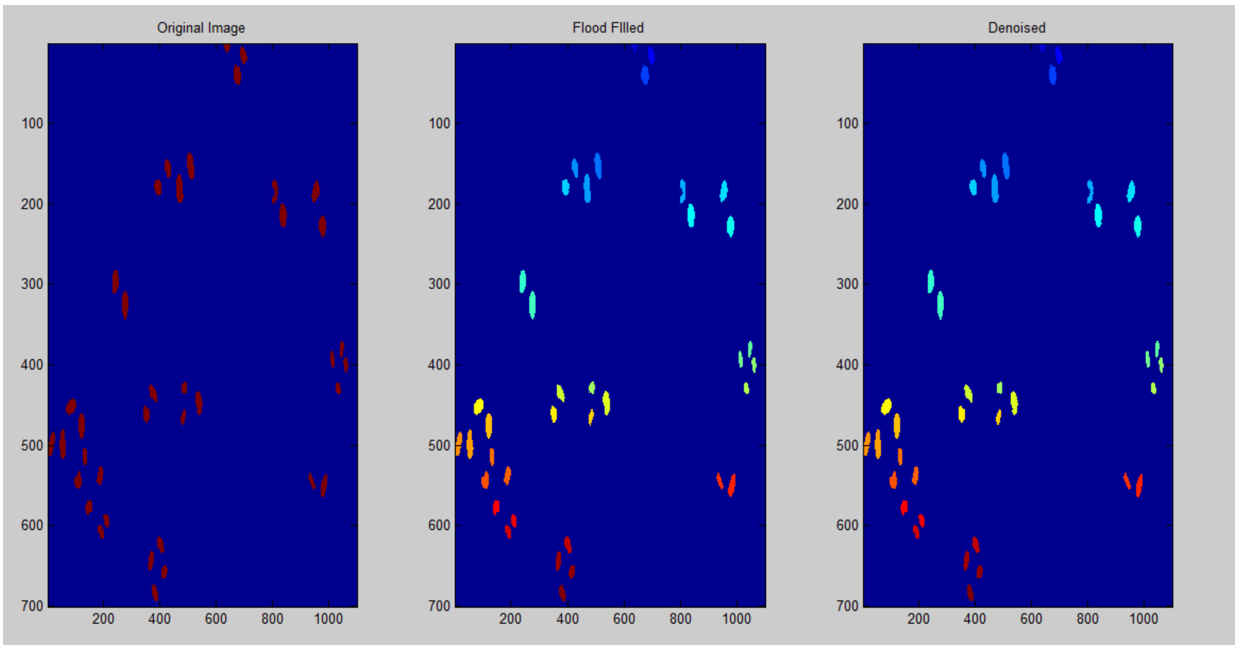


Figure 2

CODE: cellDetectAdap.m

Cell detection using adaptive thresholding method results:

Block size: 65
Minimum variance value: 0.563

```
>>Number of cells
   55
```

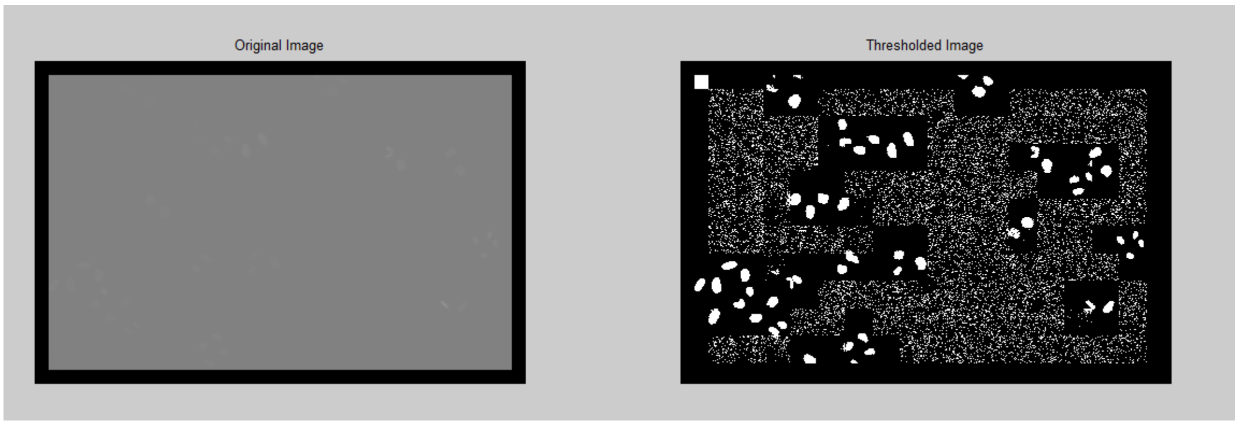


Figure 1

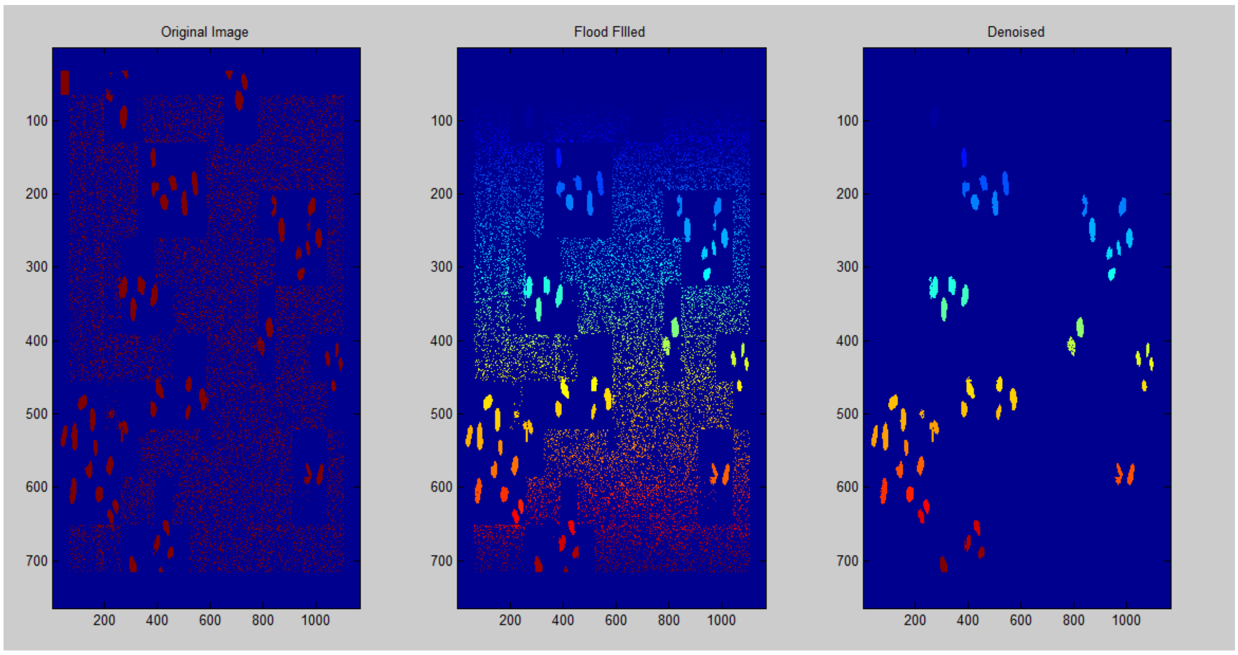


Figure 2

Analysis:

- As we can see using Otsu’s method, the global threshold is used and with adaptive thresholding, each block has a different threshold i.e. ., it is local threshold
- With adaptive threshold, using the local threshold a lot more details are uncovered as compared to global thresholding.
- Cell Count with adaptive thresholding is much accurate as compared to Otsu.
- Global threshold works well when the image does not have large variations in the intensity level.
- When adaptive thresholding is used,we can see a lot of noise. This is eliminated by using topological denoising algorithm.