

CS 4732

MACHINE VISION

PROJECT 3

Morphological Filtering

#### INSTRUCTOR

**Dr. Mahmut KARAKAYA**

**Michael Rizig**

**001008703**

**1. ABSTRACT**

In this project, we are given many different tasks to complete. To begin, we are given an image of the moon with a blur effect, and through the use of sharpening techniques such as Laplacian and Sobel filters, we are tasked with sharpening the image. The next step is to take an image of a fingerprint, generate the histogram for the image, and use the histogram to determine an ideal thresholding point, then threshold the image to convert it to binary. From there we use various techniques such as morphological dilation and erosion to remove holes from the image and improve the overall quality of the image. By utilizing various filter and structuring elements, we see different results, some better than others. Finally for our last task, we are given an image, and are asked to count the total number of cells, calculate the size of each cell, and show the boundary of the biggest cell in the image.

To view all edits, changes, and see step by step revision history, view this project on my GitHub:

<https://github.com/michaelrzg/CS4732-Projects>

**2. Test RESULTS**

**2.1 Sharpening**

**(Only a few selected images are used here to highlight the effect. All output images can be found in output>log folder in the zip submission)**

**To sharpen our moon image, we utilize both the Laplacian and the Sobel filters to give the edges more contrast and make them ‘sharper’. For Laplacian, we take the second derivative filter (Figure 1b) and apply it to the image as coefficients to the corresponding pixel neighborhood. This generates a Laplacian filtered image (Figure 1c) which we then subtract from our original image to get the output(Figure 1d). For Sobel, we make 2 passes, one with the horizontal Sobel filter (Figure 1e) and one with the vertical (Figure 1f). After the two passes we get a Sobel filtered image(Figure 1g) which we subtract from our original image to get our output image(Figure 1h).**

**Image 1a:** Original Image ‘moon.jpg. **This is the input image.**

**Image 1b**: The Laplacian filter coefficients.

**İmage 1c:** Laplacian filtered image **(positive values only).**

**Image 1d:** Laplacian filtered image **(positive and negative values).**

**İmage 1e**: Original image minus Laplacian Filtered Image (**Output of Laplacian Filtering**) .

**Image 1f**: Horizontal Sobel Filter coefficients.

**Image 1g:** Vertical Sobel Filter coefficients.

**Image 1h:** Sobel Filtered Image

**Image 1i:** Original Image minus Sobel Filtered Image **(Output of Sobel Filtering)**

**Figure 1:** (a) Original Image, (b) Laplacian Filter Coefficients, (c) Laplacian filtered image (positive only), (d) Laplacian filtered image (positive and negative values) (e) Original image minus Laplacian Filtered Image (**Output of Laplacian Filtering**), (f) Horizontal Sobel Filter coefficients, (g) Vertical Sobel Filter coefficients, (h) Sobel Filtered Image, (i) Original Image minus Sobel Filtered Image **(Output of Sobel Filtering)**

|  |  |  |
| --- | --- | --- |
| **(a)** | **(b)** | **(c)** |
| **(d)** | **(e)** | **(f)** |
| **(g)** | **(h)** | **(i)** |

**2.2 Fingerprint Morphological Processing.**

**(Only a few selected images are used here to highlight the effect. All output images can be found in output>log folder in the zip submission)**

**For the fingerprint image, the first step is to convert the original image *(2a)* into a binary image. To do this, we first must generate the histogram to determine a good thresholding point. Based on the images histogram distribution *(2b)* , we can see that there is a small spike around 75-80 and a large spike around 225. Based on this we can determine any value in between these two to be the threshold value, I choose 150 because it is a value far enough from both that we will not get any outlier values mixed up. Using this threshold we can generate the binary image *(2c)* of the fingerprints. The next step is to determine our size and shape of our structuring element. We begin with a 3x3 structuring element of a cross shape *(2d)*, as it seemed to work well in the slides. We start by dilating the image in order to close the holes found in the fingerprint. We do this by looking for 'hits’ where at least one pixel is hit by the structuring element. If we get a *hit*, we make the target pixel 255. The result is a ‘dilated’ image where elements become enlarged effectively filling the holes in the image *(2e).* Next we 3x3 erode the image, which is done by looking for ‘fit’ or where all pixels in structuring element are hit, then we make the target 255. This sequence, called “closing operation” effectively shrinks the elements back to their original size and preserves the previous effects of dilation to the imperfections, giving us output for 3x3 filtering *(2f).* We repeat this effect with a 5x5 structuring element (2g) for 5x5 dilation (2h) and 5x5 erosion (2i) and see that the larger structuring element causes bodies to combine, making the erosion effect less effective.**

**Image 2a:** We beg­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­in with the original fingerpint.jpg. **This is the input image.**

**Image 2b:** Histogram Distribution of Original Fingerprint Image.

**İmage 2c:** Binary Image generated after thresholding at 150.

**İmage 2d:** 3x3 structuring element.

**Image 2e:** Original image after 3x3 Dilation (many holes ‘closed’)

**Image 2f:** 3x3 Dilated image after 3x3 Erosion (**Output for 3x3 Morphological Processing**).

**Image 2g:** 5x5 Structuring Element

**Image 2h:** Original image after 5x5 Dilation.

**Image 2i:** 5x5 Dilated image after 5x5 Erosion.(**Output for 5x5 Morphological Processing)**

|  |  |  |
| --- | --- | --- |
| **(a)** | **(b)** | **(c)** |
| **(d)** | **(e)** | **(f)** |
| **(g)** | **(h)** | **(i)** |

**Figure 2: (a)** Original fingerprint.jpg image, (b) Histogram Distribution of Original Fingerprint Image, (c) Binary Image generated after thresholding at 150, (d) **:** 3x3 structuring element, (e) Original image after 3x3 Dilation, (f) 3x3 Dilated image after 3x3 Erosion (**Output for 3x3 Morphological Processing**), (g) 5x5 Structuring Element , (h) Original image after 5x5 Dilation. (i) 5x5 Dilated image after 5x5 Erosion. (**Output for 5x5 Morphological Processing)**

**2.3 Histogram Equalization (greyscale)**

**For the histogram equalization on the university.jpg image, we start by calculating the original image’s histogram and observing the trends. Figure 3b shows this histogram. We notice that the distribution of data is focused around x=5 with a range of (0,50). To scale this to (0,255), all we need to do is create a function to convert each pixel in the original range to the new range. This can simply be done by converting the original range to a scale from 0 to 1, then multiplying that result by the range we want (255-0=255). So for each pixel we divide its grey-level by 50, then multiply that quotient by 255 to get our resulting image.**

**Image 3a:** We begin with the original university.png. **This is the input image.**

**Image 3b:** This is our original image’s histogram, computed via cv2’s calcHist function and displayed via matplotlib.pyplot’s plot function.

**Image 3c:** This is our output image after equalization. We can see that the image has an increased contrast ratio. We also see that histogram eq preserves our darker areas better than the log or power transformations did.

**Image 3d:** This is the histogram of our output image. We can see that while we still have a sharp spike, our values now span the entire range of the 8bit greyscale, rather than just (0,50).

|  |  |
| --- | --- |
| **(a)** | **(b)** |
| **(c)** | **(d)** |

**Figure 3: (a)** Input image (input/university.png), **(b)** histogram of original image’s grey-level distribution (output/hist/university/histogramBefore.jpeg), (c) The output university image after equalization (output/hist/university/uniEqualized.png), (d) histogram of original image’s grey-level distribution after equalization (output/hist/university/histogramAfter.jpg)

**2.8 Discussion**

In this project, we tackled log and power transformers, histogram manipulation and equalization for both greyscale and color images, and noise reduction. For log transformations, we learned that varying the constant factor can lead to a large range of results, and finding the right factor for an image is important. In the power transformations, we found that the in general, lower the y power is, the brighter the image is. By using 255 as our scaling constant, and dividing our power by several gamma values, we found that the gamma plays a large role in how the contrast of the image is perceived. In the greyscale histogram equalization, we found that by using a formula to normalize the data into a range (0,1) and scaling it to 8-bit greyscale (0,255), we can take an image and expand its contrast range to better see details in the image. We also saw that in the color equalization, by manipulating the histograms for each color then equalizing the colors for the satmap image we went from an almost greyscale looking washed out image, to a full color satellite image you would expect to see on google maps. Finally, in the noise reduction, we learned that in general as we increased our filter size, we see better noise reduction but a significant increase in blur effect each step up. We also see that the median filtering is much better at handling noise reduction than average filtering, with the 3x3 median filter being more effective than even the 7x7 average filter while minimizing how much blur we take on. Given more time, it would be nice to test how these filtering and histogram techniques can be used on a wider variety of images and how it can be applied to real world projects.

**3. CODES**

**3.1 Code for Sharpening.py**

# Michael Rizig

# Project 3: Morphological Filers

# 001008703

# File 1: Sharpening

# 7/5/2024

#Import nessessary tools

import matplotlib.pyplot as plottool

import cv2

from skimage import io

import numpy as np

# below function defines the which is the sharpening filter used

def laplacian(i,j):

    #instead of using loops, im utilizing linear alg to turn the filter into a linar combinaiton of the two maticies. (c1v1 + c2v2 .. cmvm)

    s= padded[i-1, j-1]\*lapFilter[0][0]+padded[i-1, j]\*lapFilter[0][1]+padded[i-1, j + 1]\*lapFilter[0][2]+padded[i, j-1]\*lapFilter[1][0]+ padded[i][j]\*lapFilter[1][1]+padded[i, j + 1]\*lapFilter[1][2]+padded[i + 1, j-1]\*lapFilter[2][0]+padded[i + 1, j]\*lapFilter[2][1]+padded[i + 1, j + 1]\*lapFilter[2][2]

    #correcting for negative values by taking absolute value

    if(s[1]<0):

        return 256- s

    return s

def sobel(i,j):

    s=0

    #take the defined sobel mask/filter and apply it to each pixel in the range x-1,y-1 to x+1,y+1

    #as before, we are utilizing linar combinations.

    #one pass for vertical

    s= padded[i-1, j-1]\*sobelV[0][0]+padded[i-1, j]\*sobelV[0][1]+padded[i-1, j + 1]\*sobelV[0][2]+padded[i, j-1]\*sobelV[1][0]+ padded[i][j]\*sobelV[1][1]+padded[i, j + 1]\*sobelV[1][2]+padded[i + 1, j-1]\*sobelV[2][0]+padded[i + 1, j]\*sobelV[2][1]+padded[i + 1, j + 1]\*sobelV[2][2]

    #one pass for horizontal

    s+= padded[i-1, j-1]\*sobelH[0][0]+padded[i-1, j]\*sobelH[0][1]+padded[i-1, j + 1]\*sobelH[0][2]+padded[i, j-1]\*sobelH[1][0]+ padded[i][j]\*sobelH[1][1]+padded[i, j + 1]\*sobelH[1][2]+padded[i + 1, j-1]\*sobelH[2][0]+padded[i + 1, j]\*sobelH[2][1]+padded[i + 1, j + 1]\*sobelH[2][2]

    #correcting for negative values by taking absolute value

    if(s[1]<0):

        return 256- s

    return s

#define lapical filter we are using (from slides):

lapFilter = [[1,1,1],[1,-8,1],[1,1,1]]

#sobel filters vertical and horizontal (from slides):

sobelV = [[-1,-2,-1],[0,0,0],[1,2,1]]

sobelH = [[-1,0,1],[-2,0,2],[-1,0,1]]

#define path

path = 'input/moon.jpg'

#read image

moonimage = io.imread(path)

#cvt color

moonimage = cv2.cvtColor(moonimage,cv2.COLOR\_BGR2RGB)

#created padded version (for edges of image)

padded = cv2.copyMakeBorder(moonimage,1,1,1,1,cv2.BORDER\_CONSTANT,value=[0,0,0])

#define an output image to be same dimentions etc as input image

output = moonimage.copy()

#create a blank image for lapician filter results

lapic = np.zeros([moonimage.shape[0], moonimage.shape[1], 3], dtype=np.uint8)

#create a blank image for sobel filter results

sob = np.zeros([moonimage.shape[0], moonimage.shape[1], 3], dtype=np.uint8)

#display initial image

plottool.imshow(padded)

plottool.show()

#apply lapician image enhancement

for i in range(moonimage.shape[0]):

   for j in range(moonimage.shape[1]):

        lapic[i][j] = laplacian(i,j)

        output[i][j] = output[i][j]+lapic[i][j]

#display the lapician filter generated

plottool.imshow(lapic)

plottool.show()

#save filter

#cv2.imwrite('output/sharpen/laplacian/laplacianFilter1.jpg',lapic)

#display output image (output - lapican filter generated)

plottool.imshow(output)

plottool.show()

#save output

#cv2.imwrite('output/sharpen/laplacian/outLaplace.jpg',output)

#reset output for sobel

output = moonimage.copy()

#apply sobel

for i in range(moonimage.shape[0]):

   for j in range(moonimage.shape[1]):

        sob[i][j] = sobel(i,j)

#combine sobel

output=output + sob

#display the sobel filter generated

plottool.imshow(sob)

plottool.show()

#save filter

cv2.imwrite('output/sharpen/sobel/sobelFilter.jpg',sob)

#display output image (output - sobel filter generated)

plottool.imshow(output)

plottool.show()

#save output

cv2.imwrite('output/sharpen/sobel/outSobel.jpg',output)

**3.2 Code for MorphologicalProcessing.py**

# Michael Rizig

# Project 3: Morphological Filers

# 001008703

# File 2: Morph1

# 7/5/2024

#Import nessessary tools

import matplotlib.pyplot as plottool

import cv2

from skimage import io

import numpy as np

def tresh(x):

    if x>150:

        return [0,0,0]

    return [255,255,255]

def dialation(i,j):

    s=0

    s= padded[i-1, j-1]\*struct[0][0]+padded[i-1, j]\*struct[0][1]+padded[i-1, j + 1]\*struct[0][2]+padded[i, j-1]\*struct[1][0]+ padded[i][j]\*struct[1][1]+padded[i, j + 1]\*struct[1][2]+padded[i + 1, j-1]\*struct[2][0]+padded[i + 1, j]\*struct[2][1]+padded[i + 1, j + 1]\*struct[2][2]

    if s[0]>0:

        return [255,255,255]

    return [0,0,0]

def erosion(i,j):

    s=0

    s= padded[i-1][j-1][0]\*struct[0][0]+padded[i-1][j][0]\*struct[0][1]+padded[i-1][j + 1][0]\*struct[0][2]+padded[i][j-1][0]\*struct[1][0]+ padded[i][j][0]\*struct[1][1]+padded[i][j + 1][0]\*struct[1][2]+padded[i + 1][j-1][0]\*struct[2][0]+padded[i + 1][j][0]\*struct[2][1]+padded[i + 1][j + 1][0]\*struct[2][2]

    if s>1024: # value chosed bc 256 \* 4 = 1024, if the value is greater than 1024, than 5 pixels must have been hit

        return [255,255,255]

    return [0,0,0]

def dialation5(i,j):

    s=0

    s= padded[i-2][j-2]\*struct1[0][0] + padded[i-2][j-1]\*struct1[0][1] + padded[i-2][j]\*struct1[0][2] + padded[i-2][j+1]\*struct1[0][3] + padded[i-2][j+2]\*struct1[0][4] + padded[i-1][j-2]\*struct1[1][0] + padded[i-1][j-1]\*struct1[1][1] + padded[i-1][j]\*struct1[1][2] + padded[i-1][j+1]\*struct1[1][3] + padded[i-1][j+2]\*struct1[1][4] +  padded[i][j-2]\*struct1[2][0] + padded[i][j-1]\*struct1[2][1] + padded[i][j]\*struct1[2][2] + padded[i][j+1]\*struct1[2][3] + padded[i][j+2]\*struct1[2][4] +  padded[i+1][j-2]\*struct1[3][0] + padded[i+1][j-1]\*struct1[3][1] + padded[i+1][j]\*struct1[3][2] + padded[i+1][j+1]\*struct1[3][3] + padded[i+1][j+2]\*struct1[3][4] + padded[i+2][j-2]\*struct1[4][0] + padded[i+2][j-1]\*struct1[4][1] + padded[i+2][j]\*struct1[4][2] + padded[i+2][j+1]\*struct1[4][3] + padded[i+2][j+2]\*struct1[4][4]

    if s[0]>0:

# value chosed bc if the value is greater than 0, than at least 1 must have been hit

        return [255,255,255]

    return [0,0,0]

def erosion5(i,j):

    s=0

    s= padded[i-2][j-2][0]\*struct1[0][0] + padded[i-2][j-1][0]\*struct1[0][1] + padded[i-2][j][0]\*struct1[0][2] + padded[i-2][j+1][0]\*struct1[0][3] + padded[i-2][j+2][0]\*struct1[0][4] + padded[i-1][j-2][0]\*struct1[1][0] + padded[i-1][j-1][0]\*struct1[1][1] + padded[i-1][j][0]\*struct1[1][2] + padded[i-1][j+1][0]\*struct1[1][3] + padded[i-1][j+2][0]\*struct1[1][4] +  padded[i][j-2][0]\*struct1[2][0] + padded[i][j-1][0]\*struct1[2][1] + padded[i][j][0]\*struct1[2][2] + padded[i][j+1][0]\*struct1[2][3] + padded[i][j+2][0]\*struct1[2][4] +  padded[i+1][j-2][0]\*struct1[3][0] + padded[i+1][j-1][0]\*struct1[3][1] + padded[i+1][j][0]\*struct1[3][2] + padded[i+1][j+1][0]\*struct1[3][3] + padded[i+1][j+2][0]\*struct1[3][4] + padded[i+2][j-2][0]\*struct1[4][0] + padded[i+2][j-1][0]\*struct1[4][1] + padded[i+2][j][0]\*struct1[4][2] + padded[i+2][j+1][0]\*struct1[4][3] + padded[i+2][j+2][0]\*struct1[4][4]

    if s>2048: # value chosed bc 256 \* 8 = 2048, if the value is greater than 2048, than 9 pixels must have been hit

        return [255,255,255]

    return [0,0,0]

#image path

impath = 'input/fingerprint.jpg'

#import image

fingerprint = io.imread(impath)

#fix colors

fingerprint = cv2.cvtColor(fingerprint,cv2.COLOR\_BGR2RGB)

#create ouput image:

output = fingerprint.copy()

#display image

plottool.imshow(fingerprint)

plottool.show()

#calculate histogram for image using cv2

hist = cv2.calcHist([fingerprint], [0], None, [256], [0,256])

#display original image's histogram distribution

plottool.plot(hist)

plottool.title("Fingerprint image Histogram Dist:")

plottool.savefig('output/morph/finger/hist.png')

plottool.show()

#define output for thresholding

thresholdImage = fingerprint.copy()

#apply threshold function

for i in range(fingerprint.shape[0]):

    for j in range(fingerprint.shape[1]):

        thresholdImage[i][j] = tresh(fingerprint[i][j][0])

#display results of threshold

plottool.imshow(thresholdImage)

plottool.title("Thresholded Image:")

plottool.show()

#created padded verion for filtering:

padded = cv2.copyMakeBorder(thresholdImage,1,1,1,1,cv2.BORDER\_CONSTANT,value=[0,0,0])

#save threshold image

cv2.imwrite('output/morph/finger/thresh.jpg',thresholdImage)

#define structuring element:

struct = [[0,1,0],[1,1,1],[0,1,0]]

struct1=[[0,0,1,0,0],[0,0,1,0,0],[1,1,1,1,1],[0,0,1,0,0],[0,0,1,0,0]]

#3x3 dialate image:

for i in range(fingerprint.shape[0]):

    for j in range(fingerprint.shape[1]):

        output[i][j] = dialation(i,j)

#display results of dialation

plottool.imshow(output)

plottool.title("Dialated 3x3:")

plottool.show()

#save results for dialation

cv2.imwrite('output/morph/finger/dialated.jpg',output)

#updated padded

padded =  cv2.copyMakeBorder(output,1,1,1,1,cv2.BORDER\_CONSTANT,value=[0,0,0])

#3x3 erode image:

for i in range(fingerprint.shape[0]):

    for j in range(fingerprint.shape[1]):

        output[i][j] = erosion(i,j)

#display results of 3x3 erosion

plottool.imshow(output)

plottool.title("Eroded 3x3:")

plottool.show()

#save results for erosion

cv2.imwrite('output/morph/finger/eroded.jpg',output)

#reset padded image for 5x5 filter

padded =  cv2.copyMakeBorder(thresholdImage,2,2,2,2,cv2.BORDER\_CONSTANT,value=[0,0,0])

#5x5 dialaion

for i in range(fingerprint.shape[0]):

    for j in range(fingerprint.shape[1]):

        output[i][j] = dialation5(i,j)

#display results of 5x5 dialation

plottool.imshow(output)

plottool.title("Dialated 5x5:")

plottool.show()

#save results for dialation

cv2.imwrite('output/morph/finger/dialated5.jpg',output)

#updated padded

padded =  cv2.copyMakeBorder(output,2,2,2,2,cv2.BORDER\_CONSTANT,value=[0,0,0])

#5x5 erode :

for i in range(fingerprint.shape[0]):

    for j in range(fingerprint.shape[1]):

        output[i][j] = erosion5(i,j)

#display results of 5x5 erosion

plottool.imshow(output)

plottool.title("Eroded 5x5:")

plottool.show()

#save results for 5x5 morphological erosion

cv2.imwrite('output/morph/finger/eroded5.jpg',output)

**3.3 Code for histogram equalization (color) (Histogram-color.py)**

# Michael Rizig

# Project 2: Image Enhancement

# 001008703

# File 3: Histogram Color Equalization

# 6/12/2024

# nessesary imports

import cv2

from skimage import io

import matplotlib.pyplot as plottool

#define the equalization funciton for this image

# finding a function for this image was more challanging than the previous uni image

# by analizing the histogram we can see that the graph centers around the range (165,220) and has a range with of around 65

# we also see that there are a few outlier pixels in the extremes that would give us issues

def equalizationFunc(pixel):

    # to account for low outliers that would give weird artifacts in the output, I use a simple if statement

    if pixel <150:

        #if the pixel is below our main range, set it equal to 0 to prevent strange artifacting

        pixel =0

    else:

        #else we subtract the pixel from out lower bound so that we dont lose any colors on the high end, and that we convert the problem back to a simple range expansion in one directoin

        pixel -=150

    # by using the same concept from the previous histogram eq file, we divide each pixel by our range, and scale the (0,1) result to (0,255) by multiplying

    return 255 \* (pixel/65)

#define our color range and max color value for the histogram

colorrange = [0,256]

maxsize= [256]

#as usual we set path and use skimage to open file

satpath = 'input/sat\_map.png'

satImage = io.imread(satpath)

#print(satImage[0][0])

#define out output image

satImage = cv2.cvtColor(satImage,cv2.COLOR\_BGR2RGB)

out = satImage.copy()

#sshow our input image before processing

plottool.imshow(satImage)

plottool.show()

#calculate histogram for first channel (B)

hist = cv2.calcHist([satImage], [0], None, maxsize, colorrange)

#plot histrogram to show our original image's B channel

plottool.plot(hist)

plottool.title("B distribution Before:")

plottool.show()

#histogram correction for B

for k in range(0,satImage.shape[0]):

    for p in range(0,satImage.shape[1]):

        out[k][p][0] = equalizationFunc(satImage[k][p][0])

#show output image after our B channel correction

plottool.imshow(out)

plottool.show()

out = cv2.cvtColor(out,cv2.COLOR\_BGR2RGB)

cv2.imwrite('output/hist/satmap/B-Corrected.png',out)

#show equalized historgram for B channel after processing

hist = cv2.calcHist([out], [0], None, maxsize, colorrange)

plottool.plot(hist)

plottool.title("B distribution After EQ:")

plottool.show()

hist = cv2.calcHist([out], [1], None, maxsize, colorrange)

plottool.plot(hist)

plottool.title("G distribution Before EQ:")

plottool.show()

#histogram correction for G

for k in range(0,satImage.shape[0]):

    for p in range(0,satImage.shape[1]):

        out[k][p][1] = equalizationFunc(satImage[k][p][1])

#show output image after G channel histogram equalization

plottool.imshow(out)

plottool.show()

out = cv2.cvtColor(out,cv2.COLOR\_BGR2RGB)

cv2.imwrite('output/hist/satmap/G-Corrected.png',out)

# show equalized histogram for g values

hist = cv2.calcHist([out], [1], None, maxsize, colorrange)

plottool.plot(hist)

plottool.title("G distribution After EQ:")

plottool.show()

hist = cv2.calcHist([out], [2], None, maxsize, colorrange)

plottool.plot(hist)

plottool.title("R distribution Before EQ:")

plottool.show()

#correct R values

for k in range(0,satImage.shape[0]):

    for p in range(0,satImage.shape[1]):

        out[k][p][2] = equalizationFunc(satImage[k][p][2])

#show resulting image

plottool.imshow(out)

plottool.show()

out = cv2.cvtColor(out,cv2.COLOR\_BGR2RGB)

cv2.imwrite('output/hist/satmap/R-Corrected.png',out)

#show resulting histo

hist = cv2.calcHist([out], [2], None, maxsize, colorrange)

plottool.plot(hist)

plottool.title("R distribution After EQ:")

plottool.show()