Angel U. Ortega
Computer Vision – Final Project
Version 2.0
12/9/2016

Table of Contents

1. Introduction

1.1. Project Overview

The project consists of a color calibration program that uses a color passport to determine color difference. I will be implementing two different approaches (A1 and A2) to determine the best solution. The two approaches are described below:

A1. Utilize a single color-point to determine color difference.
A2. Utilize a full color passport to determine color difference through the use of nearest neighbors.

1.2. References

[1] https://drive.google.com/open?id=0B_kWRxLZdmeJNDFtNnI1Mi05bFE
[2] https://www.dpreview.com/articles/6497352654/get-more-accurate-color-with-camera-calibration-

2. Proposed Solution, Design, and Implementation

2.1. Color Passport

A color passport is a collection of multiple colors from varying hues. The color passports I will be using for this lab can be seen in Figure 1.

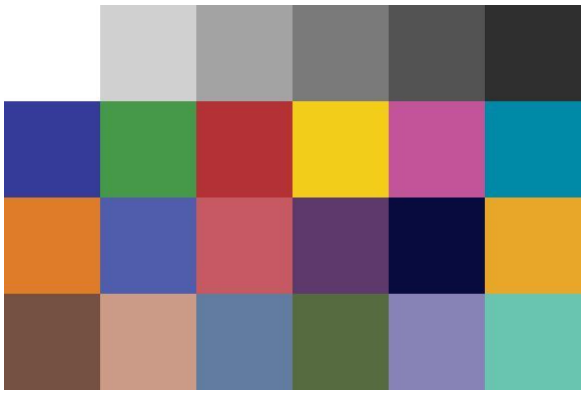


Figure 1: Color passport using multiple colors and hues.

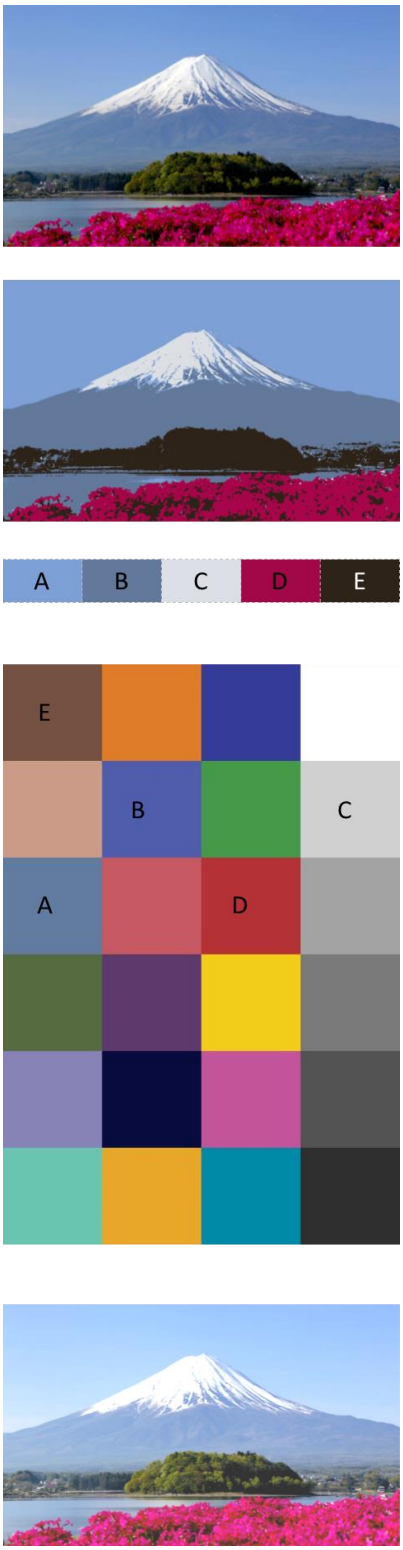
2.2. A1

For this approach I will be using a single color selection to calibrate entire picture. I will be utilizing a true color, calculating color difference between known color and retrieved color, and recoloring entire image using color difference. I collect the color from one pixel in the image, then select the color from the passport, and retrieve color difference. I then apply this color difference to each color channel in the image, and display the modified image. Results for A1 can be seen in Section 3.1.

2.3. A2

For this approach I will be using the multiple-color color passport seen in Figure 1. I will calculate color difference for each color in the color passport. I will then utilize kmeans to retrieve average colors from the image, and calibrate original image using corresponding cluster from kmeans and the matching color difference from the nearest neighbor of the color passport. Figure 2 explains the approach in more detail. First, we retrieve average colors using kmeans. We

match each color to a color from the passport. Then, I get the color difference from the matching passport color, and adjust image using corresponding color difference. Results for A2 can be seen in Section 3.2



Approach Algorithm

- We begin with a color image and its corresponding color passport capture.
- We apply k-means pixel clustering to retrieve n average colors. I will try with different values for n to compare results.
- Then we retrieve the color array. It would be useful to also generate an index image (an image of labels instead of colors) for future use.
- Then we must map the image's color passport values to those of the original color passport (true colors).
- Retrieve color difference for every pair mapped.
- Identify nearest neighbors of image's average colors and colors in image's passport.
- Generate n color calibrated images using matching colors from color passport.
- Using index image generated earlier, create the n-color calibrated image using the index from index image, and color calibrated images.
- The resulting n-color calibrated image should be more accurate to the real color image.

Figure 2: A2 Algorithm

To map one color passport to another I had created a version of the color passport that would organize its colors by size. Figure 2A shows this color passport. The issue with using this color passport was that the user would need to click the four corners so my program could begin extracting the colors. Another option was for the user to click on a color in one passport and then click the corresponding color in the other passport to do the mapping. This too would have been too much clicking for the user to do. In the end, I figured out a way in which the user only needs to click two corner squares in each color passport to create the mapping. This method does not require for me to warp the picture of the color passport, nor click every single color square on both passports. After the user clicks four times, the program runs until completion.

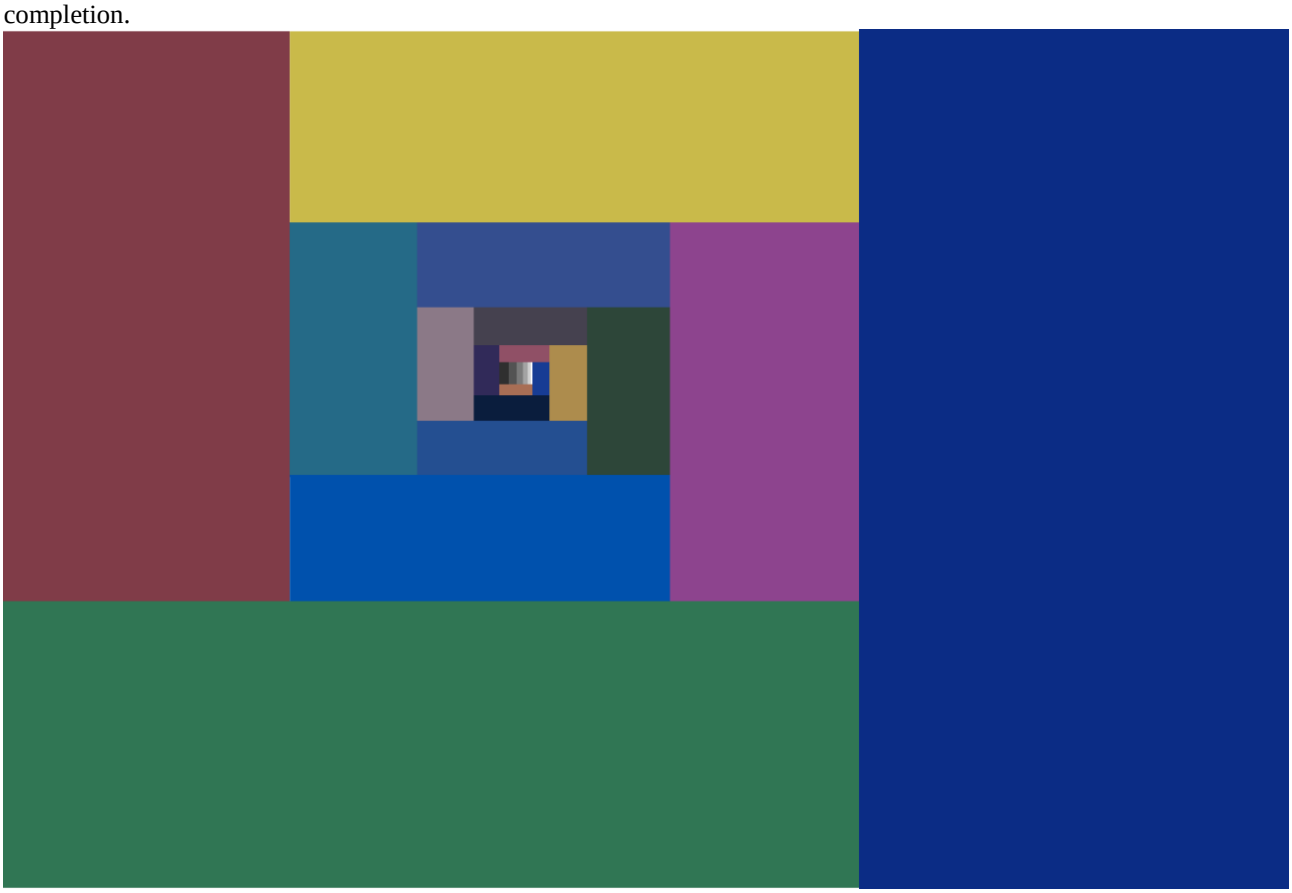


Figure 2A: Color passport that uses area to organize colors

3. Experimental Results and Conclusions

The images used for this lab can be retrieved from:
https://drive.google.com/open?id=0B_kWRxLZdmeJNDFtNnI1Mi05bFE [1]
I utilized two pairs of images to evaluate accuracy of my programs. The images are property of Steve Davey and can be retrieved from [2].

3.1. A1

Figure 3 shows the original and recalibrated images using A1. Note that the results are not perfect, especially with colors that are not similar to sample. The program, however, is very fast, clocking in at approximately 0.02 seconds for an image of size 896 x 597 pixels. Identifying multiple sample points, and using the average color difference yielded worse results. A better solution would adjust colors with similar hues with the same color difference, while using a different color difference for colors with different hues. It can be seen in Figure 3 that while the robes of the monks are much better calibrated to the true color, everything that is not orange gets a redish tint. Section 3.2 shows the results of using the same images, but also k-means pixel clustering, nearest neighbors and color blocking for better results.

Figure 3: From top to bottom, left to right: Original image, calibrated image, and runtime.

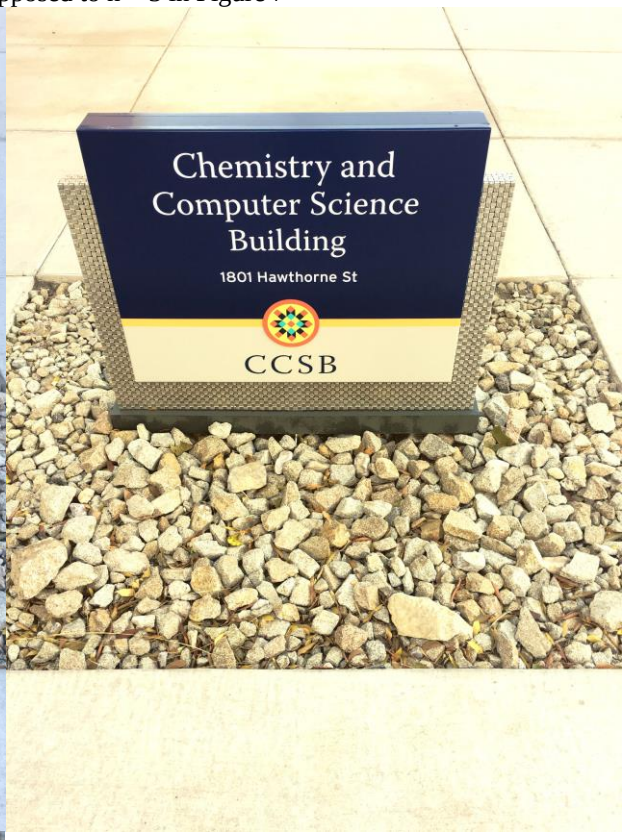
3.2. A2

Figures 4-## show results for A2. Figures 4 and 5 show the complete steps from the algorithm to exemplify how the program works. As it can be seen when we compare the calibrated image from the source, and the calibrated image resulting from the program, the generated calibrated image is more accurate, in comparison to only using one color to calibrate, like in A1.

Figure 4: Calibrated image from source and calibrated image generated from program.

Figure 5: From left to right, top to bottom: Original image, calibrated image (from source), k-means pixel clustering using n=3, the three resulting color calibrated images using the average colors, final 3-color calibrated image, and runtimes of each component.

Figures 6 and 7 compare a picture taken in broad daylight at The University of Texas at El Paso and the resulting calibration by using a color passport. As expected, the runtime for this particular test was much higher. The reason behind it is the size of the images. The images used for this test were of size 3000 x 4000 pixels. The k-means pixel clustering also takes up a lot of time. Figure 7 was much faster than Figure 6 because the images used in Figure 7 were of size 640 x 480 pixels. I also used n = 10 in Figure 6, as opposed to n = 3 in Figure 7



```
Runtime for getPassportsColorDifference: 10.5890002251 seconds
returning from posterize
Runtime for posterize: 300.128999949 seconds
Runtime for nearest neighbor: 0.00300002098083 seconds
Total runtime: 486.164000034 seconds
```

Figure 6: Original image, calibrated image, color passport picture, true color passport, and runtime.

```
Runtime for getPassportsColorDifference: 8.39199995995 seconds
Runtime for posterize: 9.59500002861 seconds
Runtime for nearest neighbor: 0.00300002098083 seconds
Total runtime: 32.5709998608 seconds
```

Figure 7: Original image, calibrated image, color passport picture, true color passport, and runtime.

4. Appendix

4.1. A1 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 26 12:06:58 2016

@author: auort
"""

import numpy as np
from PIL import Image
import scipy.misc
import time
from pylab import *
from numpy import *

def getSelection(Path):
```

```
    image = np.array(Image.open(Path))
    imshow(image)
    refPt = ginput(1)
    point = refPt[0]
    xcoor = int(point[0])
    ycoor = int(point[1])
    color = image[ycoor][xcoor]
    return color

def getPixel(Path,x,y):
    image = np.array(Image.open(Path))
    return image[x][y]

def runCallibration(Path1, Path2, Path3):
    showColor = getSelection(Path1)
    trueColor = getSelection(Path2)
    callibratedImage = callibrateImage(Path1, showColor, trueColor)
    scipy.misc.imsave(Path3, callibratedImage)

def callibrateImage(Path, showColor, trueColor):
    start_time = time.time()
    rT = trueColor[0]
    gT = trueColor[1]
    bT = trueColor[2]
    rS = showColor[0]
    gS = showColor[1]
    bS = showColor[2]

    colordifference = trueColor - showColor
    print "color difference:"
    print colordifference
    image = np.array(Image.open(Path))

    isRedSum = (trueColor[0]>showColor[0])
    isGreenSum = (trueColor[1]>showColor[1])
    isBlueSum = (trueColor[2]>showColor[2])
    print "isRedSum: %s" % isRedSum
    print "isGreenSum: %s" % isGreenSum
    print "isBlueSum: %s" % isBlueSum

    rabsdif = abs(rT-rS)
    gabsdif = abs(gT-gS)
    babsdif = abs(bT-bS)
    if (not(isRedSum)):
        rabsdif = 255-rabsdif
    if (not(isGreenSum)):
        gabsdif = 255-gabsdif
    if (not(isBlueSum)):
        babsdif = 255-babsdif
    print "red abs diff: %s" %rabsdif
    print "green abs diff: %s" %gabsdif
    print "blue abs diff: %s" %babsdif

    red, green, blue = image[:,:,0], image[:,:,1], image[:,:,2]

    redNthres = rabsdif
    greenNthres = gabsdif
    blueNthres = babsdif

    redPthres = 255-rabsdif
    greenPthres = 255-gabsdif
    bluePthres = 255-babsdif


    print "Negative Thresholds:"
    print "     redNthres: %s" %redNthres
    print "     greenNthres: %s" %greenNthres
    print "     blueNthres: %s" %blueNthres
    print "Positive Thresholds:"
    print "     redPthres: %s" %redPthres
    print "     greenPthres: %s" %greenPthres
```

```
    print "     bluePthres: %s" %bluePthres



    if (isRedSum):
        r = red+redNthres
        r[red>redPthres]=255
    else:
        r = red-redNthres
        r[red<redNthres]=0
    if (isGreenSum):
        g = green+greenNthres
        g[green>greenPthres]=255
    else:
        g = green-greenNthres
        g[green<greenNthres]=0
    if(isBlueSum):
        b = blue+blueNthres
        b[blue>bluePthres]=255
    else:
        b = blue-blueNthres
        b[blue<blueNthres]=0

    r[r > 255] = 255
    r[r < 0] = 0
    g[g > 255] = 255
    g[g < 0] = 0
    b[b > 255] = 255
    b[b < 0] = 0
    image[:,:,0] = r
    image[:,:,1] = g
    image[:,:,2] = b


    imshow(image)
    end_time = time.time()
    runtime = end_time - start_time
    print "Runtime: %s seconds" % runtime
    return image

runCallibration('monk.jpeg','monk2.jpeg','monkcalibrated.jpeg')
```

4.2. A2 Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 26 12:06:58 2016

@author: auort
"""

import numpy as np
from PIL import Image
import scipy.misc
import time
from pylab import *
from numpy import *
import copy
import cv2
import scipy

def getSelection(Path):
    image = np.array(Image.open(Path))
    imshow(image)
    refPt = ginput(1)
    point = refPt[0]
    xcoor = int(point[0])
    ycoor = int(point[1])
    color = image[ycoor][xcoor]
```

```
    return color

def getPixel(Path,x,y):
    image = np.array(Image.open(Path))
    return image[x][y]

def runPassportCallibrations(Path1, Path2):
    thresholds = np.zeros((24,9))
    for i in range(24):
        showColor = getSelection(Path1)
        trueColor = getSelection(Path2)
        thresholds[i]=getThres(trueColor, showColor)
    #print thresholds

def getThres(trueColor, showColor):
    rT = trueColor[0]
    gT = trueColor[1]
    bT = trueColor[2]
    rS = showColor[0]
    gS = showColor[1]
    bS = showColor[2]

    isRedSum = (trueColor[0]>showColor[0])
    isGreenSum = (trueColor[1]>showColor[1])
    isBlueSum = (trueColor[2]>showColor[2])

    rabsdif = abs(rT-rS)
    gabsdif = abs(gT-gS)
    babsdif = abs(bT-bS)
    if (not(isRedSum)):
        rabsdif = 255-rabsdif
    if (not(isGreenSum)):
        gabsdif = 255-gabsdif
    if (not(isBlueSum)):
        babsdif = 255-babsdif

    redNthres = rabsdif
    greenNthres = gabsdif
    blueNthres = babsdif

    redPthres = 255-rabsdif
    greenPthres = 255-gabsdif
    bluePthres = 255-babsdif

    if(redPthres<redNthres):
        temp = redPthres
        redPthres=redNthres
        redNthres=temp
    if(greenPthres<greenNthres):
        temp=greenPthres
        greenPthres=greenNthres
        greenNthres=temp
    if(bluePthres<blueNthres):
        temp=bluePthres
        bluePthres=blueNthres
        blueNthres=temp

    return redNthres, greenNthres, blueNthres, redPthres, greenPthres, bluePthres, isRedSum, isGreenSum, isBlueSum

def createCallibratedImage(Path, values):
    start_time = time.time()

    image = np.array(Image.open(Path))

    redNthres = values[0]
    greenNthres = values[1]
    blueNthres = values[2]
    redPthres = values[3]
    greenPthres = values[4]
    bluePthres = values[5]
    isRedSum = values[6]
```

```
    isGreenSum = values[7]
    isBlueSum = values[8]

    red, green, blue = image[:,:,0], image[:,:,1], image[:,:,2]

    if (isRedSum):
        r = red+redNthres
        r[red>redPthres]=255
    else:
        r = red-redNthres
        r[red<redNthres]=0
    if (isGreenSum):
        g = green+greenNthres
        g[green>greenPthres]=255
    else:
        g = green-greenNthres
        g[green<greenNthres]=0
    if(isBlueSum):
        b = blue+blueNthres
        b[blue>bluePthres]=255
    else:
        b = blue-blueNthres
        b[blue<blueNthres]=0

    r[r > 255] = 255
    r[r < 0] = 0
    g[g > 255] = 255
    g[g < 0] = 0
    b[b > 255] = 255
    b[b < 0] = 0
    image[:,:,0] = r
    image[:,:,1] = g
    image[:,:,2] = b

    end_time = time.time()
    runtime = end_time - start_time
    print "Runtime of Image Callibration: %s seconds" % runtime
    return image

def getPassportCoords(Path):
    image = np.array(Image.open(Path))
    imshow(image)
    refPt = ginput(2)
    close()
    pointI = refPt[0]

    Ix = int(pointI[0])
    Iy = int(pointI[1])
    pointF = refPt[1]

    Fx = int(pointF[0])
    Fy = int(pointF[1])

    #coordsv = np.zeros((24,2))
    coordsa = np.zeros((6,4,2))
    colors = np.zeros((6,4,3))
    for i in range (6):
        x = 0
        y = 0
        width = Fx-Ix
        height = Fy-Iy
        x = (width/5)* i + Ix
        for j in range(4):
            y = (height/3)* j +Iy
            coordsa[i][j][0]=x
            coordsa[i][j][1]=y
            colors[i][j][0]=image[y][x][0]
            colors[i][j][1]=image[y][x][1]
            colors[i][j][2]=image[y][x][2]

    return coordsa, colors
```

```
def getPassportsColorDifference(Path1, Path2):
    coorA, colorA = getPassportCoords(Path1)
    coorB, colorB = getPassportCoords(Path2)
    pCD = np.zeros((6,4,9))#passportColorDifference
    for i in range(6):
        for j in range(4):

pCD[i][j][0],pCD[i][j][1],pCD[i][j][2],pCD[i][j][3],pCD[i][j][4],pCD[i][j][5],pCD[i][j][6],pCD[i][j][7],pCD[i][j][
colorA[i][j])

    #print "Finished getting passports' color difference"
    return pCD, colorA

def getimage(array, label, img):
    #print "getting image for posterize"
    array = np.uint8(array)#convert from float to int
    result = array[label.flatten()]

    resultimage = result.reshape((img.shape))#convert to original shape
    height = len(img)
    width = len(img[0])
    indeximage = np.zeros((height, width))
    #print "creating indeximage"
    for i in range (len(resultimage)):
        for j in  range (len(resultimage[0])):
            for k in range (len(array)):
                if (array[k][0]==resultimage[i][j][0] and array[k][1]==resultimage[i][j][1] and array[k][2]==resul
                    indeximage[i][j]=k
    return resultimage, array, indeximage

def posterize(imagename, numcolors, savedirectory):
    #print "Posterize called!"
    img = cv2.imread(imagename)
    #img2 = cv2.imread("//utep//passport.jpg")
    array = img.reshape((-1,3))
    array = np.float32(array) #cv2.kmeans needs float array, soarray must be converted
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)#stop iteration if epsilon is reached
occured; max iterations = 10, epsilon = 1.0
    ret,label,result=cv2.kmeans(array, numcolors, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

    resultimage, colors, indeximage = getimage(result, label, img)
    cv2.imwrite(savedirectory, resultimage)
    #print "returning from posterize"
    return colors, indeximage

def runmulticalibration(Path1, Path2, Image, n, SaveDir):
    starttime = time.time()
    stime = time.time()
    pCD, pcolors = getPassportsColorDifference(Path1, Path2)
    etime = time.time()
    print "Runtime for getPassportsColorDifference: %s seconds" % (etime-stime)
    #print "Calling posterize..."
    stime = time.time()
    icolors, indeximage = posterize(Image, n, 'temporaryposter.jpg')
    etime = time.time()
    print "Runtime for posterize: %s seconds" % (etime-stime)
    #print "len(icolors) should be 2: %s" % len(icolors)
    indexes = np.zeros((len(icolors),2))
    #print "Entering nested for loop!"
    stime = time.time()

    for i in range (len(icolors)):
        indexes[i][0]=0
        indexes[i][1]=0
        value=254*3
        for j in range (len(pcolors)):
            for k in  range (len(pcolors[0])):

                red = abs(abs(icolors[i][0])-abs(pcolors[j][k][0]))
                green = abs(abs(icolors[i][1])-abs(pcolors[j][k][1]))
```

```
                blue = abs(abs(icolors[i][2])-abs(pcolors[j][k][2]))
                difference = red+green+blue


            difference = abs(red)+abs(green)+ abs(blue)
            #print "difference: %s" % difference

            if (difference < value):
                #print "new smaller difference"
                value = difference
                indexes[i][0] = j
                indexes[i][1] = k
                #print indexes[i]

    etime = time.time()
    print "Runtime for nearest neighbor: %s seconds" % (etime-stime)

    stime = time.time()
    #print "Starting calibrated image creation"

    calibratedimages = np.zeros((n, len(indeximage), len(indeximage[0]), 3))

    for i in range(n):
        identifier = str(i)
        x = indexes[i][0]
        y = indexes[i][1]
        image = createCallibratedImage(Image, pCD[x][y])
        calibratedimages[i]=image
        name = "calibratedimage" + identifier + ".jpg"
        scipy.misc.imsave(name, image)
        #imshow(image)

    length = len(indeximage)
    width = len(indeximage[0])
    finalimage = np.zeros((length,width, 3))
    for i in range (len(indeximage)):
        #print "running loop... %s /%s" % i, length
        for j in range (len(indeximage[0])):
            index = indeximage[i][j]
            finalimage[i][j][0] = calibratedimages[index][i][j][0]
            finalimage[i][j][1] = calibratedimages[index][i][j][1]
            finalimage[i][j][2] = calibratedimages[index][i][j][2]
    #imshow(finalimage)
    scipy.misc.imsave("FinalCalibratedImage.jpg", finalimage)
    endtime = time.time()
    print "Total runtime: %s seconds" % (endtime-starttime)

runmulticalibration('IMG_0218.jpg','passportlarge.jpg','IMG_0219.jpg',3,'multicalibration10.jpg')
```

4.3. Color Passport Processing Code

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 19 16:07:54 2016

@author: auort

Use k means clustering to posterize an image with n number of colors

"""

import numpy as np
import cv2
import scipy.misc
from PIL import Image
from pylab import *


def getarray(img):
```

```
    array = img.reshape((-1,3))
    array = np.float32(array) #cv2.kmeans needs float array, soarray must be converted
    return array

def getimage(array, label, img):
    array = np.uint8(array)#convert from float to int
    result = array[label.flatten()]
    resultimage = result.reshape((img.shape))#convert to original shape
    return resultimage

def posterize(imagename, numcolors, savedirectory):
    img = cv2.imread(imagename)
    array = getarray(img)
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)#stop iteration if
epsilon is reached or if max iterations has occured; max iterations = 10, epsilon = 1.0
    ret,label,result=cv2.kmeans(array, numcolors, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
    print result
    resultimage = getimage(result, label, img)
    cv2.imwrite(savedirectory, resultimage)

    '''
    cv2.imshow('posterized image using k means with k=' + str(numcolors) +' of '+ imagename,
resultimage)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    '''
def expandImage(Im):
    length = len(Im)*10
    height = len(Im[0])*10
    retIm = np.zeros((length, height, 3))
    for y in range(height):
        for x in range(length):
            indexx = int(x/10)
            indexy = int(y/10)
            retIm[x][y] = Im[indexx][indexy]
    return retIm

def getPassport():
    image = np.zeros((6,4,3))
    image[0][0]=[115,82,69]
    image[1][0]=[194,150,132]
    image[2][0]=[98,122,156]
    image[3][0]=[87,107,67]
    image[4][0]=[133,128,176]
    image[5][0]=[104,189,169]

    image[0][1]=[212,122,44]
    image[1][1]=[81,92,166]
    image[2][1]=[191,90,98]
    image[3][1]=[93,60,107]
    image[4][1]=[15,18,65]
    image[5][1]=[222,161,47]

    image[0][2]=[56,62,148]
    image[1][2]=[71,148,74]
    image[2][2]=[173,54,58]
    image[3][2]=[230,197,32]
    image[4][2]=[186,86,148]
    image[5][2]=[8,135,161]

    image[0][3]=[242,242,242]
    image[1][3]=[199,199,199]
    image[2][3]=[158,158,158]
    image[3][3]=[120,120,120]
    image[4][3]=[84,84,84]
    image[5][3]=[51,51,51]

    return image

def generatePassports():
    passportS = getPassport()
```

```
    scipy.misc.imsave('passportsmall.jpg', passportS)

    passportM = expandImage(passportS)
    scipy.misc.imsave('passportmedium.jpg', passportM)

    passportL = expandImage(passportM)
    scipy.misc.imsave('passportlarge.jpg', passportL)

    passportXL = expandImage(passportL)
    scipy.misc.imsave('passportextralarge.jpg', passportXL)

def getPassports():
    pS = np.array(Image.open('passportsmall.jpg'))
    pM = np.array(Image.open('passportmedium.jpg'))
    pL = np.array(Image.open('passportlarge.jpg'))
    pXL = np.array(Image.open('passportextralarge.jpg'))
    return pS, pM, pL, pXL

pS, pM, pL, pXL = getPassports()
```

5. End of Document

End of Document