Angel U. Ortega

Computer Vision – Lab 2

Version 1.0

9/27/2016

Table of Contents

1. Introduction

1.1. Lab Overview

The lab is composed of three problems, P1, P2, and P3(Optional). I worked on problems 1 and 2. The problem statement is as follows:

Your program should open image I1 and prompt the user to select a region of interest R using the mouse. It will then find and mark the region that is most similar to R in each of the images I2, .., In. As metric of similarity use:
P1. The pixel-to-pixel difference in the regions.
P2. The difference of the histograms of gradients of the regions. To obtain better results, you may want to partition the regions into subregions and use the concatenated histograms of the subregions as features
P3. Extra credit: SIFT descriptors, as explained in the textbook.

1.2. References

[1] http://www.pyimagesearch.com/2015/03/09/capturing-mouse-click-events-with-python-and-opencv/
[2] https://drive.google.com/open?id=0B_kWRxLZdmeJSTFoZ2syVmcwTXc

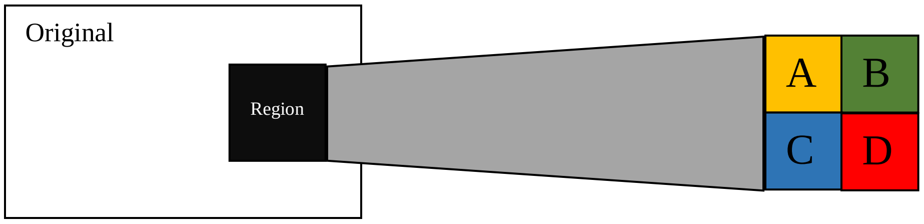
2. Proposed Solution Design and Implementation

2.1. P1

For this problem, I wanted to make my program as efficient as possible. I began with the region select for the image. I followed a tutorial from [1] to make my region select. From there, I decided to use the region selected (A), and subtract from a subregion of the image being compared (B). I did this by simply selecting a subregion from the 2d array that was my second image. I then subtracted A from B. I was able to make this process faster by selecting the subregion from the second image using advanced indexing. Since I used the size of A to derive B, the two 3d arrays were of the same size, allowing me to do a direct subtraction instead of having to parse through the different values within each array. I had to use two nested while loops to parse through the 5 pixel steps on the y- and x-axes. I also used cumsum to derive the average difference, and max to find a single-value average difference for an entire subregion. I constructed a simplified matrix with these average differences. I just had to find the min value in this matrix to find the area of interest. I created a function to show the compared image and the region of interest found outlined by a red rectangle.

2.2. P2

For this problem, I reused most of the code from P1. I had already created a HOGGen method, which would return the Histogram of Oriented Gradients of a region in an image and also of four subregions in an image:



I then found the difference between the four HOGS for the region selected (A), and the subregion of the image being compared (B). I then found the

2.3. P3

I did not work on this problem.

3. Experimental Results and Conclusions

All images used for this lab can be accessed at:
https://drive.google.com/open?id=0B_kWRxLZdmeJSTFoZ2syVmcwTXc [1]
Below are just a few examples from the images used and the images derived.

3.1. P1E1

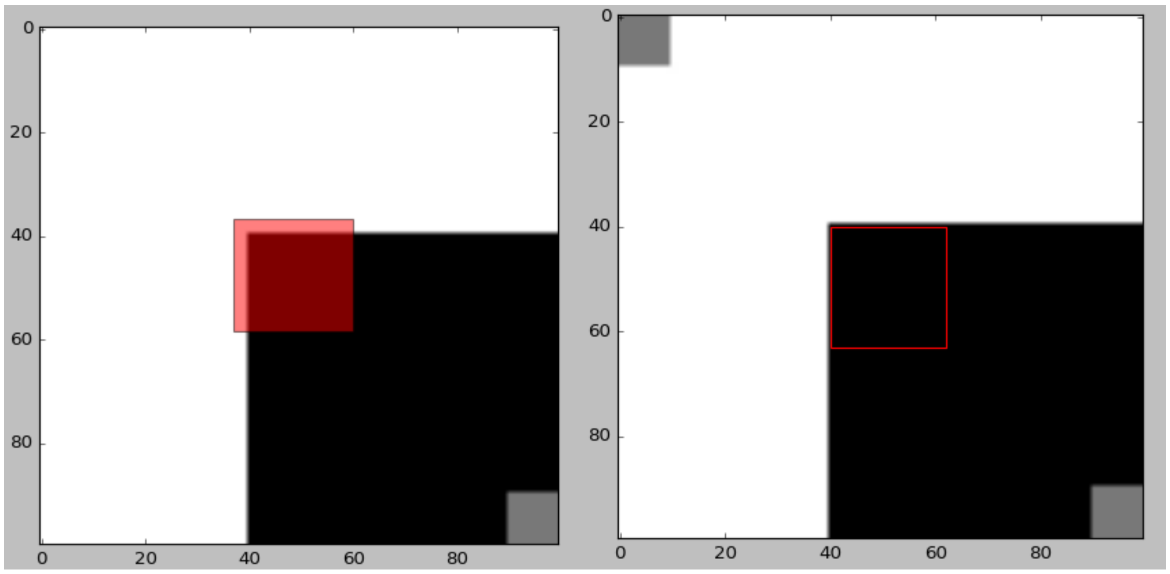
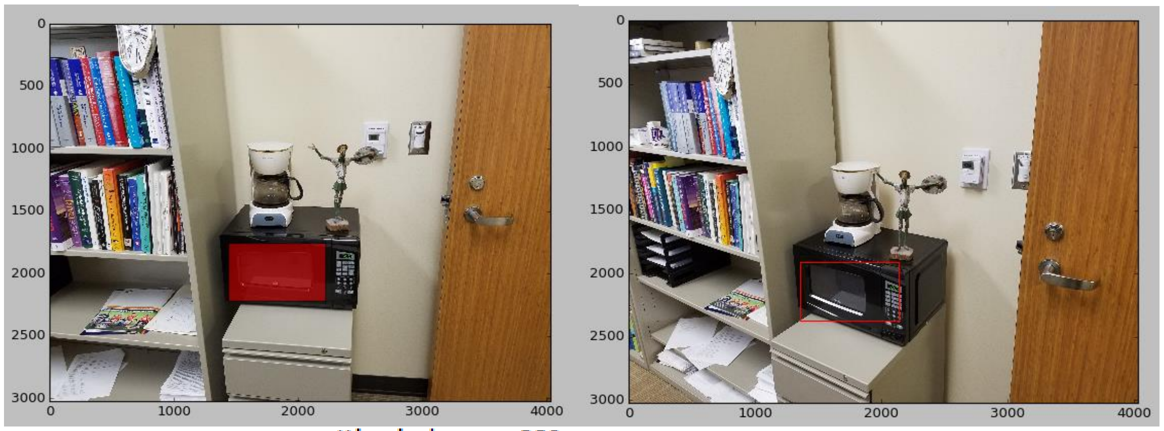


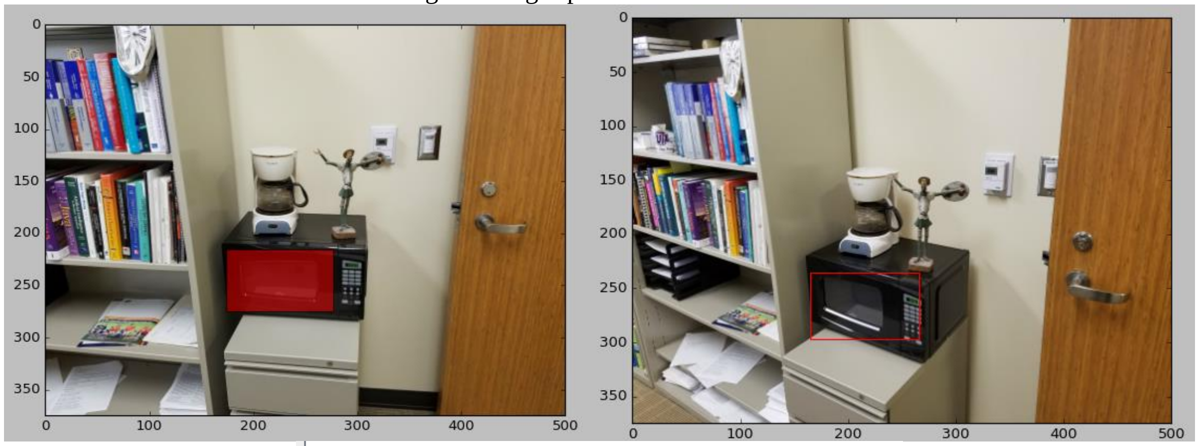
Figure 1: Above images show original image and selection, and pixel-by-pixel detection on a similar image respectively.

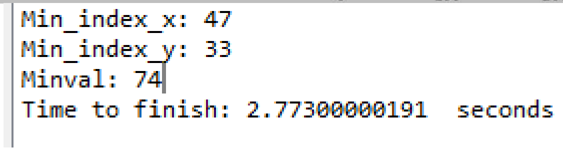


```
Min_index_x: 382
Min_index_y: 271
Minval: 85
Time to finish: 11989.658  seconds
```

Figure 2: Above images show original image and selection from lab's photo library, and pixel-by-pixel detection on a similar image (image 2) respectively. The third image also shows runtime of pixel by pixel in an

image of size 4032 x 3024 pixels. The region selected was of size 778 x 463 pixels. The images used were the first two original images provided in the course website.





```
Min_index_x: 47
Min_index_y: 33
Minval: 74
Time to finish: 2.77300000191  seconds
```

Figure 3: Above images show original image and selection from lab's photo library, and pixel-by-pixel detection on a similar image respectively. The third image also shows runtime of pixel by pixel in an image of size 500 x 375 pixels. The images used were shrunk to facilitate runtime.

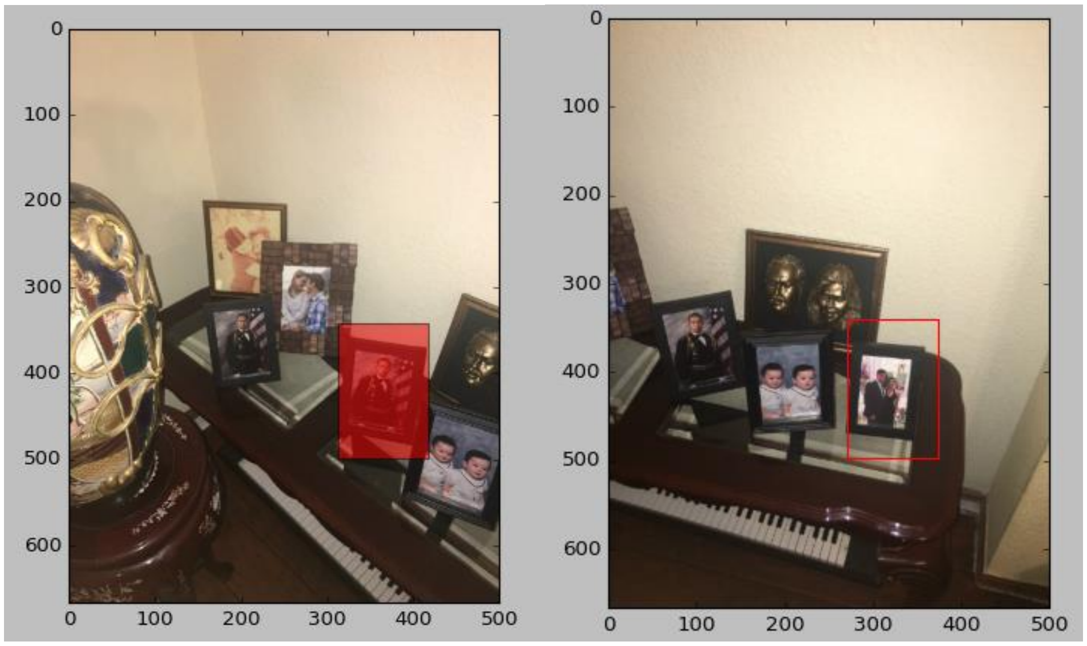
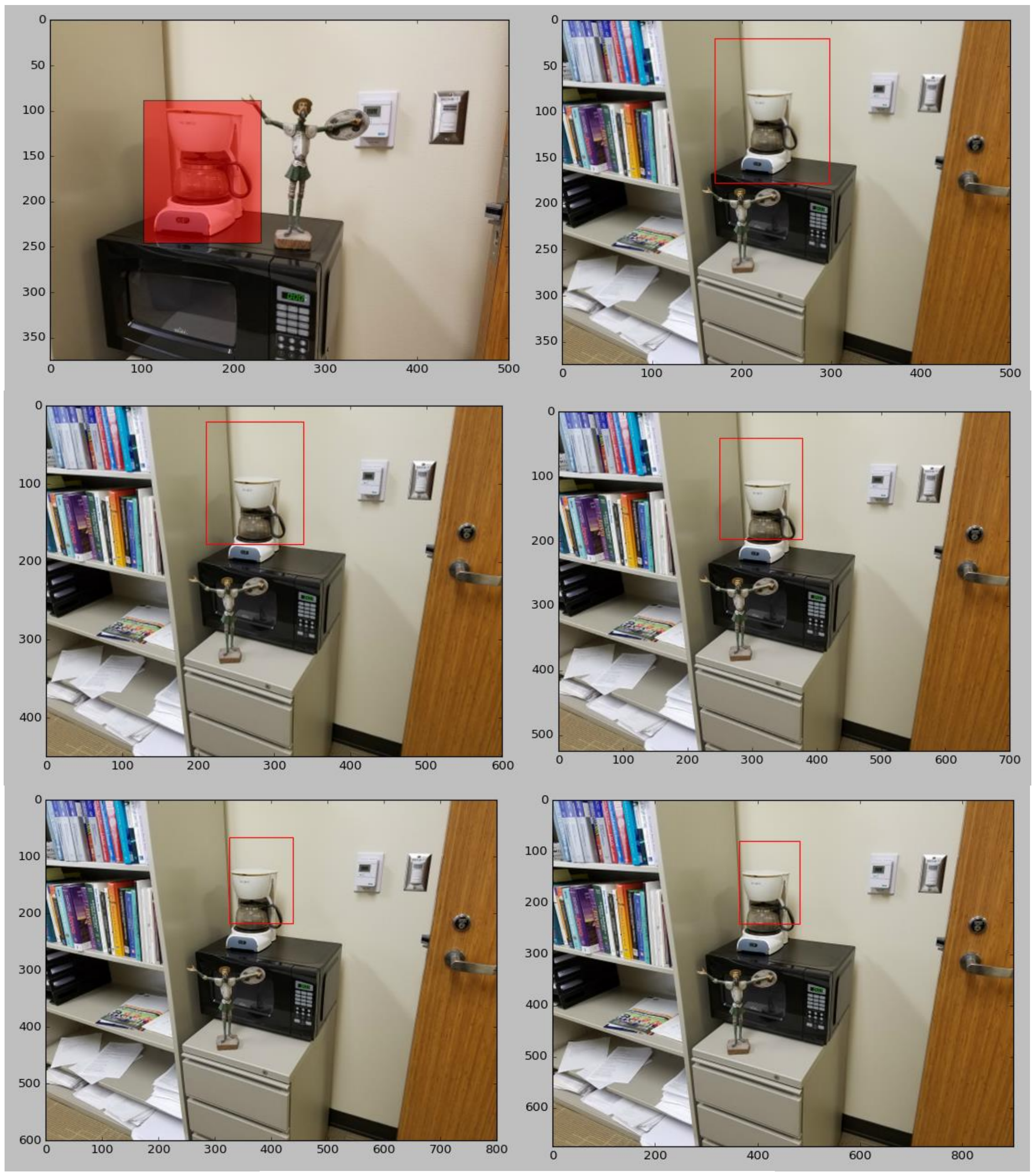
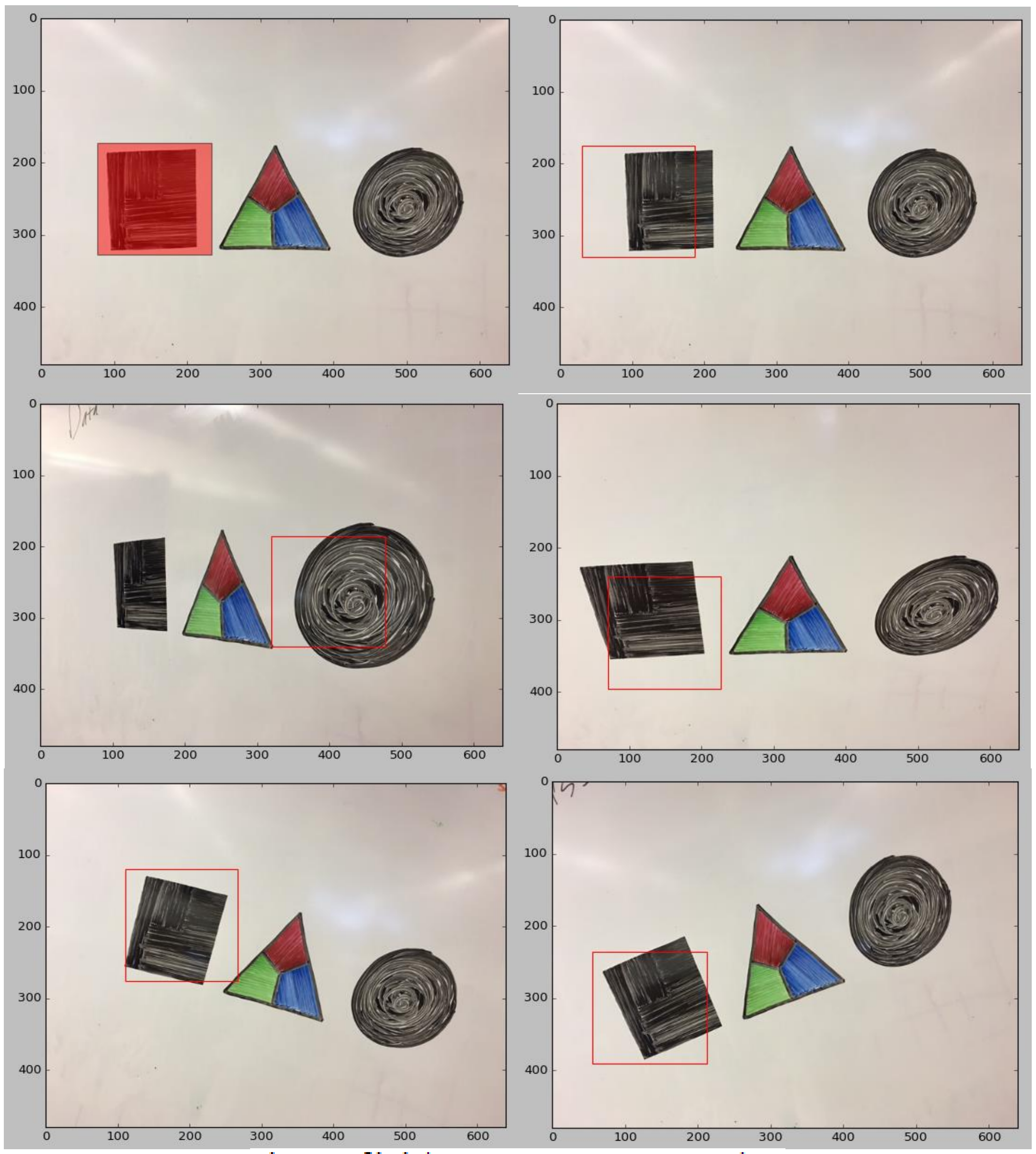


Figure 4: The images above are "8small.jpg" and "10small.jpg". The first image was used to make the selection, the second was used as a comparison.



```
Time to finish: 48.0299999714  seconds
```

Figure 5: From left to right and top to bottom, Original image (3small.jpg) and selection, result of pixel-by-pixel comparison of original second image (4small.jpg), result of pixel-by-pixel with scale 1.2, result of pixel-by-pixel with scaale 1.4, result of pixel-by-pixel with scale 1.6, and result of pixel-by-pixel with scale 1.8, respectively. The runtime for all there comparisons was 48.02 seconds cummulatively.



```
Time to finish: 28.4709999561  seconds
```

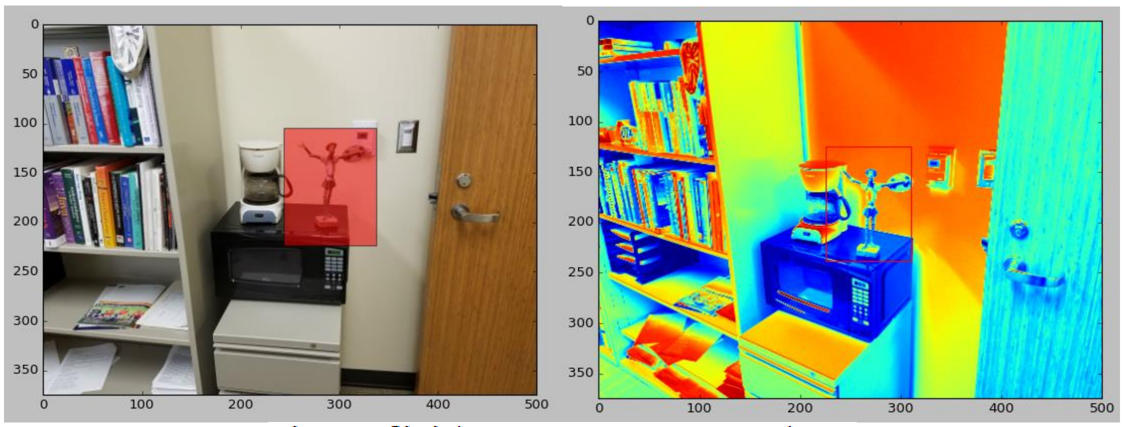
Figure 6: Pixel-by-pixel comparison using “13.jpg” as original and “[13-17].jpg” as comparisons, respectively. The time dispayed is cumulative.

Figure 1 shows a two test images created with different squares. The first image in Figure 1 shows the selection of the area of interest. The second image, slightly different that the first, shows the area returned by pixel-by-pixel comparison. Although not perfect, it can be seen that the area returned is pretty similar to the area originally selected.

As it can be seen in Figure 2, processing images of higher dimensions affect the running time exponentially. The images and selections for Figures 2 and 3 were fairly the same (with regards to selection), and identical with regards to the images used. The only difference was that Figure 3 used images of size 500 x 375 pixels, whereas Figure 2 used images of size 4032 x 3024. That is, roughly 12% or a scale of 1 to 8. The run time, however, increased from 2.77 seconds to 11,989.65 seconds

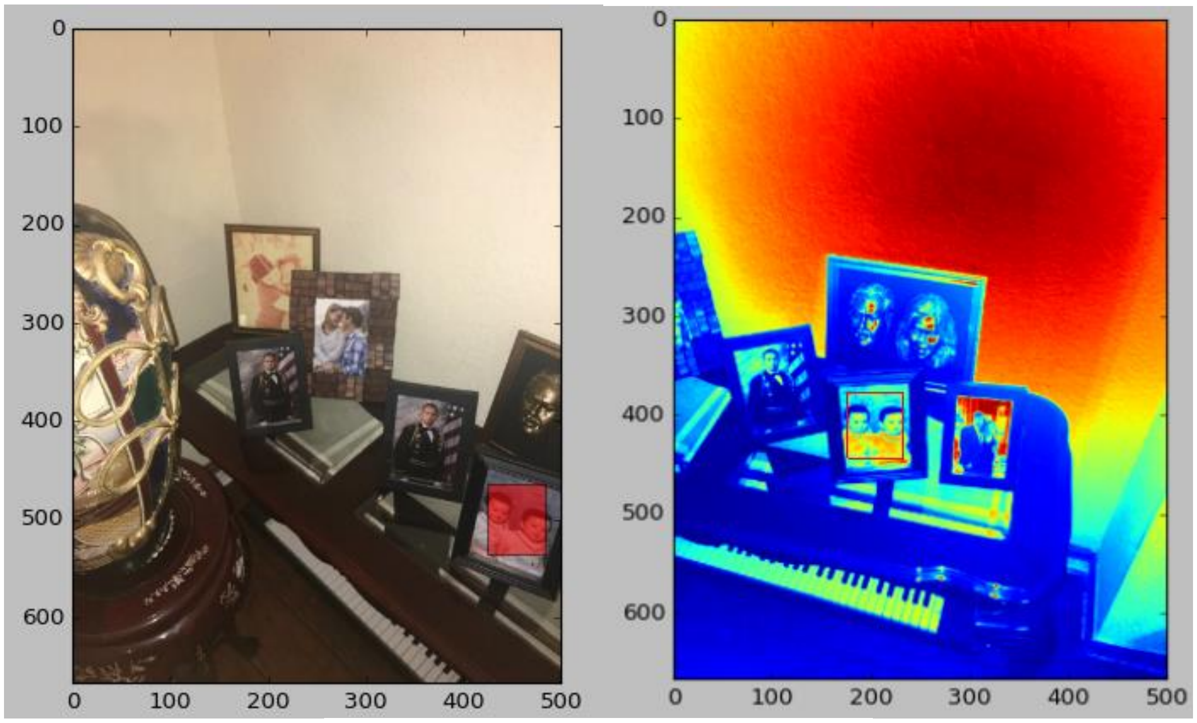
(3.3304 hours). The rest of the images used for the results were smaller, making the runtime of the pixel-by-pixel comparison much faster.
As stated before, Figure 3 shows an identical test run as Figure 2, only with smaller images. The Figure 4 shows an example of one of pixel-by-pixel comparison’s disadvantages. The pixel-by-pixel is dependent on orientation of region of interest. The first image in Figure 4 shows that the selection was a frame slightly tilted clockwise. The second image, used to compare selection, shows the same picture frame, but now slightly tilted counterclockwise. There is another picture frame that matches the angle and size of the area of interest, so pixel-by-pixel comparison returns this second picture frame as the selection.
Figure 5 shows pixel-by-pixel comparison using “3small.jpg” as original, and “4small.jpg” as comparison. The program also uses different ratios of “4small.jpg” as comparisons, specifically *1.2*, *1.4*, *1.6*, and *1.8*. The pixel-by-pixel comparison works in identifying similar regions of interest, but not the exact match. For the runtime of this program, I think the results are pretty good.
Figure 6 shows the pixel-by-pixel comparison of images “[14-17].jpg” using “14.jpg” as the original image. All these images are of size 640 x 480 pixels. Like in Figure 5, pixel-by-pixel comparison returns a region of interest near the actual object. An interesting finding during this test was the second comparison (“14.jpg” v “15.jpg”). Since the square in “15.jpg” was thinner that the selected square of “14.jpg”, pixel-by-pixel returned a region of interest containing the circle, since the area in black of the circle is closer to the original selection, than the area in black of the square. Interestingly enough, this test returned good approximations with a tilted image (“16.jpg” and “17.jpg”).

3.2. P1E2



```
Time to finish: 78.2090001106  seconds
```

Figure 7: From top to bottom: A – Original; B – theta = 1, n = 0.1; C – theta = 300, n = 0.1; D – theta = 300, n = .5



```
Time to finish: 280.950999975  seconds
```

Figure 8: HOG Comparison of “8small.jpg” and “10small.jpg”

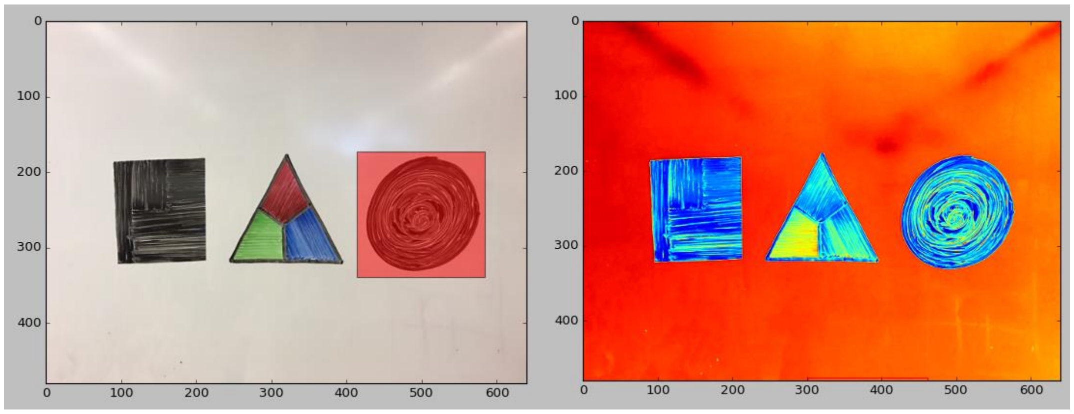


Figure 9: HOG Comparison of “13.jpg” and itself

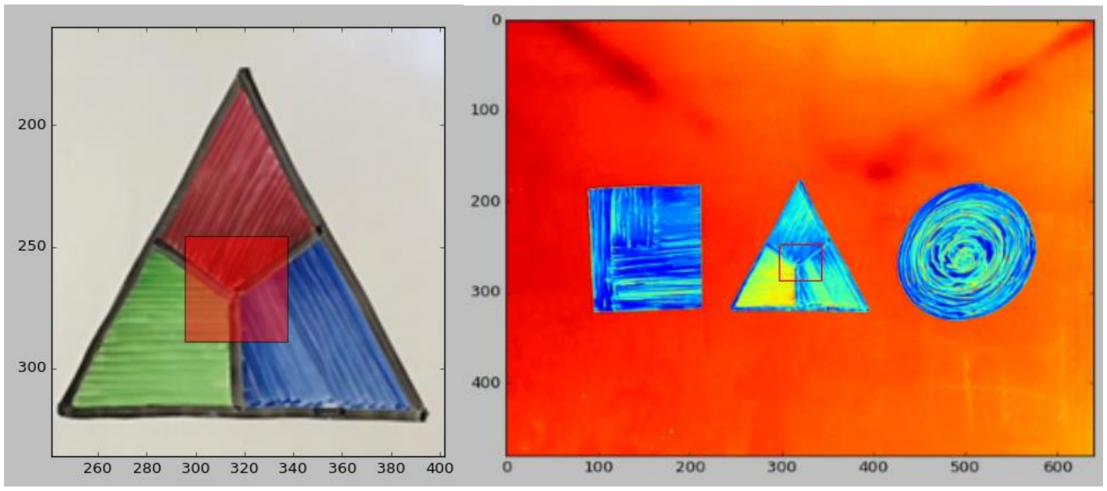
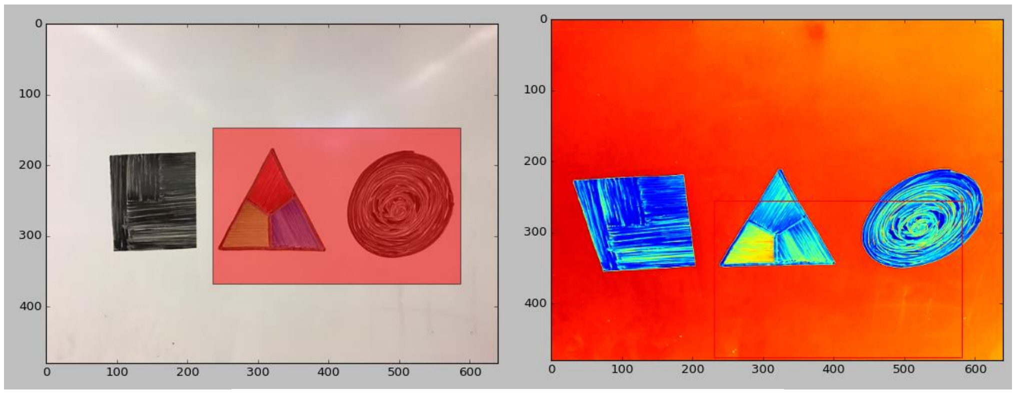
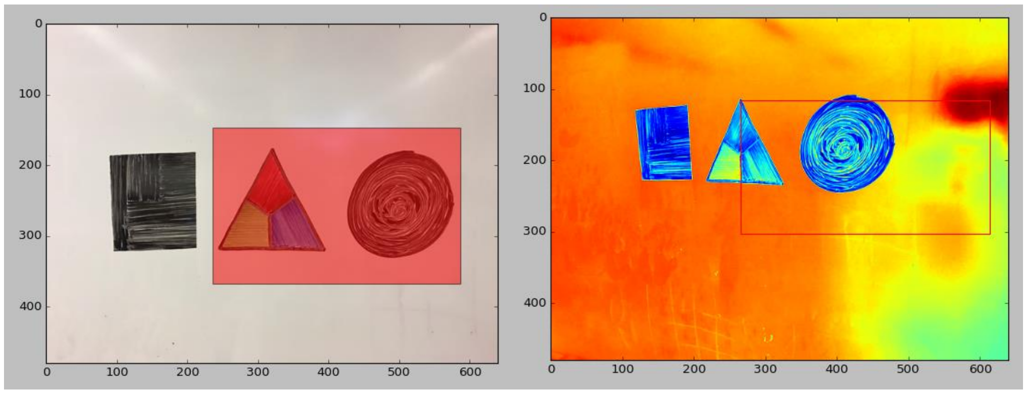


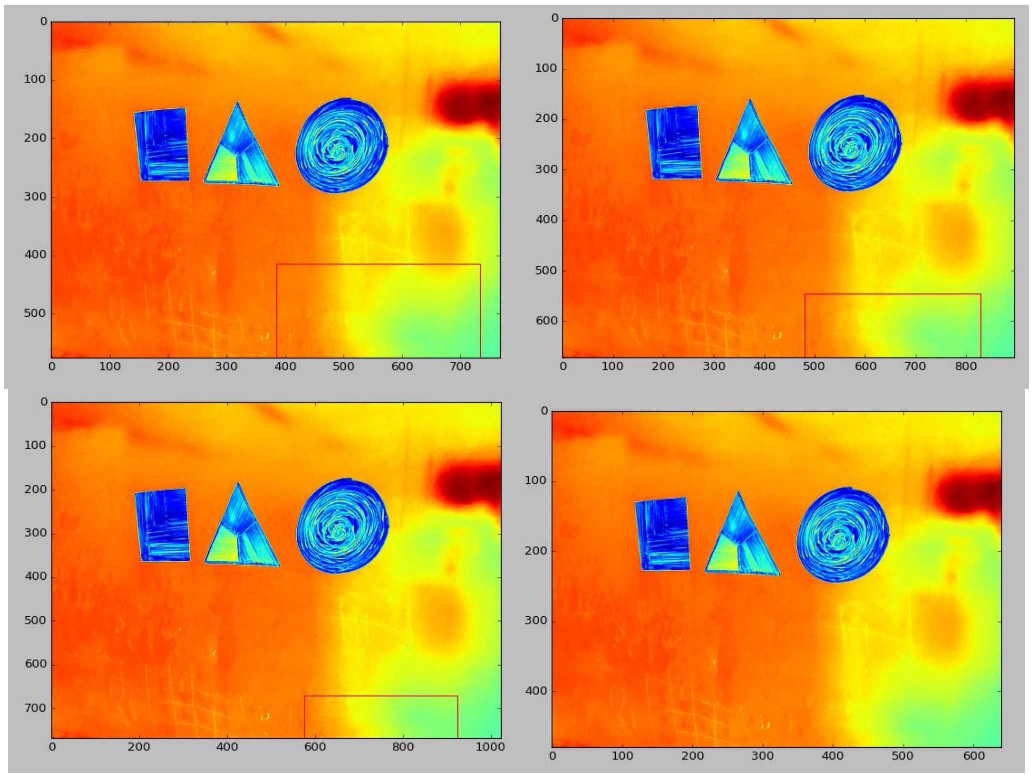
Figure 10: HOG Comparison of “13.jpg” and itself



```
Time to finish: 89.5090000629  seconds
```

Figure 11: HOG Comparison of “13.jpg” and “15.jpg”, and the time for program to finish





```
Time to finish: 35.1429998875  seconds
```

Figure 12: From left to right and top to bottom, Original image (13.jpg) and selection, result of HOG comparison of original second image (18.jpg), result of comparison with scale 1.2, result of comparison with scaale 1.4, result of comparison with scale 1.6, and result of comparison with scale 1.8, respectively. The runtime for all there comparisons was 35.14 seconds

As it can be seen in Figure 7, HOG comparison return a much better approximation on more complex objects like the Quixote than pixel-by-pixel comparison. The HOG Comparison, however, takes much longer to run than pixel-by-pixel, at least on my implementations. Figure 8 supports the claim that HOG comparison can detect items with more detail, with slight changes in orientation like the two faces in the image. My implementation of HOG comparison, however, did not do so well as pixel-by-pixel when detecting general shapes, Figure 9. On the other hand, detecting details, as shown in Figure 10, was accomplished using HOG comparison

As expected from the previous two results, HOG does a somewhat decent job at detecting general figures with details, although not exact as shown in Figure 11. It is able to detect the details, but falls a bit short on the identification of general shapes.

Scaling does not seem to work well with my implementation of HOG comparison as seen in Figure 12. I compared a selection of image “13.jpg” and different scales of image “18.jpg” with bad results.

4. Appendix

4.1. Code

I know I followed some bad coding practices for this lab, however, I was more preoccupied in getting the programs working that making them highly-cohesive, loosely-coupled, and readable. I understand that using global variables is not best coding practice, and goes agains multiple software design principles, but to my defense, the programs run, and somewhat do what they are supposed to do. If time is allotted, I could improve my existing code.

4.1.1. Problem 1

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 14 15:32:33 2016

@author: auort
"""

#Region select

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import matplotlib.widgets as widgets
import scipy.misc
import time
from numpy import random
from scipy.ndimage import filters
from pylab import *
from numpy import *
from copy import copy, deepcopy
import matplotlib.patches as patches



def onselect(eclick, erelease):
    global start_time
    global aX
    global aY
    global bX
    global bY



    if eclick.ydata>erelease.ydata:
        eclick.ydata,erelease.ydata=erelease.ydata,eclick.ydata
    if eclick.xdata>erelease.xdata:
        eclick.xdata,erelease.xdata=erelease.xdata,eclick.xdata
    ax.set_ylim(erelease.ydata,eclick.ydata)
    ax.set_xlim(eclick.xdata,erelease.xdata)
    fig.canvas.draw()

    """
    print 'aX: ',eclick.xdata
    print 'aY: ',eclick.ydata
    print 'bX: ',erelease.xdata
    print 'bY: ',erelease.ydata
    """
    aX = int(eclick.xdata)
    aY = int(eclick.ydata)
    bX = int(erelease.xdata)
    bY = int(erelease.ydata)

    #printGlobals()
    saveSelection()
    img = Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\13.jpg')
    image = np.array(img)
    selection = np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\Selection.jpg'))

    """
    imagescaled1 = getscaledimage(img, 1.2)
    imagescaled2 = getscaledimage(img, 1.4)
    imagescaled3 = getscaledimage(img, 1.6)
    imagescaled4 = getscaledimage(img, 1.8)
    imagescaled5 = getscaledimage(img, 2.0)
    """
    image2 = np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\14.jpg'))
    image3 = np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\15.jpg'))
    image4 = np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\16.jpg'))
    image5 = np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\17.jpg'))

    start_time = time.time()


    xval, yval = runPixelDetection(image, selection)
    drawareaofinterest(image, xval, yval, selection)
    xval, yval = runPixelDetection(image2, selection)
    drawareaofinterest(image2, xval, yval, selection)
    xval, yval = runPixelDetection(image3, selection)
    drawareaofinterest(image3, xval, yval, selection)
    xval, yval = runPixelDetection(image4, selection)
    drawareaofinterest(image4, xval, yval, selection)
    xval, yval = runPixelDetection(image5, selection)
    drawareaofinterest(image5, xval, yval, selection)

    """
    xval, yval = runPixelDetection(image, selection)
    drawareaofinterest(image, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled1, selection)
    drawareaofinterest(imagescaled1, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled2, selection)
    drawareaofinterest(imagescaled2, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled3, selection)
    drawareaofinterest(imagescaled3, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled4, selection)
    drawareaofinterest(imagescaled4, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled5, selection)
    drawareaofinterest(imagescaled5, xval, yval, selection)
    """
    end_time = time.time()
    '''
    drawareaofinterest(image, xval, yval, selection)
    drawareaofinterest(imagescaled1, xval, yval, selection)
    drawareaofinterest(imagescaled2, xval, yval, selection)
    drawareaofinterest(imagescaled3, xval, yval, selection)
    '''
    print " "
    print "Time to finish: %s" %(end_time - start_time), " seconds"
    #scipy.misc.imsave('C:\\Users\\auort\\Desktop\\CV_L2\\PixelbyPixel2.jpg', result)

def getscaledimage (img, scale):
    basewidth = int(img.size[0] * scale)
    wpercent = (basewidth/float(img.size[0]))
    hsize = int((float(img.size[1])*float(wpercent)))
    img = img.resize((basewidth,hsize), Image.ANTIALIAS)
    imagescaled = np.array(img)
    return imagescaled
```

```
def drawareaofinterest(image, xval, yval, selection):
    x = xval*5
    y = yval*5
    #im = np.array(Image.open('stinkbug.png'), dtype=np.uint8)

    # Create figure and axes
    fig,ax = plt.subplots(1)

    # Display the image
    ax.imshow(image)

    # Create a Rectangle patch
    rect = patches.Rectangle((y,x),len(selection[0]),len(selection),linewidth=1,edgecolor='r',facecolor='none')

    # Add the patch to the Axes
    ax.add_patch(rect)

    plt.show()

def printGlobals():
    print 'aX: ',aX
    print 'aY: ',aY
    print 'bX: ',bX
    print 'bY: ',bY

def saveSelection():
    #print "saveSelection called!"
    global image
    selection = image[aY:bY,:][:,aX:bX]
    scipy.misc.imsave('C:\\Users\\auort\\Desktop\\CV_L2\\Selection.jpg', selection)

def runPixelDetection(image, selection):
    #print "runPixelDetection called!"
    heightindex = 0
    widthindex = 0

    result = initializezeroarray((len(image)),(len(image[1])))
    imageheight = len(image[1])

    heightlimit = (len(image)-len(selection))
    widthlimit = (len(image[0])-len(selection[0]))
    imagewidth = len(image)
    imageheight = len(image[0])

    selectheight = len(selection)
    selectwidth = len(selection[0])

    resultmatrixwidth = int(imagewidth/5)
    resultmatrixheight = int(imageheight/5)

    resultmatrix = initializezeroarray (resultmatrixwidth, resultmatrixheight)
    minval = 160;
    minvalx = 0;
    minvaly = 0;
    while (heightindex < heightlimit):#height traversal
        widthindex = 0
        while (widthindex < widthlimit):#width traversal

            compare = (image[widthindex:(widthindex+len(selection)),:][:,heightindex:(heightindex+len(selection[0]))])
            if (len(compare)!= len(selection)):
                compare = deepcopy(selection)
                compare[compare > 0] = 0
            if (len(compare[0])!= len(selection[0])):
                compare = deepcopy(selection)
                compare[compare > 0] = 0

            #compare2 = selection[0:len(compare),:][:,0:len(compare[0])]

            subtraction = (compare) -(selection)

            subtraction[subtraction < 0] = 0

            avgsum = cumsum(cumsum(subtraction, 1), 0)#/(len(selection)*len(selection[1]))
            avgsum = avgsum.max(axis = 1)
            avgsum = avgsum.max(axis = 0)
            avgsum = avgsum.max()/(len(selection)*len(selection[1]))

            result[heightindex][widthindex] = avgsum

            resultmatrixheightindex = int(heightindex/5)
            resultmatrixwidthindex = int(widthindex/5)

            if(resultmatrixheightindex < len(resultmatrix)):
                #print "inside check for result matrix, resultmatrixheightindex < resultmatrix"
                if (resultmatrixwidthindex < len(resultmatrix[0])):
                    #print "inside check for result matrix, resultmatrixheightindex < resultmatrix[0]"
                    resultmatrix[resultmatrixheightindex][resultmatrixwidthindex] = avgsum
                    #print "        minval: %s" %minval
                    #print "        avgsum: %s" %avgsum
                    if (minval >= avgsum):
                        #print "                minval checking to update minval indexes"
                        minval = avgsum
                        minvalx = resultmatrixwidthindex
                        minvaly = resultmatrixheightindex

            widthindex = widthindex + 5

        heightindex = heightindex + 5
        if(widthindex >= len(image[0]) + 1):
            print "Caught out of bounds"
            break

    return minvalx, minvaly

def initializezeroarray(w, h):
    #print "initializezeroarray called!"
    zeroarray = [[0 for x in range(h)] for y in range(w)]
    return zeroarray

def getSelectionWidth():
    print "getSelectionWidth called!"
    return (bX-aX)

def getSelectionHeight():
    print "getSelectionHeight called!"
    return (bY-aY)

identifier = 1
stringidentifier = str(identifier)
fig = plt.figure()
ax = fig.add_subplot(111)
directory = 'C:\\Users\\auort\\Desktop\\CV_L2\\'
im = Image.open(directory + '13.jpg')
image = np.array(im)
```

```
aX = 0
aY = 0
bX = 0
bY = 0
start_time = 0

arr = np.asarray(im)
plt_image=plt.imshow(arr)
rs=widgets.RectangleSelector(
    ax, onselect, drawtype='box',
    rectprops = dict(facecolor='red', edgecolor = 'black', alpha=0.5, fill=True))
plt.show()
```

4.1.2. Problem 2

```
# -*- coding: utf-8 -*-
"""
Created on Wed Sep 14 15:32:33 2016

@author: auort
"""

#Region select

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import matplotlib.widgets as widgets
import scipy.misc
import time
from numpy import random
from scipy.ndimage import filters
from pylab import *
from numpy import *
from copy import copy, deepcopy
import matplotlib.patches as patches
from pylab import *
import scipy.misc



def onselect(eclick, erelease):
    global start_time
    global aX
    global aY
    global bX
    global bY



    if eclick.ydata>erelease.ydata:
        eclick.ydata,erelease.ydata=erelease.ydata,eclick.ydata
    if eclick.xdata>erelease.xdata:
        eclick.xdata,erelease.xdata=erelease.xdata,eclick.xdata
    ax.set_ylim(erelease.ydata,eclick.ydata)
    ax.set_xlim(eclick.xdata,erelease.xdata)
    fig.canvas.draw()

    """
    print 'aX: ',eclick.xdata
    print 'aY: ',eclick.ydata
    print 'bX: ',erelease.xdata
    print 'bY: ',erelease.ydata
    """
    aX = int(eclick.xdata)
    aY = int(eclick.ydata)
    bX = int(erelease.xdata)
    bY = int(erelease.ydata)

    printGlobals()
    saveSelection()
    img = Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\18.jpg').convert('L')
    image = np.array(img)
    selection                                                                    =
np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\Selection.jpg').convert('L'))

    imagescaled1 = getscaledimage(img, 1.2)
    imagescaled2 = getscaledimage(img, 1.4)
    imagescaled3 = getscaledimage(img, 1.6)
    imagescaled4 = getscaledimage(img, 1.8)

    """
    image2                                                                       =
np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\14.jpg').convert('L'))
    image3                                                                       =
np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\15.jpg').convert('L'))
    image4                                                                       =
np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\16.jpg').convert('L'))
    image5                                                                       =
np.array(Image.open('C:\\Users\\auort\\Desktop\\CV_L2\\17.jpg').convert('L'))
    """
    start_time = time.time()


    xval, yval = runPixelDetection(image, selection)
    drawareaofinterest(image, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled1, selection)
    drawareaofinterest(imagescaled1, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled2, selection)
    drawareaofinterest(imagescaled2, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled3, selection)
    drawareaofinterest(imagescaled3, xval, yval, selection)
    xval, yval = runPixelDetection(imagescaled4, selection)
    drawareaofinterest(imagescaled4, xval, yval, selection)

    '''
    xval, yval = runHOGDetection(image, selection)
    drawareaofinterest(image, xval, yval, selection)
    '''
    '''
    xval, yval = runHOGDetection(image2, selection)
    drawareaofinterest(image2, xval, yval, selection)
    xval, yval = runHOGDetection(image3, selection)
    drawareaofinterest(image3, xval, yval, selection)
    xval, yval = runHOGDetection(image4, selection)
    drawareaofinterest(image4, xval, yval, selection)
    xval, yval = runHOGDetection(image5, selection)
    drawareaofinterest(image5, xval, yval, selection)
    '''



    end_time = time.time()
    drawareaofinterest(image, xval, yval, selection)
    print " "
    print "Time to finish: %s" %(end_time - start_time), " seconds"
    #scipy.misc.imsave('C:\\Users\\auort\\Desktop\\CV_L2\\PixelbyPixel2.jpg', result)

def getscaledimage (img, scale):
    basewidth = int(img.size[0] * scale)
    wpercent = (basewidth/float(img.size[0]))
    hsize = int((float(img.size[1])*float(wpercent)))
    img = img.resize((basewidth,hsize), Image.ANTIALIAS)
    imagescaled = np.array(img)
    return imagescaled

def HOGGen(image, ax, ay, bx, by, barnumber):
    regionheight = by-ay
    regionwidth = bx-ax

    subregionheight = int(regionheight/2)
    subregionwidth = int(regionwidth/2)

    #The region will be separated into four quadrant subregions
    and an overlapping center subregion
    subregionA = image[ay:ay+subregionheight,:][:,ax:ax+subregionwidth
    subregionB = image[by-subregionheight:by,:][:,ax:ax+subregionwidth]
    subregionC = image[ay:ay+subregionheight,:][:,bx-subregionwidth:bx]
    subregionD = image[by-subregionheight:by,:][:,bx-subregionwidth:bx]

    histogram = getGradient(image)
    histogramA = getGradient(subregionA)
    histogramB = getGradient(subregionB)
    histogramC = getGradient(subregionC)
    histogramD = getGradient(subregionD)

    return histogram, histogramA, histogramB, histogramC, histogramD




def getGradient (im):

    gx, gy = gradient(im)
    rad = arctan2(gy, gx)
    variableA = 180/math.pi
    deg = (rad*(variableA))
    print deg.flatten()
    return deg.flatten()

def drawareaofinterest(image, xval, yval, selection):
    x = xval*5
    y = yval*5

    # Create figure and axes
    fig,ax = plt.subplots(1)

    # Display the image
    ax.imshow(image)

    # Create a Rectangle patch
    rect = patches.Rectangle((y,x),len(selection[0]),
    len(selection),linewidth=1,
    edgecolor='r',facecolor='none')

    # Add the patch to the Axes
    ax.add_patch(rect)
    plt.show()

def printGlobals():
    print 'aX: ',aX
    print 'aY: ',aY
    print 'bX: ',bX
    print 'bY: ',bY

def saveSelection():
    print "saveSelection called!"
    global image
    selection = image[aY:bY,:][:,aX:bX]
    scipy.misc.imsave('C:\\Users\\auort\\Desktop\\CV_L2\\Selection.jpg', selection)

def runHOGDetection(image, selection):
    print "runPixelDetection called!"
    heightindex = 0
    widthindex = 0

    imageheight = len(image[1])
    heightlimit = (len(image)-len(selection))
    widthlimit = (len(image[0])-len(selection[0]))
    imagewidth = len(image)
    imageheight = len(image[0])
    selectheight = len(selection)
    selectwidth = len(selection[0])

    resultmatrixwidth = int(imagewidth/5)
    resultmatrixheight = int(imageheight/5)

    resultmatrix = initializezeroarray (resultmatrixwidth, resultmatrixheight)
    minval = 160;
    minvalx = 0;
    minvaly = 0;

    HO, HA, HB, HC, HD = HOGGen(selection, 0, 0, len(selection)-1,
    len(selection[0])-1, 10)

    while (heightindex < heightlimit):#height traversal
        widthindex = 0
        while (widthindex < widthlimit):#width traversal
            print "widthindex: %s." % widthindex
            print "widthlimit: %s." % widthlimit
            print "heightindex: %s." % heightindex
            print "heightlimit: %s." % heightlimit

            bx = selectwidth + widthindex
            by = selectheight + heightindex
            HO2, HA2, HB2, HC2, HD2 = HOGGen(image, widthindex, heightindex,
            bx, by, 10)

            if(len(HA2) != len(HA)):
                HA2 = deepcopy(HA)
                HA2[HA2 > 0] = 0
            if(len(HB2) != len(HB)):
                HB2 = deepcopy(HB)
                HB2[HB2 > 0] = 0
            if(len(HC2) != len(HC)):
                HC2 = deepcopy(HC)
                HC2[HC2 > 0] = 0
            if(len(HD2) != len(HD)):
                HD2 = deepcopy(HD)
                HD2[HD2 > 0] = 0
            #subO = (HO) - (HO2)
            subA = (HA) - (HA2)
            subB = (HB) - (HB2)
            subC = (HC) - (HC2)
            subD = (HD) - (HD2)

            #making sure subs only have positive values
            #subO[subO < 0] = 0
            subA[subA < 0] = 0
            subB[subB < 0] = 0
            subC[subC < 0] = 0
            subD[subD < 0] = 0


            avgsumA = cumsum(subA, 0)#/(len(selection)*len(selection[1]))
            avgsumA = avgsumA.max(axis = 0)
            avgsumA = avgsumA.max()/len(subA)

            avgsumB = cumsum(subB, 0)#/(len(selection)*len(selection[1]))
            avgsumB = avgsumB.max(axis = 0)
            avgsumB = avgsumB.max()/len(subB)

            avgsumC = cumsum(subC, 0)#/(len(selection)*len(selection[1]))
            avgsumC = avgsumC.max(axis = 0)
            avgsumC = avgsumC.max()/len(subC)

            avgsumD = cumsum(subD, 0)#/(len(selection)*len(selection[1]))
            avgsumD = avgsumD.max(axis = 0)
            avgsumD = avgsumD.max()/len(subD)

            avgsum = avgsumA + avgsumB + avgsumC + avgsumD / 4
            npixsubregions = len(selection/4)*len(selection[1]/4)

            resultmatrixheightindex = int(heightindex/5)
            resultmatrixwidthindex = int(widthindex/5)

            if(resultmatrixheightindex < len(resultmatrix)):
                #print "inside check for result matrix,
                resultmatrixheightindex < resultmatrix"
                if (resultmatrixwidthindex < len(resultmatrix[0])):
                    #print "inside check for result matrix,
                    resultmatrixheightindex < resultmatrix[0]"
                    resultmatrix[resultmatrixheightindex]
                    [resultmatrixwidthindex] = avgsum

                    if (minval >= avgsum):
                        minval = avgsum
                        minvalx = resultmatrixwidthindex
                        minvaly = resultmatrixheightindex

            widthindex = widthindex + 5

        heightindex = heightindex + 5
        if(widthindex >= len(image[0]) + 1):
            print "Caught out of bounds"
            break

    return minvalx, minvaly

def initializezeroarray(w, h):
    print "initializezeroarray called!"
    zeroarray = [[0 for x in range(h)] for y in range(w)]
    return zeroarray

def getSelectionWidth():
    print "getSelectionWidth called!"
    return (bX-aX)

def getSelectionHeight():
    print "getSelectionHeight called!"
    return (bY-aY)

identifier = 1
stringidentifier = str(identifier)
fig = plt.figure()
ax = fig.add_subplot(111)
directory = 'C:\\Users\\auort\\Desktop\\CV_L2\\'
im = Image.open(directory + '13.jpg')
image = np.array(im)
aX = 0
aY = 0
bX = 0
bY = 0
start_time = 0

arr = np.asarray(im)
plt_image=plt.imshow(arr)
rs=widgets.RectangleSelector(
    ax, onselect, drawtype='box',
    rectprops = dict(facecolor='red', edgecolor = 'black', alpha=0.5, fill=True))
plt.show()
```

End of Document

End of Document