**CSC 481**

**Assignment 5**

**Hithesh Shanmugam**

**Problem 1: Edge Detection with Color (10/10)**

Choose a color image. Convert it to grayscale using an average of all three-color channels and find the edges (anyway you like). Then, convert the original image to HSI, and find edges on the I component using the same method you used for the grayscale image. Finally, find edges on the H component using the same method. Compare the three edge images you found and discuss similarities and/or differences that you notice. Make sure to include in your comparison which method gave you the best results and why you think so.

#Import packages

import matplotlib.pyplot as plt

import cv2

import numpy as np

from skimage import filters, color,io

from skimage.io import imshow, imread

from skimage.color import rgb2hsv, hsv2rgb

#Load and display image

path\_of\_img = r'C:/Users/sures/OneDrive - DePaul University/Desktop/balloon.jpg'

image = cv2.imread(path\_of\_img)

#Convert to rgb

image\_process = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

#Plot image

fig = plt.figure()

plt.imshow(image\_process)

fig.set\_size\_inches(10, 10)

plt.xticks([]), plt.yticks([])

plt.show()

**Output:**

****

#Convert to greyscale

grey = color.rgb2gray(io.imread(path\_of\_img))

#Plot

fig = plt.figure()

plt.imshow(grey)

fig.set\_size\_inches(10, 10)

plt.xticks([]), plt.yticks([])

plt.show()

**Output:**



#Detect edges with sobel

image\_sobel= filters.sobel(grey)

fig = plt.figure()

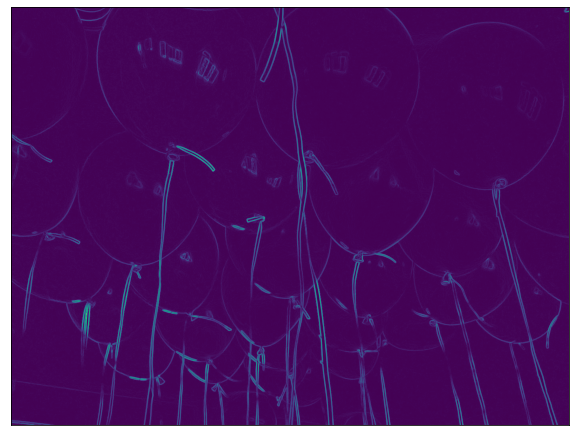
plt.imshow(image\_sobel)

fig.set\_size\_inches(10, 10)

plt.xticks([]), plt.yticks([])

plt.show()

**Output:**

****

#HSI

HSI\_image = cv2.cvtColor(image\_process, cv2.COLOR\_RGB2HLS)

#Split channels

h, s, i = cv2.split(HSI\_image)

#Edges for intensity component

sobel\_intensity = filters.sobel(i)

#Edges for hue component

sobel\_hue = filters.sobel(h)

#Plotting edge images

rows = 2

cols = 2

axes = []

fig = plt.figure()

images = [image\_sobel, sobel\_intensity, sobel\_hue]

edges = ['Sobel-Average-Grayscale', 'Sobel-HSI-Intensity', 'Sobel-HSI-Hue']

for i in range((rows \* cols) - 1):

img\_plot = images[i]

fig.set\_size\_inches(20, 20)

axes.append(fig.add\_subplot(rows, cols, i + 1))

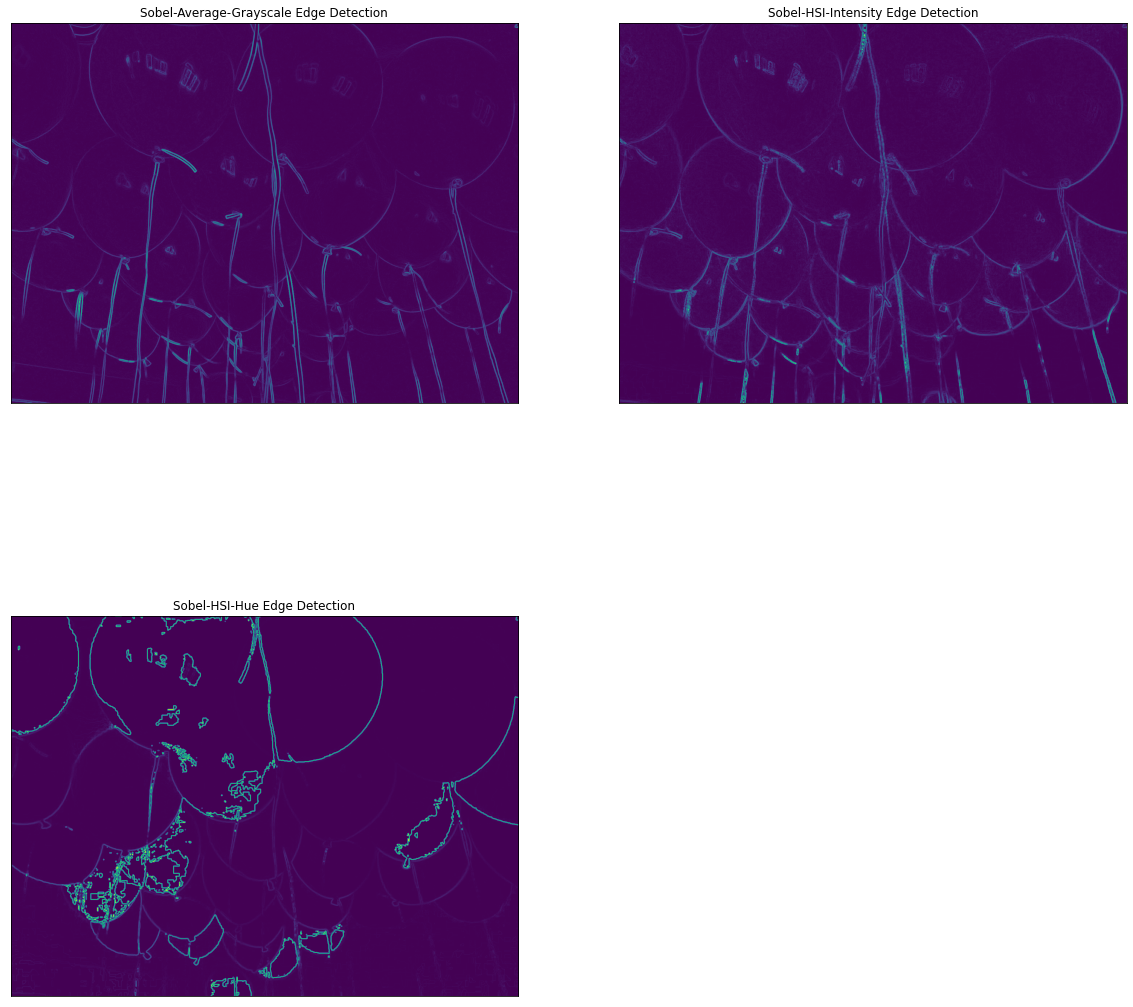
subplot\_title = ("{0} Edge Detection".format(edges[i]))

axes[-1].set\_title(subplot\_title)

plt.xticks([]), plt.yticks([])

plt.imshow(img\_plot)

**Output:**

****

By comparing the three methods the Sobel with intensity in the HSI makes the best edge detection in the input image. The Hue detects the unwanted edges in the balloon like inside the balloon it detects the edge as we can see from the image.

**Problem 2: Color Segmentation**

A natural cue to use in segmenting objects from their surroundings in images is color. **This problem contrasts segmenting color regions using red, green, and blue thresholds** (aligned with the RGB axes of color space) **with segmentation using hue, saturation, and intensity bounds** (aligned with the coordinate system of HSI or HSV space). For this assignment, it will be more educational to choose an image with strongly colored objects. For example, an image of party balloons works well – make sure they are on a dark or light background.

**a) (10/10)** First, segment your image into objects and background using a threshold on the intensity of the pixels. You can get a grayscale image from an RGB image simply by averaging the three color components of each pixel. Demonstrate your segmentation by replacing the background pixels with a visually distinct color. (In fact, just the reverse -- replacing blue or green pixels with those of some preset image -- is the technique used in TV or movies to superimpose objects against some preset background. Since thresholding is used, this (and not fashion) is why so few weathercasters wear saturated blue items. This technique is called travelling matte. Blue is good because it turns black under a red filter; green is good because most digital cameras have less noise in the green channel. Matte techniques have become even more sophisticated as digital video becomes more sophisticated).

#Blur image

blur\_img = filters.gaussian(grey, sigma = 2)

#Create mask

mask = blur\_img < 0.5

#Plot mask

fig = plt.figure()

plt.imshow(mask)

fig.set\_size\_inches(10, 10)

plt.xticks([]), plt.yticks([])

plt.show()

**Output:**

****

b) **(10/10)** Second, use thresholds in each RGB color band to isolate the objects in your image. Again, display the results by "bluing out" the intended object region. Provide some commentary on how the segmentation succeeded and failed. (**481 Students (5)**: use an automatic thresholding approach instead of choosing thresholds by hand. Hint: look at otsuthresh.)

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

plt.imshow(image)

plt.show()

**Output:**

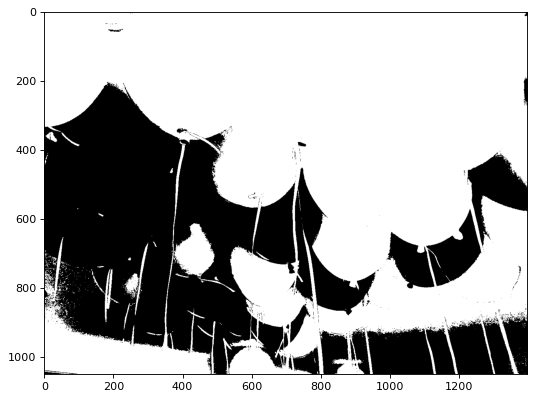
****

red\_filtered\_image = (image[:,:,0] > 150)

plt.figure(num=None, figsize=(8, 6), dpi=80)

plt.imshow(red\_filtered\_image, cmap = 'gray')

**Output:**

****

red\_image\_new = image.copy()

red\_image\_new[:, :, 0] = red\_image\_new[:, :, 0]\*red\_filtered\_image

red\_image\_new[:, :, 1] = red\_image\_new[:, :, 1]\*red\_filtered\_image

red\_image\_new[:, :, 2] = red\_image\_new[:, :, 2]\*red\_filtered\_image

plt.figure(num=None, figsize=(8, 6), dpi=80)

plt.imshow(red\_image\_new)

**Output:**

****

def rgb\_splitter(image1):

rgb\_list = ['Reds','Greens','Blues']

fig, ax = plt.subplots(1, 3, figsize=(15,5), sharey = True)

for i in range(3):

ax[i].imshow(image1[:,:,i], cmap = rgb\_list[i])

ax[i].set\_title(rgb\_list[i], fontsize = 15)

rgb\_splitter(image)

**Output:**

****

red\_filtered = (image[:,:,0] > 150) & (image[:,:,1] < 100) & (image[:,:,2] < 110)

plt.figure(num=None, figsize=(8, 6), dpi=80)

red\_image\_new = image.copy()

red\_image\_new[:, :, 0] = red\_image\_new[:, :, 0] \* red\_filtered

red\_image\_new[:, :, 1] = red\_image\_new[:, :, 1] \* red\_filtered

red\_image\_new[:, :, 2] = red\_image\_new[:, :, 2] \* red\_filtered

plt.imshow(red\_image\_new)

**Output:**

****

**Explanation:**

By canceling the blue and green channel using appropriate thresholds the red channel is shown as above image where we can see the images with red pixels in it like the colors pink and orange are also visible in this.

from matplotlib import pyplot as plt

img = cv2.imread('C:/Users/sures/OneDrive - DePaul University/Desktop/balloon.jpg',0)

img = cv2.medianBlur(img,5)

ret,gloT = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)

mean = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C,cv2.THRESH\_BINARY,11,2)

gaus = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,cv2.THRESH\_BINARY,11,2)

names = ['Original', 'Global','Mean', 'Gaussian']

im = [img, gloT, mean, gaus]

for i in range(4):

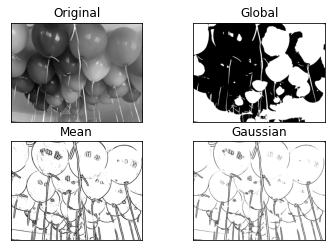
plt.subplot(2,2,i+1),plt.imshow(im[i],'gray')

plt.title(names[i])

plt.xticks([]),plt.yticks([])

plt.show()

**Output:**

****

**Explanation:**

The grayscale thresholding with the automatic filtering is less sensitive to individual thresholding in the RGB channels. The mean thresholding appears to be more effective.

c) **(10/10)** Repeat the segmentation using thresholds of hue in HSI space. MatLab has a function for converting from RGB to HSI (MatLab calls it HSV):

B = rgb2hsv(A);

Note: When described in matlab's HSV space, the hue (first component) of a pixel ranges from 0.0 (red) to 1.0 (red again), passing through orange, yellow, green, cyan, blue, purple, and magenta along the way. The second component, saturation, varies from 0.0 (grayscale) to 1.0 (completely saturated -- no white at all). The final component, intensity (or "value"), also ranges from 0.0 (no intensity) to 1.0 (max intensity).

There is also an inverse function

A = hsv2rgb(B);

It returns an RGB image with pixel components between 0.0 and 1.0

Note: A "threshold" of the hue component of pixels must be an interval, because the hue actually wraps around and is best envisioned as a circle. Thus, to segment a blue region, you need to accept only hues around 2/3 (0 = red, 1/3 = green, 2/3 = blue).

How does your segmentation based on hue differ from your segmentations based on RGB?

def display\_as\_hsv(image1):

img = cv2.imread(image1)

img\_hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

hsv\_list = ['Hue','Saturation','Value']

fig, ax = plt.subplots(1, 3, figsize=(15,7), sharey = True)

red\_girl\_hsv = ax[0].imshow(img\_hsv[:,:,0], cmap = 'hsv')

ax[0].set\_title(hsv\_list[0], fontsize = 20)

ax[0].axis('off')

ax[1].imshow(img\_hsv[:,:,1], cmap = 'Greys')

ax[1].set\_title(hsv\_list[1], fontsize = 20)

ax[1].axis('off')

ax[2].imshow(img\_hsv[:,:,2], cmap = 'gray')

ax[2].set\_title(hsv\_list[2], fontsize = 20)

ax[2].axis('off')

fig.tight\_layout()

display\_as\_hsv('C:/Users/sures/OneDrive - DePaul University/Desktop/balloon.jpg')

**Output:**

****

bal\_path = imread('C:/Users/sures/OneDrive - DePaul University/Desktop/balloon.jpg')

bal\_path\_hsv = rgb2hsv(bal\_path)

image\_hsv = rgb2hsv(image)

plt.figure(num=None, figsize=(8, 6), dpi=80)

plt.imshow(image\_hsv[:,:,0], cmap = 'hsv')

plt.colorbar();

**Output:**

****

lower\_mask = image\_hsv [:,:,0] > 0.90

upper\_mask = image\_hsv [:,:,0] < 1.00

mask = upper\_mask\*lower\_mask

red = image[:,:,0]\*mask

green = image[:,:,1]\*mask

blue = image[:,:,2]\*mask

image\_masked = np.dstack((red,green,blue))

plt.figure(num=None, figsize=(8, 6), dpi=80)

imshow(image\_masked);

**Output:**

****

lower\_mask = image\_hsv [:,:,0] > 0.90

upper\_mask = image\_hsv [:,:,0] < 1.00

saturation = image\_hsv [:,:,1] > 0.50

mask = upper\_mask\*lower\_mask\*saturation

red = image[:,:,0]\*mask

green = image[:,:,1]\*mask

blue = image[:,:,2]\*mask

image\_masked = np.dstack((red,green,blue))

plt.figure(num=None, figsize=(8, 6), dpi=80)

imshow(image\_masked);

**Output:**

****

**Explanation:**

With the HSV split the orange pixels are almost deleted but the pink pixels are as much as same as the rgb.

import numpy as np

import matplotlib.pyplot as plt

import cv2 as cv

import math

def img\_show(img):

img = cv.cvtColor(img, cv.COLOR\_BGR2RGB) # we use open cv to read the image

plt.figure(figsize=(10,7))

plt.axis('off')

plt.imshow(img, cmap = 'gray')

plt.show()

img\_path = 'C:/Users/sures/OneDrive - DePaul University/Desktop/balloon.jpg'

img = cv.imread(img\_path)

def adaptive\_mean\_thresholding(img,B,C):

gray = cv.cvtColor(img, cv.COLOR\_BGR2GRAY )

thresholded\_img = np.zeros((gray.shape)).astype(np.uint8)

H = np.arange(0,gray.shape[0],B)

H = np.append(H, gray.shape[0])

W = np.arange(0,gray.shape[1],B)

W = np.append(W, gray.shape[1])

for i in range(len(H)-1):

for j in range(len(W)-1):

block = gray[H[i]:H[i+1],W[j]:W[j+1]]

threshold = np.mean(block) - C

th, block = cv.threshold(block, threshold, 255, cv.THRESH\_BINARY)

thresholded\_img[H[i]:H[i+1],W[j]:W[j+1]] = block

img\_show(gray)

img\_show(thresholded\_img)

# result

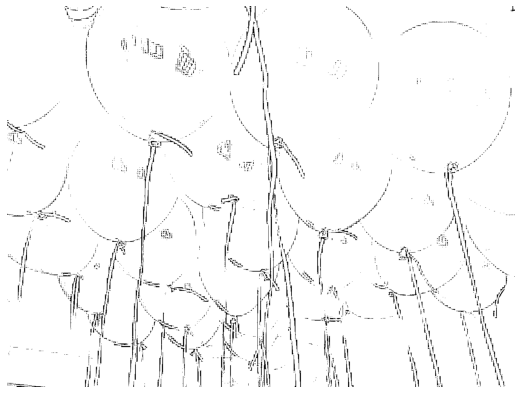
B = 5

C = 10

adaptive\_mean\_thresholding(img,B,C)

**Output:**

****

****

**Explanation:**

The HSV was not supported by the open cv in python so I created a function for automatic thresholding, the function is a mean thresholding and we can clearly see the output.