CSE 344 Computer Vision Assignment 2

Name: Vaha Vahini
Roll Number: 2018302

In [1]:
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
%matplotlib inline
import math
import sklearn as sk
from sklearn import cluster, metrics
from sklearn.cluster
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from sklearn.manifold import TSNE
import pandas
import seaborn
import os
from os import listdir
from os.path import isfile, join
from os.path import basename
from os.path import splitext
from os.path import dirname
```

Question 1

Write a program to implement a region segmentation algorithm using the hoop_1.mpeg algorithm as correlated "Shifting" data set of an image. Write a program to produce the segmentation of the image for consecutive frames. (2 marks)

In [2]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
img = cap.read()[1]
plt.imshow(img[:,:,::-1])
```

In [3]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
for i in range(0, 100):
    img = cap.read()[1]
plt.imshow(img[:,:,::-1])
```

Out[3]:

In [4]:
```
img = cv.VideoCapture('hoop_1.mpeg')
for i in range(0, 100):
    img = cap.read()[1]
plt.imshow(img[:,:,::-1])
```

Out[4]:

In [5]:
```
img = cv.VideoCapture('hoop_1.mpeg')
for i in range(0, 100):
    img = cap.read()[1]
plt.imshow(img[:,:,::-1])
```

Out[5]:

In [6]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
img = cap.read()[1]
plt.imshow(img[:,:,::-1])
```

In [7]:
```
def frame_diff(frame, prev):
    global cnt
    if cnt == 0:
        return frame
    else:
        diff = cv.absdiff(frame,prev)
        cnt += 1
        ret, th = cv.threshold(diff,15,255,0)
        fgmask = th[:,:,0]
        for i in range(0, h):
            for j in range(0, w):
                if fgmask[i, j] == 255:
                    fgmask[i, j] = 255
                else:
                    fgmask[i, j] = 0
        kernel = np.ones((5,5),np.uint8)
        fgmask = cv.dilate(fgmask,kernel,iterations=1)
        fgmask = cv.erode(fgmask,kernel,iterations=1)
        fgmask = cv.GaussianBlur(fgmask,(5,5),0)
        ret, fgmask = cv.threshold(fgmask,127,255,0)
        return fgmask
```

In [8]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame
```

In [9]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame
```

In [10]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame
```

In [11]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame
```

In [12]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame
```

In [13]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame
```

In [14]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame
```

Question 2

Write a program to obtain the spatial and contrast cues using 8x32 superpixels of an image instead of pixels. (2 marks)

In [15]:
```
img_path = "./images/2.jpg"
image = cv.imread(img_path)
plt.imshow(image[:,:,::-1])
print("Dimensions: ",image.shape)
```

Out[15]:

In [16]:
```
img = cv.imread(img_path)
h = img.shape[0]
w = img.shape[1]
print("Height: ",h,"\nWidth: ",w)
```

Out[16]:
```
Height:  1024
Width:  1280
```

In [17]:
```
def get_spatial_feature(image):
    feat = image.ravel()
    return feat[:3*32*8]
```

In [18]:
```
feat = get_spatial_feature(image)
print("Dimensions: ",feat.shape)
```

Out[18]:
```
Dimensions:  (768)
```

In [19]:
```
def get_contrast_feature(image, bins):
    feat = []
    for c in range(3):
        hist = cv.calcHist([image[:,:,c]], [0], None, [bins], [0, 256])
        hist = hist.ravel()
        hist = hist/(8*32)
        feat = feat + list(hist)
    return feat
```

In [20]:
```
feat = get_contrast_feature(image, 256)
print("Dimensions: ",len(feat))
```

Out[20]:
```
Dimensions:  768
```

In [21]:
```
def get_feature(image, bins):
    spatial = get_spatial_feature(image)
    contrast = get_contrast_feature(image, bins)
    feat = spatial + contrast
    return feat
```

In [22]:
```
feat = get_feature(image, 256)
print("Dimensions: ",len(feat))
```

Out[22]:
```
Dimensions:  1536
```

In [23]:
```
def get_features(image, bins):
    for i in range(0, 8):
        for j in range(0, 32):
            patch = image[i*(h//8):(i+1)*(h//8), j*(w//32):(j+1)*(w//32), :]
            feat = get_feature(patch, bins)
            if i == 0 and j == 0:
                features = np.array([feat])
            else:
                features = np.vstack((features, np.array([feat])))
    return features
```

In [24]:
```
features = get_features(image, 256)
print("Dimensions: ",features.shape)
```

Out[24]:
```
Dimensions:  (256, 1536)
```

In [25]:
```
def plot_image(image, title=""):
    image = image[:,:,::-1]
    plt.imshow(image)
    plt.title(title, fontsize=16)
    pass
```

In [26]:
```
plot_image(image, "Original Image")
```

Out[26]:

Spatial Cue

In human visual system, the regions near the image center draw more attention than the other regions. When the relative measure the spatial cue of the image is concerned, the efficient spatial representation. The simple spatial cue is the **Spatial Map** with a huge image dimension. The spatial cue is represented by the concatenation of the pixel values in the image patch. The spatial cue of 8 x 32 patch is defined as:

$$f_s(i,j) = \sum_{x=8i}^{8i+7} \sum_{y=32j}^{32j+31} \sum_{c=1}^{3} I(x,y,c) \cdot 2^{(x+y+c)}$$

where f_s is the Kronecker delta function, I denotes the center of image I^c, and Gaussian kernel $W(.)$ suppresses the Euclidean distance of the pixel from the center of the image. The spatial cue is a vector of dimension 768 (8 x 32 x 3) and is normalized to have the sum of one. The contrast cue of patch I^c, different from the single image cues, we spatial cue I^c represents the relative cues of the center image which is a global central bias in the multiple images.

The spatial cue is implemented, the spatial cue is **not stable in both single and multiple images**.

Reference: Google Research Football in Kaggle

In [27]:
```
img = cv.imread(img_path)
plot_image(image, "Original Image")
print("Dimensions: ",image.shape)

patch = image[0:(h//8), 0:(w//32), :]
plot_image(patch, "Patch")
print("Dimensions: ",patch.shape)

patch = np.flatten(patch)
print(patch)

k = 0
for c in range(3):
    for i in range(0, 8):
        for j in range(0, 32):
            patch = image[i*(h//8):(i+1)*(h//8), j*(w//32):(j+1)*(w//32), c]
            print("Channel: ",c,"\nRow: ",i,"\nCol: ",j,"\n",patch,"\n")
            k = k + 1
            if k == 5:
                break
print("Total: ",k)

print("Patch[0,0,0] = ", patch[0,0,0])
print("Patch[0,1,0] = ", patch[0,1,0])
print("Patch[1,0,0] = ", patch[1,0,0])
print("Patch[1,1,0] = ", patch[1,1,0])
print("Patch[0,0,1] = ", patch[0,0,1])
print("Patch[0,1,1] = ", patch[0,1,1])
print("Patch[1,0,1] = ", patch[1,0,1])
print("Patch[1,1,1] = ", patch[1,1,1])
print("Patch[0,0,2] = ", patch[0,0,2])
print("Patch[0,1,2] = ", patch[0,1,2])
print("Patch[1,0,2] = ", patch[1,0,2])
print("Patch[1,1,2] = ", patch[1,1,2])

patch = np.reshape(patch, (3,8,32))
print("Reshape")
print(patch)
print("Dimensions: ",patch.shape)

for c in range(3):
    print("Channel: ",c,"\n",patch[c,:,:],"\n")

patch = np.reshape(patch, (8,32,3))
print("Reshape")
print(patch)
print("Dimensions: ",patch.shape)

patch = np.reshape(patch, (3,8,32))
for c in range(3):
    print("Channel: ",c,"\n",patch[c,:,:],"\n")

patch = np.reshape(patch, (8,32,3))
plot_image(patch, "Patch")
print("Dimensions: ",patch.shape)
```

Out[27]:

```
Dimensions:  (1024, 1280, 3)
[ 255 255 255 ... 255 255 255]
Channel:  0
Row:  0
Col:  0
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]]

Channel:  0
Row:  0
Col:  1
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]]

Channel:  0
Row:  0
Col:  2
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]]

Channel:  0
Row:  0
Col:  3
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]]

Channel:  0
Row:  0
Col:  4
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]]

Total:  5
Patch[0,0,0] =  255
Patch[0,1,0] =  255
Patch[1,0,0] =  255
Patch[1,1,0] =  255
Patch[0,0,1] =  255
Patch[0,1,1] =  255
Patch[1,0,1] =  255
Patch[1,1,1] =  255
Patch[0,0,2] =  255
Patch[0,1,2] =  255
Patch[1,0,2] =  255
Patch[1,1,2] =  255
Reshape
[[[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]]
Dimensions:  (3, 8, 32)
Channel:  0
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]

Channel:  1
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]

Channel:  2
 [[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
  255 255 255 255 255 255 255 255 255 255 255 255 255 255]
Reshape
[[[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]
  [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   255 255 255 255 255 255 255 255 255 255 255 255 255 255]]
Dimensions:  (8, 32, 3)
```

Question 3

Implement the separation measure discussed in the 2D in of the following paper to obtain quality scores for the test code provided. Compute the quality scores as weights while performing the weighted sum of the two cues for grouping the frame sequence in Question 1.

In [28]:
```
def rgb2lab(rgb):
    lab = cv.cvtColor(rgb, cv.COLOR_RGB2LAB)
    return lab

for c in range(3):
    for i in range(0, 8):
        for j in range(0, 32):
            patch = image[i*(h//8):(i+1)*(h//8), j*(w//32):(j+1)*(w//32), c]
            print("Channel: ",c,"\nRow: ",i,"\nCol: ",j,"\n",patch,"\n")

patch = rgb2lab(patch)
print("Dimensions: ",patch.shape)

for c in range(3):
    print("Channel: ",c,"\n",patch[c,:,:],"\n")
```

In [29]:
```
def plot_histogram(image, title=""):
    for c in range(3):
        hist = cv.calcHist([image[:,:,c]], [0], None, [256], [0, 256])
        if c == 0:
            plot_hist = hist
        else:
            plot_hist = np.vstack((plot_hist, hist))
    plot_hist = np.ravel(plot_hist)
    plot_hist = np.reshape(plot_hist, (3,256))
    for c in range(3):
        plot_hist[c,:] = plot_hist[c,:]/(image.shape[0]*image.shape[1])
    for c in range(3):
        plt.plot(plot_hist[c,:])
    plt.ylabel('Normalized Count')
    plt.xlabel('Pixel Value')
    plt.title(title, fontsize=16)
    pass
```

In [30]:
```
img = cv.imread(img_path)
plot_histogram(image, "Histogram of Image")
```

Out[30]:

In [31]:
```
def get_separation(image, bins):
    for c in range(3):
        hist = cv.calcHist([image[:,:,c]], [0], None, [bins], [0, 256])
        if c == 0:
            plot_hist = hist
        else:
            plot_hist = np.vstack((plot_hist, hist))
    plot_hist = np.ravel(plot_hist)
    plot_hist = np.reshape(plot_hist, (3,bins))
    for c in range(3):
        plot_hist[c,:] = plot_hist[c,:]/(image.shape[0]*image.shape[1])
    for c in range(3):
        plot_hist[c,:] = np.cumsum(plot_hist[c,:])
    sep = np.zeros(bins)
    for i in range(0, bins):
        for c in range(3):
            sep[i] = sep[i] + plot_hist[c,i]*(1-plot_hist[c,i])
    return sep
```

In [32]:
```
sep = get_separation(image, 256)
print("Dimensions: ",sep.shape)
print("Sum: ",sum(sep))
print("Sum/Dim: ",sum(sep)/sep.shape[0])
```

Out[32]:
```
Dimensions:  (256)
Sum:  [0.99999]
Sum/Dim:  [0.00390]
```

In [33]:
```
def plot_separation(image, title=""):
    sep = get_separation(image, 256)
    plot_sep = np.cumsum(sep)
    plot_sep = plot_sep/sum(sep)
    plt.plot(plot_sep)
    plt.ylabel('Cumulative Separation')
    plt.xlabel('Pixel Value')
    plt.title(title, fontsize=16)
    pass
```

In [34]:
```
img = cv.imread(img_path)
plot_separation(image, "Separation Measure")
```

Out[34]:

Separation Measure

Separation measure is a perceptual level and separation theory and background distributions like a gaussian/multi-lobed. Separating distributions is defined as the amount of overlap between two distributions. We attempt to measure the separation between them by μ_1, μ_2, σ_1, σ_2 and π_1 (π_2 is redundant). Hence, we define the separation measure between two cues μ_1, μ_2, σ_1, σ_2 and π_1 (Gaussian and background distribution, respectively, intersection area in close proximity between 0 and 1. Specifically, the Gaussian and background distribution, respectively, intersection area in close proximity between 0 and 1. Specifically,

$$S(\mu_1, \mu_2, \sigma_1, \sigma_2, \pi_1) = \int_{-\infty}^{\infty} \min(\pi_1 G(\mu_1, \sigma_1)(x), (1-\pi_1) G(\mu_2, \sigma_2)(x)) dx$$

It is clear that the less the two distributions overlap with each other, the better the separation is. μ_1 the foreground and background are well separated, μ_2 the foreground and background are well separated, σ_1 the foreground and background are well separated, σ_2 the foreground and background are well separated, π_1 the foreground and background are well separated.

Other ways to define the separation measure:

$$\int_{-\infty}^{\infty} \min(\pi_1 G(\mu_1, \sigma_1)(x), (1-\pi_1) G(\mu_2, \sigma_2)(x)) dx$$

Having obtained σ (overlap $\int_{-\infty}^{\infty}$) it can now be extended to

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \min(\pi_1 G(\mu_1, \sigma_1)(x), (1-\pi_1) G(\mu_2, \sigma_2)(x)) dx$$

And finally, separation measure of the optimal cues μ is obtained as

$$S = \frac{\sigma_1 + \sigma_2 - 2\sigma}{\sigma_1 + \sigma_2}$$

where σ is an number of bins used for representing the two distributions.

Reference: Color-Based Image Segmentation Using Spectral and Texture Cues: Perceptual Interpretation and Computation

In [35]:
```
cap = cv.VideoCapture('hoop_1.mpeg')
prev = cap.read()[1]
prev = prev[0:h, 0:w]
cnt = 0
for i in range(0, 3):
    frame = cap.read()[1]
    frame = frame[0:h, 0:w]
    if i == 2:
        cv.imwrite('frame_diff_1.jpg',frame)
    fgmask = frame_diff(frame, prev)
    if i == 2:
        cv.imwrite('frame_diff_2.jpg',fgmask)
    prev = frame

img = cv.imread('frame_diff_1.jpg')
for c in range(3):
    print("Channel: ",c,"\n",img[:,:,c],"\n")

img = rgb2lab(img)
for c in range(3):
    print("Channel: ",c,"\n",img[:,:,c],"\n")

plot_histogram(img, "Histogram of Image")

sep = get_separation(img, 256)
print("Dimensions: ",sep.shape)
print("Sum: ",sum(sep))
print("Sum/Dim: ",sum(sep)/sep.shape[0])

plot_separation(img, "Separation Measure")
```

Out[35]:
```
Channel:  0
 [[ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]]

Channel:  1
 [[ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]]

Channel:  2
 [[ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]]

Channel:  0
 [[ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]
 [ 0 0 0 ... 0 0 0]]

Channel:  1
 [[128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]]

Channel:  2
 [[128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]
 [128 128 128 ... 128 128 128]]
```

Out[35]:


```
Dimensions:  (256)
Sum:  [0.99999]
Sum/Dim:  [0.00390]
```

Out[35]:

In [18]:
```
Original Image Shape: (1000,1000,3)
Original Image Dtype: uint8
Original Image Min: 0
Original Image Max: 255
Original Image Mean: 0.5
Original Image Median: 0.5
```

Max Contrast Cap

In [19]:
```
Image_Thresholded = 0.5
```

```
Image_Thresholded_Shape = (0,0,0)
Image_Thresholded_Dtype = uint8
Image_Thresholded_Contrast_Cap_Shape = (0,0,0)
Image_Thresholded_Contrast_Cap_Dtype = uint8
Image_Thresholded_Contrast_Cap_Min = 0.0
Image_Thresholded_Contrast_Cap_Max = 0.0
Image_Thresholded_Contrast_Cap_Mean = 0.0
Image_Thresholded_Contrast_Cap_Median = 0.0
```

Min Contrast Cap

In [20]:
```
Image_Thresholded = 0.5
```



```
Image_Thresholded_Shape = (0,0,0)
Image_Thresholded_Dtype = uint8
Image_Thresholded_Contrast_Cap_Shape = (0,0,0)
Image_Thresholded_Contrast_Cap_Dtype = uint8
Image_Thresholded_Contrast_Cap_Min = 0.0
Image_Thresholded_Contrast_Cap_Max = 0.0
Image_Thresholded_Contrast_Cap_Mean = 0.0
Image_Thresholded_Contrast_Cap_Median = 0.0
```

In [21]:
```
Image_Thresholded_Shape = Image_Thresholded.shape
Max_Line_Length_Pixel = 0
Max_Line_Gap_Pixel = 0
Image_Hough_Lines = np.zeros(Image_Thresholded.shape[0],dtype=np.int)
```

In [22]:
```
Image_Hough_Lines_Shape = Image_Hough_Lines.shape
Image_Hough_Lines_Dtype = Image_Hough_Lines.dtype
Image_Hough_Lines_Min = np.min(Image_Hough_Lines)
Image_Hough_Lines_Max = np.max(Image_Hough_Lines)
```

In [23]:
```
print Image_Hough_Lines_Shape, Image_Hough_Lines_Dtype
print Image_Hough_Lines_Min, Image_Hough_Lines_Max
```

```
(1000,) int64
0 0
```

In [24]:
```
Rho=1.0, Theta=np.pi/180, Threshold=0.5
```