Machine Perception 2020

Curtin University - Perth

Jonathan Wright
Curtin University
Computer Science Student
Perth, WA
19779085@student.curtin.edu.au

Fig. 1: Source Images:
1 - Red Diamond Playing Card
2 - Dugong, Calf with Seaweed.

***Abstract*—This paper will explore questions asked by the Machine Perception 2020 Assignment 1 Paper, this expands to 1 - Image Histograms, Harris Corner and SIFT Key variance/invariance. 2 - LBP, HOG and SIFT Feature Extraction. 3 - Object Extraction using CCL on Figures 1 and 2. 4 - Image Segmentation on Figures 1 and 2 using K-Means and different colorspaces.**

I. Task One

A. Generated Images

For this section I have generated images of various rotations: (180°, 45°, -45°, 90°, -90°).
I have also generated images of various scales: (100%, 50%, 25%, 50% Height 100% Width, 50% Width 100% Height)

B. Image Histograms

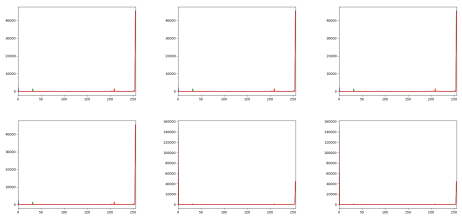

Fig. 2: Diamond Rotation Histograms: (Neutral, -90°, 90°, 180°, 45°and -45° NOTE: 45 Degrees will not be the same due to adding black)

Rotation:
Image Histograms are invariant to rotation, as you can see with my histograms they remain constant through rotations (90°, 180°, -90°, and 0°) a important note to make is that while 45°and -45°are not the same as the rest, this is because I had to add additional black to the image to fit it on the canvas.

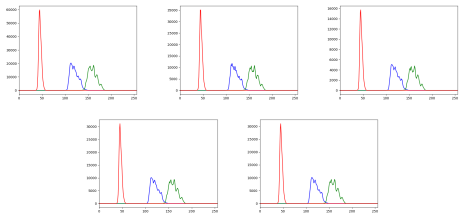

Fig. 3: Dugong Scaling Histograms: (Neutral, 75%, 50%, 50% width, 50% height)

Scaling:
Image Histograms are invariant to scaling, as you can see with my histograms they remain constant through scaling.

C. Harris Corner Detectors

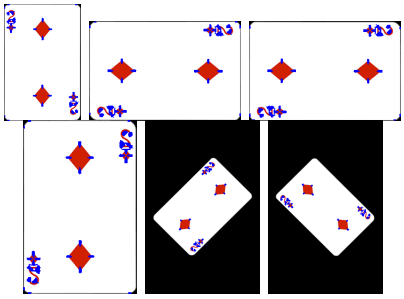

Fig. 4: Diamond Rotation Harris Corner Detections (Neutral, -90°, 90°, 180°, 45°and -45°)

Rotation:
From the results above we can see that the same edges are being detected on all rotations with no variancy, as such we can conclude that Harris Corner Detection is rotation invariant.

Scaling: From the results above I can conclude from visual

Identify applicable funding agency here. If none, delete this.

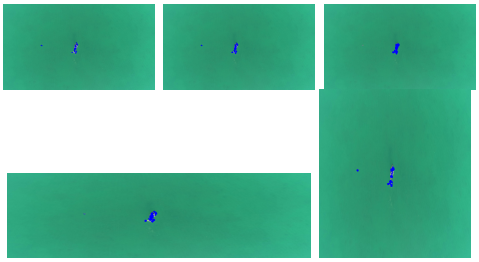

Fig. 5: Dugong Scaling Corner Detection: (Neutral, 75%, 50%, 50% height, 50% width)
While hard to see they are not the same!

inspection that Harris Corner Detection is scale variant, I conclude this by the fact that different edges are detected at different scales, sometimes the seaweed is not picked up (as in case 50% height).

D. SIFT Feature Detectors

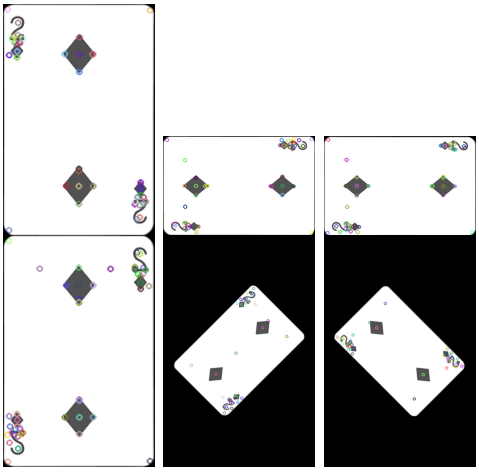

Fig. 6: Diamond SIFT Detections (Neutral, -90°, 90°, 180°, -45°and 45°)

Rotation:
As you can see from the above results, SIFT is rotation invariant as it detects the same points at each rotation.

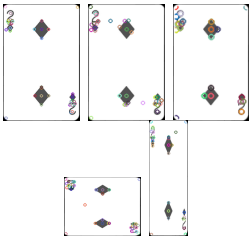

Fig. 7: Diamond Scaling SIFT: (Neutral, 75%, 50%, 50% height, 50% width)

Scale:
Sift appears to be mostly scale invariant based on my results above, so I can only conclude that it is scale invariant.

II. Task Two

A. LBP, HOG, SIFT Image Feature Detection

In this section I will describe the main steps for LBP, HOG and SIFT before proceeding to show SIFT vs HOG for variance.
LBP: First the image is converted to grayscale, the image is then segmented into 3 x 3 blocks, for each block we take the center value and compare it to each other pixel in the block, if the center is less than the value it is recorded as 0, if its greater than or equal to it is 1. We then combine the 0 and 1's to a binary number for that block, that number is stored in a frequency table. After we do LBP for the whole image we can say how many unique LBPs there are for a image with 256 gray levels and from this we can say how many features are in this image i.e 68 unique LBPs = 68 features in the image.
HOG: First the image is 'zipped' or 'cropped' into a smaller more compressed form (i.e 64x64 or 64x128) to a feature that you want to detect.
Then we divide the smaller image into a 8x8 block of cells, then for each cell we calculate a histogram of the gradients in that cell. Then once the histograms are calculated we iterate the image in bigger 16 x 16 blocks and normalise the 4 histograms (each 8x8 block will contain a histogram so 16 x 16 will contain 4 histograms), this will reduce lighting effects. Finally we calculate the final feature vector from by combining all of the 36 histograms into a vector.
SIFT: First SIFT will find a scale that works best for detection using Gaussian filters, it will segment the image and apply Gaussian to it, then it will calculate the difference of Gaussian and use this to find the scale.
Then SIFT will find a ideal orientation by computing gradient magnitudes and orientation using pixel differences of the guassian smoothed versions of image. Selects the best peaks of a histogram and chooses this as the orientation.
SIFT will then assign a descriptor to a region to describe the region.

B. Generated Images

For the Dugong and Diamond image I generated images at various rotations (0°, 180°, -90°, 90°)
and at various scales. (100%, 50%, 50% width, 50% height)
(For the diamond I selected a number, for the dugong I selected the dugong)

C. Hog Variancy

To test for HOG Variancy I was unsure how to calculate or visualise how variant it was so I used the rule of H0 - H1 where H0 is hog feature of original images and H1 is hog feature of new image, and if the distance is less than H0 value then its not variant. For both scaled and rotated images 'Transformation is invariant as distance is relatively small compared to norm of H0' was output by my tests, however, after some research I believe this is wrong as it is meant to be invariant.

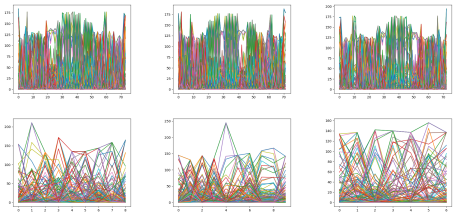

Fig. 8: SIFT Variancy (First 3: Rotations (90°, 180°, -90°) Last 3: Scaled (50%, 50% Height, 50% Width))

D. SIFT Variancy

The histograms above conclude that while SIFT is rotationally invariant it appears that SIFT is not scale invariant, however, I would have to say that something is wrong with my code as SIFT stands for 'SCALE' invariant feature detection, so it should be scale invariant...

III. Task Three

A. Seperate Of Images Into Binary Images

* Note: You can see them as binary numbers in the relevant .txt file. Above you can see my seperation of the foreground

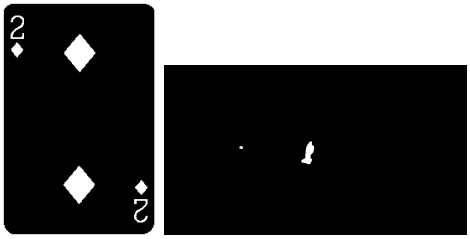

Fig. 9: Binary Versions Of Diamond and Dugong

(White) and background (Black), you can also see it saved as a txt file where 1 is foreground, 0 is background in my source code out_files folder.

B. Card Objects

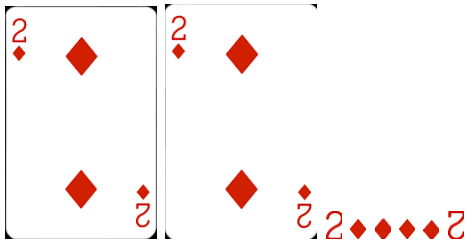

Fig. 10: Card Detected Objects Via CCL: Border (Black), Card BG, Number 2, Diamond, Diamond, Diamond, Diamond, Number 2

Using the Connected-Component Labelling algorithm (two-pass) I seperated the card into 8 objects: the border, the card's background, the four diamonds and the two 2s.

Fig. 11: Dugong Detected Objects Via CCL: Ocean, Dugong & Calf and Seaweed

C. Dugong Objects

Using the Connected-Component Labelling algorithm (two-pass) I seperated the card into 3 objects: The seaweed, dugong and calf and ocean.

IV. Image Segmentation with K-Means

For each image I used HSV, LAB, RGB, XYZ, YUV colorspaces for each image.

A. Card

I tested card with 1, 2 and 3 clusters as ¿4 made no impact. After testing image segmentation with 1 cluster, 2 clusters, 3

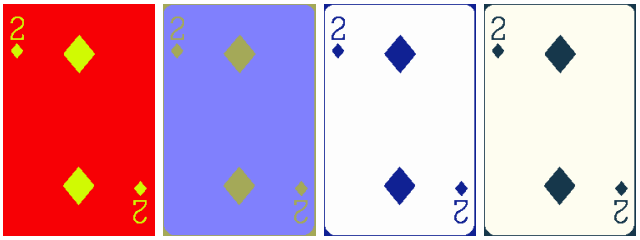

Fig. 12: Card Image Segmentation (2 Clusters as it worked the same as 3): HSV, LAB, RGB, XYZ

clusters, 4 clusters ... up to 5 I concluded that 1 - 3 clusters were the only impacts on image, I also concluded that color spaces didn't really matter for this image as all colour spaces successfully segmented the card image at 2 clusters.

B. Dugong

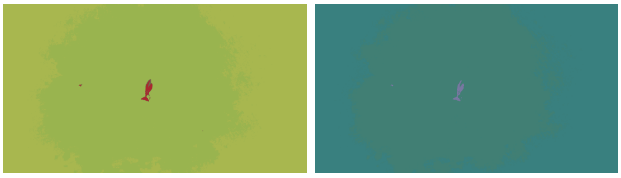

Fig. 13: Dugong Image Segmentation (HSV at 4, YUV at 3): HSV, YUV

After testing image segmentation with 1 cluster, 2 clusters, 3 clusters, 4 clusters ... up to 10 I concluded that 1 - 4 clusters were the only impacts on image, I found that 4 clusters HSV and 3 clusters YUV extracted the most useful data i.e YUV at 4 and HSV at 3 detected alot of the dugong and the seaweed.

V. Appendix 1 - (Please Refer To Appendix 5 For utils.py)

```
import cv2 as cv
import numpy as np
import utils
import math
import random
title = {'./images/diamond2.png': 'diamond', './images/Dugong.jpg': 'Dugong'} #
imageSources = ['./images/diamond2.png', './images/Dugong.jpg']
for imageSource in imageSources: # ROTATION
  images = {
    'neutral': [cv.imread(imageSource, cv.IMREAD_COLOR)],
    'deg45': [utils.rAngle(cv.imread(imageSource, cv.IMREAD_COLOR),
              math.radians(45))],
    'degn45': [utils.rAngle(cv.imread(imageSource, cv.IMREAD_COLOR),
               math.radians(-45))],
    'deg90': [cv.rotate(cv.imread(imageSource, cv.IMREAD_COLOR),
              cv.ROTATE_90_CLOCKWISE)],
    'degn90': [cv.rotate(cv.imread(imageSource, cv.IMREAD_COLOR),
               cv.ROTATE_90_COUNTERCLOCKWISE)],
    'flipped': [cv.flip(cv.imread(imageSource, cv.IMREAD_COLOR), 0)]
  }
  for key, image in images.items(): # Hist
    utils.calcHist(image[0],
      './out_files/rotation/{0}{1} Hist.png'.format(title[imageSource], key))
  for image in images.values(): # Corner Detection
    cpy = image[0].copy()
    cpy[cv.cornerHarris(cv.cvtColor(image[0],
        cv.COLOR_BGR2GRAY), 5, 9, 0.05)>0.01*cpy.max()]=[255,0,0]
    image.append(cpy)
  cv.imwrite('./out_files/rotation/{0}Neutral Corners.png'
    .format(title[imageSource]), images['neutral'][1])
  cv.imwrite('./out_files/rotation/{0}Deg 45 Corners.png'
    .format(title[imageSource]), images['deg45'][1])
  cv.imwrite('./out_files/rotation/{0}Deg -45 Corners.png'
    .format(title[imageSource]), images['degn45'][1])
  cv.imwrite('./out_files/rotation/{0}Deg 90 Corners.png'
    .format(title[imageSource]), images['deg90'][1])
  cv.imwrite('./out_files/rotation/{0}Deg -90 Corners.png'
    .format(title[imageSource]), images['degn90'][1])
  cv.imwrite('./out_files/rotation/{0}180 Deg Corners.png'
    .format(title[imageSource]), images['flipped'][1]) # Rotationally Invariant
  for image in images.values(): # SIFT
    SIFT = cv.xfeatures2d.SIFT_create()
    kp, des = SIFT.detectAndCompute(
      cv.cvtColor(image[0], cv.COLOR_BGR2GRAY), None)
    image.append(cv.drawKeypoints(
      cv.cvtColor(image[0], cv.COLOR_BGR2GRAY), kp, image[0]))
  cv.imwrite('./out_files/rotation/{0}Neutral SIFT.png'
    .format(title[imageSource]), images['neutral'][2])
  cv.imwrite('./out_files/rotation/{0}Deg 45 SIFT.png'
    .format(title[imageSource]), images['deg45'][2])
  cv.imwrite('./out_files/rotation/{0}Deg -45 SIFT.png'
    .format(title[imageSource]), images['degn45'][2])
  cv.imwrite('./out_files/rotation/{0}Deg 90 SIFT.png'
    .format(title[imageSource]), images['deg90'][2])
```

```
  cv.imwrite('./out_files/rotation/{0}Deg -90 SIFT.png'
    .format(title[imageSource]), images['degn90'][2])
  cv.imwrite('./out_files/rotation/{0}180 Deg SIFT.png'
    .format(title[imageSource]), images['flipped'][2]) # Rotationally Invariant

for imageSource in imageSources: # Scaling
  images = {
    'neutral': [cv.imread(imageSource, cv.IMREAD_COLOR)],
    '75%': [utils.scaleImg(
      cv.imread(imageSource, cv.IMREAD_COLOR), 0.75, 0.75)],
    '50%': [utils.scaleImg(
      cv.imread(imageSource, cv.IMREAD_COLOR), 0.50, 0.50)],
    'widthReduced': [utils.scaleImg(
      cv.imread(imageSource, cv.IMREAD_COLOR), 0.50, 1)],
    'heightReduced': [utils.scaleImg(
      cv.imread(imageSource, cv.IMREAD_COLOR), 1, 0.50)]
  }

  # Hist
  for key, image in images.items():
    utils.calcHist(image[0], './out_files/scale/{0}{1} Hist.png'
    .format(title[imageSource], key))
  for image in images.values():
    cpy = image[0].copy()
    cpy[cv.cornerHarris(cv.cvtColor(image[0],
        cv.COLOR_BGR2GRAY), 5, 9, 0.05)>0.01*cpy.max()]=[255,0,0]
    image.append(cpy)
  cv.imwrite('./out_files/scale/{0}Neutral Corners.png'
    .format(title[imageSource]), images['neutral'][1])
  cv.imwrite('./out_files/scale/{0}75 Corners.png'
    .format(title[imageSource]), images['75%'][1])
  cv.imwrite('./out_files/scale/{0}50 Corners.png'
    .format(title[imageSource]), images['50%'][1])
  cv.imwrite('./out_files/scale/{0}widthReduced Corners.png'
    .format(title[imageSource]), images['widthReduced'][1])
  cv.imwrite('./out_files/scale/{0}heightReduced Corners.png'
    .format(title[imageSource]), images['heightReduced'][1]) # Scale Invariant
  for image in images.values(): # SIFT
    SIFT = cv.xfeatures2d.SIFT_create()
    kp, des = SIFT.detectAndCompute(cv.cvtColor(image[0], cv.COLOR_BGR2GRAY), N
    image.append(cv.drawKeypoints(cv.cvtColor(image[0], cv.COLOR_BGR2GRAY), kp,
  cv.imwrite('./out_files/scale/{0}Neutral SIFT.png'
    .format(title[imageSource]), images['neutral'][2])
  cv.imwrite('./out_files/scale/{0}75 SIFT.png'
    .format(title[imageSource]), images['75%'][2])
  cv.imwrite('./out_files/scale/{0}50 SIFT.png'
    .format(title[imageSource]), images['50%'][2])
  cv.imwrite('./out_files/scale/{0}widthReduced SIFT.png'
    .format(title[imageSource]), images['widthReduced'][2])
  cv.imwrite('./out_files/scale/{0}heightReduced SIFT.png'
    .format(title[imageSource]), images['heightReduced'][2]) # Mostly Scale Inv
```

```
import cv2 as cv
import numpy as np
import utils
import math
from matplotlib import pyplot as plt
hog = cv.HOGDescriptor('hog.xml')
winStride = (8,8)
padding = (8,8)
locations = ((0,0),)
card = cv.imread('./images/diamond2.png', cv.IMREAD_GRAYSCALE)
card90 = cv.rotate(card, cv.ROTATE_90_CLOCKWISE)
cardn90 = cv.rotate(card, cv.ROTATE_90_COUNTERCLOCKWISE)
card180 = cv.rotate(card, cv.ROTATE_180)
dugong = cv.imread('./images/Dugong.jpg', cv.IMREAD_GRAYSCALE)
card = card[5:69, 0:64] # crop to 64x64 number 2 feature.
dugong = dugong[209:209+64, 390:390+64] # crop to 64x64 dugong feature.
dugong90 = cv.rotate(dugong, cv.ROTATE_90_CLOCKWISE)
dugongn90 = cv.rotate(dugong, cv.ROTATE_90_COUNTERCLOCKWISE)
dugong180 = cv.rotate(dugong, cv.ROTATE_180)
cardHog = [
  (hog.compute(card,winStride,padding,locations), 'No Rotation'),
  (hog.compute(card90,winStride,padding,locations), '90 Degree Rotation'),
  (hog.compute(cardn90,winStride,padding,locations), '-90 Degree Rotation'),
  (hog.compute(card180,winStride,padding,locations), '180 Degree Rotation')
]
dugongHog = [
  (hog.compute(dugong,winStride,padding,locations), 'No Rotation'),
  (hog.compute(dugong90,winStride,padding,locations), '90 Degree Rotation'),
  (hog.compute(dugongn90,winStride,padding,locations), '-90 Degree Rotation'),
  (hog.compute(dugong180,winStride,padding,locations), '180 Degree Rotation')
]
with open("./out_files/task_2/rotation/cardHogResults.txt", "w") as f:
  for h in cardHog:
    f.write("{0} = {1}\n".format(h[1],
      utils.hogVariation(cardHog[0][0][0], h[0][0][0])))
with open("./out_files/task_2/rotation/dugongHogResults.txt", "w") as f:
  for h in dugongHog:
    f.write("{0} = {1}\n".format(h[1],
      utils.hogVariation(dugongHog[0][0][0], h[0][0][0])))

scaledDugongs = [dugong, utils.placeOn(utils.scaleImg(dugong, 0.50, 0.50), 64,
                 utils.placeOn(utils.scaleImg(dugong, 0.25, 0.25), 64, 64),
                 utils.placeOn(utils.scaleImg(dugong, 0.50, 1), 64, 64),
                 utils.placeOn(utils.scaleImg(dugong, 1, 0.50), 64, 64)]
scaledCards = [card, utils.placeOn(utils.scaleImg(card, 0.50, 0.50), 64, 64),
               utils.placeOn(utils.scaleImg(card, 0.25, 0.25), 64, 64),
               utils.placeOn(utils.scaleImg(card, 0.50, 1), 64, 64),
               utils.placeOn(utils.scaleImg(card, 1, 0.50), 64, 64)]
cardHog = [
  (hog.compute(card,winStride,padding,locations), 'No Scale'),
  (hog.compute(scaledCards[1],winStride,padding,locations), '50%_y 50%_x'),
  (hog.compute(scaledCards[2],winStride,padding,locations), '25%_y 25%_x'),
  (hog.compute(scaledCards[3],winStride,padding,locations), '100%_x 50%_y'),
  (hog.compute(scaledCards[4],winStride,padding,locations), '100%_y 50%_x')
```

```
]
dugongHog = [
  (hog.compute(dugong,winStride,padding,locations), 'No Scale'),
  (hog.compute(scaledDugongs[1],winStride,padding,locations), '50%_x 50%_y'),
  (hog.compute(scaledDugongs[2],winStride,padding,locations), '25%_x 25%_y'),
  (hog.compute(scaledDugongs[3],winStride,padding,locations), '50%_x 100%_y'),
  (hog.compute(scaledDugongs[4],winStride,padding,locations), '100%_x 50%_y')
]
with open("./out_files/task_2/scaled/cardHogResults.txt", "w") as f:
  for h in cardHog:
    f.write("{0} = {1}\n"
      .format(h[1], utils.hogVariation(cardHog[0][0][0], h[0][0][0])))
with open("./out_files/task_2/scaled/dugongHogResults.txt", "w") as f:
  for h in dugongHog:
    f.write("{0} = {1}\n"
      .format(h[1], utils.hogVariation(dugongHog[0][0][0], h[0][0][0])))

# Sift
SIFT = cv.xfeatures2d.SIFT_create()
cardHists = [
  (SIFT.detectAndCompute(card, None),
  './out_files/task_2/rotation/noRotation-card-SIFT.png'),
  (SIFT.detectAndCompute(card90, None),
  './out_files/task_2/rotation/90deg-card-SIFT.png'),
  (SIFT.detectAndCompute(cardn90, None),
  './out_files/task_2/rotation/n90deg-card-SIFT.png'),
  (SIFT.detectAndCompute(card180, None),
  './out_files/task_2/rotation/180deg-card-SIFT.png')
]
dugongHists = [
  (SIFT.detectAndCompute(dugong, None),
  './out_files/task_2/rotation/noRotation-dugong-SIFT.png'),
  (SIFT.detectAndCompute(dugong90, None),
  './out_files/task_2/rotation/90deg-dugong-SIFT.png'),
  (SIFT.detectAndCompute(dugongn90, None),
  './out_files/task_2/rotation/n90deg-dugong-SIFT.png'),
  (SIFT.detectAndCompute(dugong180, None),
  './out_files/task_2/rotation/180deg-dugong-SIFT.png')
]
for hist in cardHists + dugongHists:
  plt.plot(hist[0][1])
  plt.savefig(hist[1])
  plt.clf()
cardHists = [
  (SIFT.detectAndCompute(scaledCards[0], None),
  './out_files/task_2/scaled/noScale-card-sift.png'),
  (SIFT.detectAndCompute(scaledCards[1], None),
  './out_files/task_2/scaled/50w50h-card-sift.png'),
  (SIFT.detectAndCompute(scaledCards[2], None),
  './out_files/task_2/scaled/25w25h-card-sift.png'),
  (SIFT.detectAndCompute(scaledCards[3], None),
  './out_files/task_2/scaled/50w1h-card-sift.png'),
  (SIFT.detectAndCompute(scaledCards[4], None),
  './out_files/task_2/scaled/1w50h-card-sift.png')
]
```

```
dugongHists = [
  (SIFT.detectAndCompute(scaledDugongs[0], None),
  './out_files/task_2/scaled/noScale-dugong-sift.png'),
  (SIFT.detectAndCompute(scaledDugongs[1], None),
  './out_files/task_2/scaled/50w50h-dugong-sift.png'),
  (SIFT.detectAndCompute(scaledDugongs[3], None),
  './out_files/task_2/scaled/50w1h-dugong-sift.png'),
  (SIFT.detectAndCompute(scaledDugongs[4], None),
  './out_files/task_2/scaled/1w50h-dugong-sift.png')
]
for hist in dugongHists:
  plt.plot(hist[0][1])
  plt.savefig(hist[1])
  plt.clf()
```

```
import cv2 as cv
import numpy as np
import utils
import random

card = cv.imread("./images/diamond2.png", cv.IMREAD_GRAYSCALE)
dugong = cv.imread('./images/Dugong.jpg', cv.IMREAD_GRAYSCALE)
blockSize = 11
inbuiltBlur = 5
dugong = cv.adaptiveThreshold(dugong,255, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, blockSize, inbuiltBlur)
ker = np.ones((10,10), np.uint8)
dugong = cv.morphologyEx(dugong, cv.MORPH_OPEN, ker)
ker = np.ones((4,5), np.uint8)
dugong = cv.morphologyEx(dugong, cv.MORPH_CLOSE, ker)
_, card = cv.threshold(card, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
dugong = cv.bitwise_not(dugong)
card = cv.bitwise_not(card)
cv.imwrite("./out_files/task_3/dugong_objects/BinaryImage.png", dugong)
cv.imwrite("./out_files/task_3/card_objects/BinaryImage.png", card)
dugong = cv.bitwise_not(dugong)
card = cv.bitwise_not(card)
dugong = utils.imgToBinary(dugong)
card = utils.imgToBinary(card)
utils.toFile(dugong, './out_files/task_3/dugong_objects/dugong.txt')
utils.toFile(card, './out_files/task_3/card_objects/card_before.txt')

utils.concon(dugong); utils.concon(card) # my connected components method

utils.toFile(dugong, './out_files/task_3/dugong_objects/dugong_objects.txt')
utils.toFile(card, './out_files/task_3/card_objects/card_objects.txt')
dugongObjs = utils.conObjects(dugong)
cardObjs = utils.conObjects(card)
objs = [(dugongObjs, './images/Dugong.jpg',
'./out_files/task_3/dugong_objects/detectedObjectsCCL.png',
'./out_files/task_3/dugong_objects/'),
(cardObjs, './images/diamond2.png',
'./out_files/task_3/card_objects/detectedObjectsCCL.png',
'./out_files/task_3/card_objects/')]
for obj in objs:
  clr = cv.imread(obj[1], cv.IMREAD_COLOR)
  for key, coords in obj[0].items():
    color =  (random.randint(0, 255), random.randint(0, 255),
    random.randint(0, 255))
    for cord in coords:
      clr = cv.rectangle(clr,
      (cord[1], cord[0]), (cord[1], cord[0]), color)
  cv.imwrite(obj[2], clr)

  # Extract The Objects Into Seperate Files
  objectNo = 1
  for key, coords in obj[0].items():
    xMin = coords[0][1]; xMax = coords[0][1]
    yMin = coords[0][0]; yMax = coords[0][0]
```

```
    for coord in coords:
      if coord[1] < xMin:
        xMin = coord[1]
      elif coord[1] > xMax:
        xMax = coord[1]
      if coord[0] < yMin:
        yMin = coord[0]
      elif coord[0] > yMax:
        yMax = coord[0]
    out = cv.imread(obj[1], cv.IMREAD_COLOR)
    out = out[yMin:yMax, xMin:xMax]
    cv.imwrite("{}DetectedObject-{}.png".format(obj[3], objectNo), out)
    objectNo += 1
```

VIII. Appendix 4 - (Please Refer To Appendix 5 For utils.py)

```
import cv2 as cv
import numpy as np

colorspaces = {
  "RGB": cv.COLOR_BGR2RGB,
  "LAB": cv.COLOR_BGR2LAB,
  "XYZ": cv.COLOR_BGR2XYZ,
  "YUV": cv.COLOR_BGR2YUV,
  "HSV": cv.COLOR_BGR2HSV
}

def kmeans(image, pixel_val, eps=0.2, k = 3): # default epislon of 0.2
  pixel_val = np.float32(pixel_val)
  criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100, eps)
  _, labels, (centers) = cv.kmeans(pixel_val, k, None, criteria, 10,
    cv.KMEANS_RANDOM_CENTERS)
  centers = np.uint8(centers)
  labels = labels.flatten()
  segmented_image = centers[labels.flatten()]
  segmented_image = segmented_image.reshape(image.shape)
  return segmented_image

dugong = cv.imread("./images/Dugong.jpg")
diamond = cv.imread("./images/diamond2.png")
for colorspace, transform in colorspaces.items():
  for i in range(1, 4):
    img = kmeans(cv.cvtColor(diamond, transform),
      cv.cvtColor(diamond, transform).reshape((-1, 3)), k=i)
    cv.imwrite("./out_files/task_4/card/{} clusters {}.png"
      .format(i, colorspace), img)
  for i in range(1, 5):
    img = kmeans(cv.cvtColor(dugong, transform),
      cv.cvtColor(dugong, transform).reshape((-1, 3)), k=i)
    cv.imwrite("./out_files/task_4/dugong/{} clusters {}.png"
      .format(i, colorspace), img)
```

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import math

def rAngle(img, ang): # angle in radians.
  canvas = np.zeros((int(img.shape[0] * 2), int(img.shape[1] * 2), 3),
    dtype=np.uint8)
  h, w, z = img.shape
  hh, ww, z = canvas.shape
  yoff = round((hh-h)/2)
  xoff = round((ww-w)/2)
  result = canvas.copy()
  result[yoff:yoff+h, xoff:xoff+w] = img
  img = result
  degrees = (ang * 180) / math.pi
  center = tuple(np.array(img.shape[1::-1]) / 2)
  rotateMatrix = cv.getRotationMatrix2D(center, degrees, 1.0)
  return cv.warpAffine(img, rotateMatrix, img.shape[1::-1],
    flags=cv.INTER_LINEAR)

def scaleImg(img, widthScale, heightScale):
  newW = int(img.shape[1] * widthScale)
  newH = int(img.shape[0] * heightScale)
  img = cv.resize(img, (newW, newH))
  return img

def calcHist(img, filename):
  color = ('b','g','r')
  for i, col in enumerate(color):
    histr = cv.calcHist([img], [i], None, [256], [0, 256])
    plt.plot(histr, color = col)
    plt.xlim([0, 256])
  plt.savefig(filename)
  plt.clf()

def placeOn(img, desiredWidth, desiredHeight):
  canvas = np.zeros((desiredHeight, desiredWidth), dtype=np.uint8)
  canvas[0:img.shape[0], 0:img.shape[1]] = img
  return canvas

def toFile(arr, filename):
  with open(filename, 'w') as file:
    for y in range(len(arr)):
      for x in range(len(arr[y])):
        file.write(str(arr[y][x]))
      file.write('\n')

def hogVariation(H0, H1):
  distance = cv.norm(H0 - H1)
  if (distance <= cv.norm(H0)):
    return "Transformation is invariant as distance is relatively small compare
  else:
    return "Transformation is variant as distance is not relatively small compa
```

```
def concon(arr):
  conflicts = {} # dict
  cc = 0
  for y in range(len(arr)):
    for x in range(len(arr[y])):
      if arr[y][x] == 1:
        if y > 0:
          if x > 0:
            if arr[y][x-1] != 0:
              if arr[y-1][x] != 0:
                if arr[y-1][x] != arr[y][x-1]:
                  if conflicts.get(arr[y-1][x]) != None: #TOP
                    conflicts[arr[y-1][x]].add(arr[y][x-1]) # add left to up
                    conflicts[arr[y-1][x]].add(arr[y-1][x]) # add up to left
                  else:
                    z = set()
                    z.add(arr[y][x-1]) # add left
                    z.add(arr[y-1][x]) # add up
                    conflicts[arr[y-1][x]] = z
                  if conflicts.get(arr[y][x-1]) != None: # LEFT
                    conflicts[arr[y][x-1]].add(arr[y-1][x]) # add up
                    conflicts[arr[y][x-1]].add(arr[y][x-1]) # add left
                  else:
                    z = set()
                    z.add(arr[y-1][x]) # add up
                    z.add(arr[y][x-1]) # add left
                    conflicts[arr[y][x-1]] = z
                  # merge the sets.
                  conflicts[arr[y-1][x]].update(conflicts[arr[y][x-1]])
                  conflicts[arr[y][x-1]].update(conflicts[arr[y-1][x]])
                  # UPDATE THEIR CHILDREN
                  for child in conflicts[arr[y-1][x]]:
                    conflicts[child].update(conflicts[arr[y-1][x]])
                  for child in conflicts[arr[y][x-1]]:
                    conflicts[child].update(conflicts[arr[y][x-1]])
                arr[y][x] = arr[y][x-1]
              else:
                arr[y][x] = arr[y][x-1]
            elif arr[y-1][x] != 0:
              arr[y][x] = arr[y-1][x]
            else: #not connected
              cc += 1
              arr[y][x] = cc
          elif arr[y-1][x] != 0:
            arr[y][x] = arr[y-1][x]
          else: # not connected
            cc += 1
            arr[y][x] = cc
        elif x > 0:
          if arr[y][x-1] != 0:
            arr[y][x] = arr[y][x-1]
          else: # not connected
            cc += 1
```

```
            arr[y][x] = cc
        else: # not connected
          cc += 1
          arr[y][x] = cc
      elif arr[y][x] == 0:
        arr[y][x] = 0
  for y in range(len(arr)):
    for x in range(len(arr[y])):
      if conflicts.get(arr[y][x]) != None:
        arr[y][x] = min(conflicts[arr[y][x]])

def conObjects(arr):
  uniqueObjs = {}
  for y in range(len(arr)):
    for x in range(len(arr[y])):
      if uniqueObjs.get(arr[y][x]) != None:
        uniqueObjs[arr[y][x]].append((y,x))
      else:
        new = []
        new.append((y,x))
        uniqueObjs[arr[y][x]] = new
  return uniqueObjs

def imgToBinary(img):
  binOutput = np.zeros(shape = (img.shape[0], img.shape[1]), dtype=np.int)
  for y in range(img.shape[0]):
    for x in range(img.shape[1]):
      if img[y, x] == 255:
        binOutput[y][x] = 0
      else:
        binOutput[y][x] = 1
  return binOutput
```