EGE ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
434 GÖRÜNTÜ İŞLEME
2019-2020 GÜZ YARIYILI
TAKE HOME

Deniz Yürekdeler 0518000001

1) Paint Shop Pro, Photoshop ve GIMP yazılımlarının özelliklerini inceleyiniz ve bu üç yazılımı karşılaştırınız (tablo şeklinde de yazılabilir) [1 sayfa]
Ücretli olanlar internet üzerinden araştırılabilir veya deneme sürümleri indirilebilir. (20 p)

Answer:

Paintshop Pro	Photoshop	GIMP
Easier to Learn	Harder to Master	Easier to Learn
Cheaper	Expensive	Free of Use
Most Compatible with PC	Most Compatible with MAC	Most Compatible with PC
Built-in Tutorials	Avaible Smart Phone Version	Customizable Interface
	High-End Technologies	Open Source & Modifiable
	Cloud Storage	Lighter on Hardware

Figure 1 Comprasion of Paintshop Pro, Photoshop and GIMP

2) İki farklı resim bulunuz. Bu resimleri 2 farklı bölütleme (segmentation) yöntemi (threshold, watershed, region growing, split-and-merge, border tracing, …) ile bölütleyiniz. Görüntülerin ilk ve son hallerini rapora ekleyerek, yöntemleri kısaca anlatıp sonuçları yorumlayınız. Görüntülerde kaçar nesne olduğunu saydırınız ve raporda belirtiniz. (20 p)
İpucu : https://scikit-image.org/docs/dev/api/skimage.segmentation.html

Answer :

- Threshold Segmentation :

  Thresholding is the simplest method of image segmentation. From a grayscale image, thresholding can be used to create binary images. Binary images are one that consists of pixels that can have one of exactly two colors, usually black and white. Binary image processing often arise in digital image processing as masking or thresholding.

Thresholding methods replace each pixel in an image with a black pixel if the image intensity is less than some fixed constant T, or a while pixel if the image intensity is greater than that constant.

In the image below, Otsu Thresholding method is implemented for this document.

Otsu's method is an adaptive thresholding way for binarization in image processing. By going through all possible threshold values (from 0 to 255), it can find the optimal threshold value of input image.

This code first reads an image and then converts it to grayscale in order to transform it to binary image with thresholding. After the throsholding, some holes that still exist are eliminated with binary_fill_holes method(or morphological closing can be used). After the process finishes, each segment is labeled and the maximum label value is outputed as the total number of segments in the image.

```python
from skimage import io, filters
from scipy import ndimage
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage import measure

image = io.imread('pusheen.jpg')
#work on extra copy of image to keep the original untouched
im = image
#convert image to gray
im = rgb2gray(im)
#threshold segmentation
val = filters.threshold_otsu(im)
drops = ndimage.binary_fill_holes(im < val)

labels = measure.label(drops)
#prints max value label which is equal to total num.
print(labels.max())

# display results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8), sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title("Original")

ax[1].imshow(drops, cmap="gray")
ax[1].set_title("Segmented")

for a in ax:
    a.axis('off')

fig.tight_layout()
plt.show()
```

Figure 2 Threshold Segmentation Python Code

These are the before and after pictures of Thresholding Segmentation.

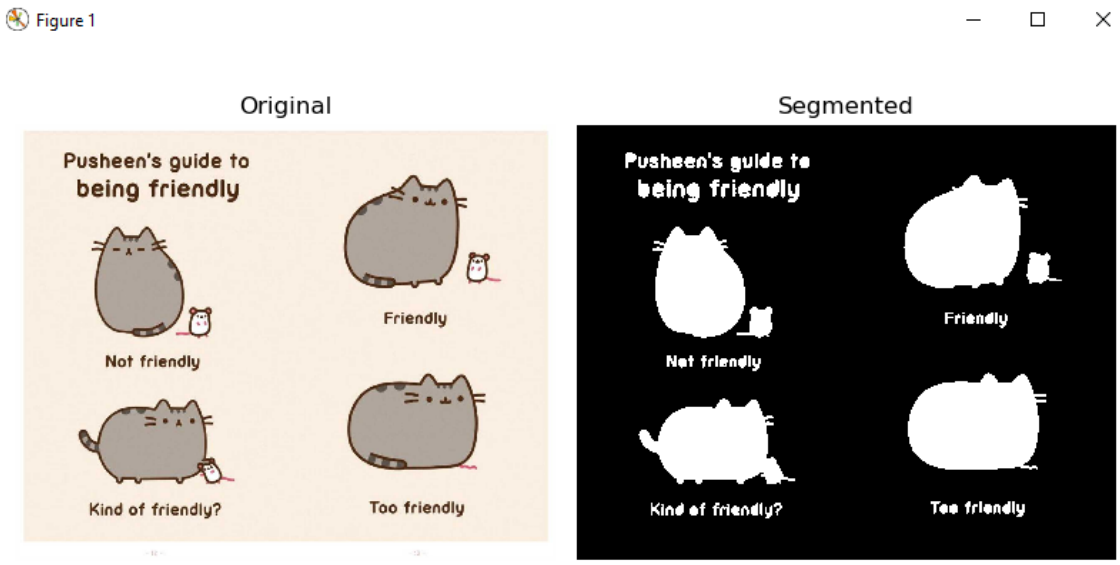


Figure 3 Original Image and Threshold Segmentation Result

Figure 4 Original Image and Threshold Segmentation Result

- Watershed Segmentation :

Watershed is a tranformation defined on a grayscale image. This method seperates adjacent objects in the image. The watershed transformation treats the image as if it is operating upon a topographic map, with the brightness of each point representing the height, and finds the lines that run along the tops of ridges.

In the image below the Watershed method is implemented for this document. This method starts by reading an image and converting it to grayscale. After the transformation to grayscale, noises are eliminated by using median filtering. When noises are eliminated, marking the segments starts. Outputs the maximum label value as the total number of segments in the image.

```python
from scipy import ndimage as ndi
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage.morphology import watershed, disk
from skimage.filters import rank
from skimage import io, measure

image_o = io.imread("pusheen.jpg")
#convert image to grayscale
image = rgb2gray(image_o)

# denoise image
denoised = rank.median(image, disk(2))
# find continuous region (low gradient - where less than 10 for this image) --> markers
# disk(5) is used here to get a more smooth image
markers = rank.gradient(denoised, disk(5)) < 10
markers = ndi.label(markers)[0]
# local gradient (disk(2) is used to keep edges thin)
gradient = rank.gradient(denoised, disk(2))
# process the watershed
labels = watershed(gradient, markers)

count = measure.label(labels)
#print label with max value , which is equal to total num
print(count.max())

# display results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8),sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(image_o, cmap=plt.cm.gray)
ax[0].set_title("Original")

ax[1].imshow(image, cmap=plt.cm.gray)
ax[1].imshow(labels, cmap=plt.cm.nipy_spectral, alpha=.7)
ax[1].set_title("Segmented")

for a in ax:
    a.axis('off')

fig.tight_layout()
plt.show()
```

Figure 5 Watershed Segmentation & Labeling Python Code

These are the before and after pictures of the Watershed Segmentation.

Figure 6 Original Image and Watershed Segmentation & Area Labeling Result

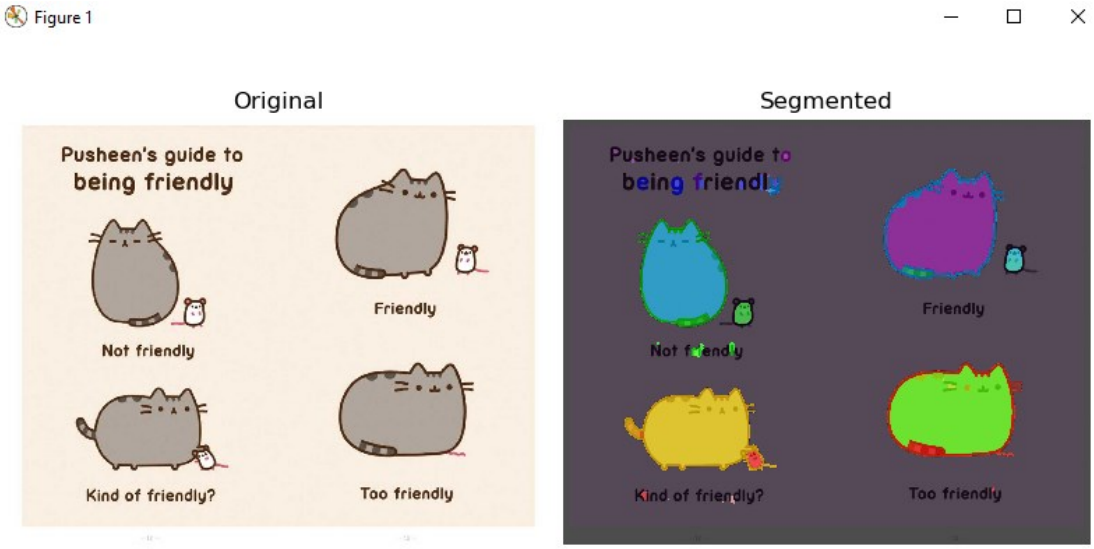


Figure 7 Original Image and Watershed Segmentation & Area Labeling Result

3) Görüntü İşleme ile ilgili olarak, fotoğraflar üzerinde geçmiş yıllarda düşünüp de yaptıramadığınız çok zor olmayan bir işlemi kısaca anlatınız (Böyle bir şey yoksa, P1 ve take-home'da olmayan bir işlem araştırıp onun üzerinden gidebilirsiniz). Derste öğrendiklerinizle birlikte araştırarak bu işlemi kodlayarak yapmaya çalışınız. İşlem ile ilgili görüntüleri rapora ekleyerek kodlamanın sonucunu yorumlayınız. (20 p)

Answer :

People often want to hide their impurities from the photos of themselves. Before the camera beauty filterings started to get popular, filtering photos had been done by using editor programs such as Adobe Photoshop to edit photos. But using an editor program to process each photo taken was taking a lot of time and expertise. To avoid editing photos, camera filterings became quite popular and they work very well.

But there is something these camera filterings cannot do, keeping the natural beauty after the filtering. So I have always thought of such an image processing method that could change minor impurities while not changing any other part of the photo to keep it the most natural looking. My thoughts that only eliminating the red acne scars on the face is enough to make the photo beautiful and natural.

There are numerous basic image filtering methods that can be used in combinations to achieve this acne removal filter.

Basic Version of AcneRemoval Filter:

For achieving this filter, OpenCV's inRange method can be used.
Firstly red color's lowest and highest hues are determined. Then image's color scale is tranformed from BGR to HSV. Then, pixels which are in range of the lowest and the highest red colors are masked. Finally, changing these masked pixel's color in from any red hue in between the lowest and highest to a predefined beige color.

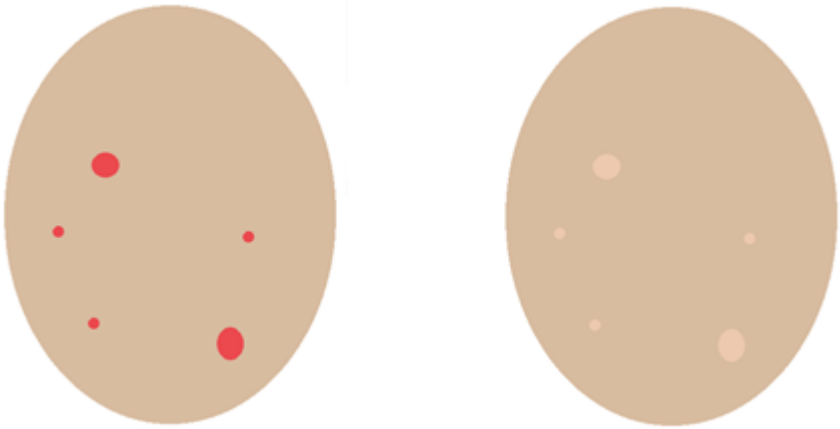


Figure 8 On Left Image Illustration of Acne on a Face,On the Right Image After Using Filter to Clear Them With the Basic Filtering

This filtering method is very simple and it is working in a way but there is a problem this filter cannot overcome, if there are red colored objects(hair, clothing...) or make up(red lipstick, blush...) on the photo this filter will turn them into a predefined beige color as well.

```
import cv2 as cv
import numpy as np

# Load the image and convert to HSV colourspace
image = cv.imread("acnepaint.png")
org = cv.imread("acnepaint.png")
hsv=cv.cvtColor(image,cv.COLOR_BGR2HSV)

# Define lower and uppper limits of red
red_low = np.array([20,20,50])
red_high = np.array([250,250,255])

# Mask image to only select browns
mask=cv.inRange(hsv,brown_lo,brown_hi)

# Change image to beige where red is found
image[mask>0]=(175,201,237)

cv.imshow("result",image)

cv.waitKey(0)
```

Figure 9 Python Code of Acne Remove Filter

Enhanced AcneRemoval Filter :

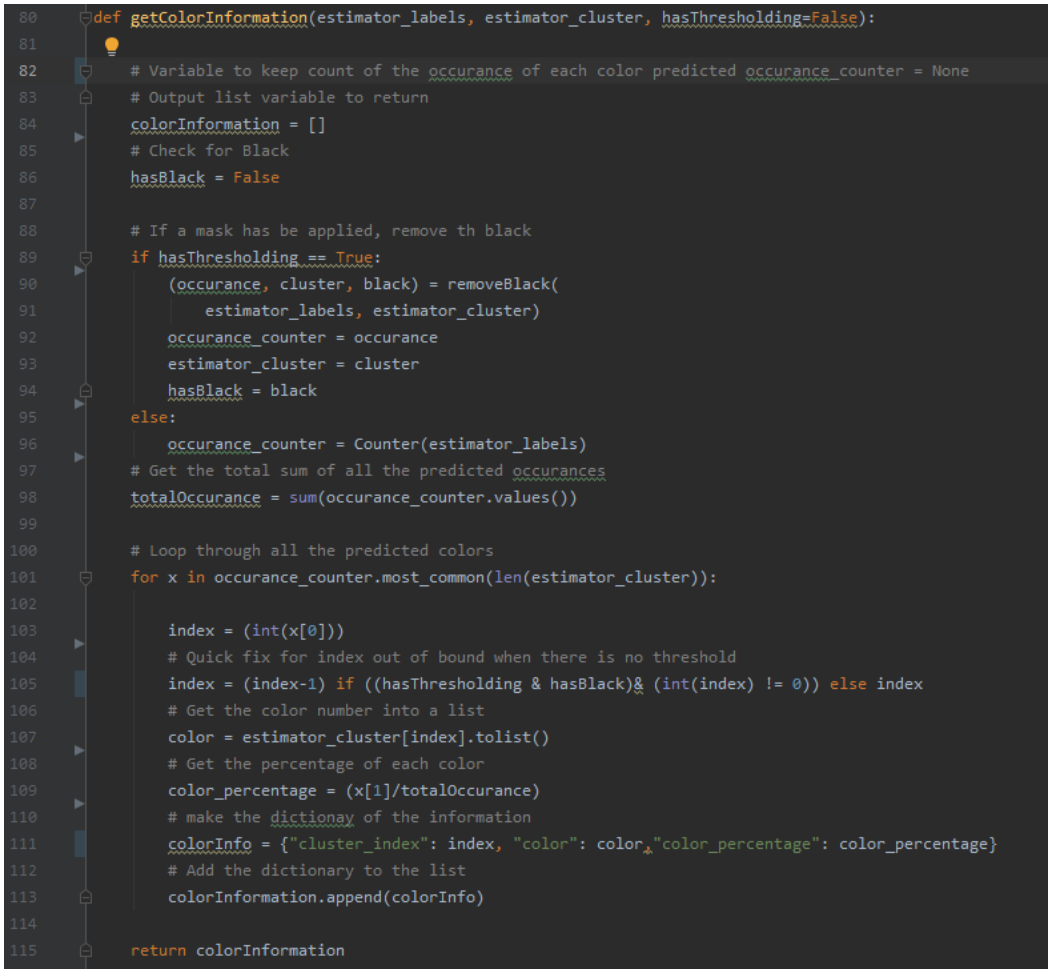
To enchance this basic filter which was replacing any red pixels' colors, a blob detection segmentation algorithm is used to mask only the acne shapes rather than anything "red".
Then, by using a skin detection segmentation all the colors and their percentages in the photo are calcualte. Result of this percentage and color is used to replace acne redness instead of replacing with a predefined beige color.
Finally applying the first filter algorithm on blob detection mask will result in a much better accuracy.

```python
from sklearn.cluster import KMeans
from collections import Counter
import imutils
from matplotlib import pyplot as plt
import cv2
import numpy as np


def detect_small_blob(dominant_color, image):
    # Convert image to Gray
    img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    cv2.imshow("Original Image", image)

    # Apply threshold to make image black and white
    ret, img = cv2.threshold(img, 125, 255, cv2.THRESH_BINARY)
    # Combine original image and the threshold mask with and operation to only extract only acne blobs
    masked = cv2.bitwise_and(image, image,mask = img)
    cv2.imshow("bitwise operations result", masked)

    hsv = cv2.cvtColor(masked, cv2.COLOR_BGR2HSV)
    # Define pure black scale
    black = np.array([0,0,0])

    # Mask image to only select browns
    mask = cv2.inRange(hsv, black, black)

    # Change image to max. percentage color

    dominant_color_red = dominant_color[2]
    dominant_color_green = dominant_color[1]
    dominant_color_blue = dominant_color[0]
    masked[mask > 0] = [dominant_color_red,dominant_color_green,dominant_color_blue]

    cv2.imshow("Result", masked)
    cv2.waitKey(0)
```

Figure 10 Enhanced Acne Removal Python Code Part 1

```
36
37  def extractSkin(image):
38      # Taking a copy of the image
39      img = image.copy()
40      # Converting from BGR Colours Space to HSV
41      img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
42
43      # Defining HSV Threadholds
44      lower_threshold = np.array([0, 48, 80], dtype=np.uint8)
45      upper_threshold = np.array([20, 255, 255], dtype=np.uint8)
46
47      # Single Channel mask,denoting presence of colours in the about threshold
48      skinMask = cv2.inRange(img, lower_threshold, upper_threshold)
49      # Cleaning up mask using Gaussian Filter
50      skinMask = cv2.GaussianBlur(skinMask, (3, 3), 0)
51      # Extracting skin from the threshold mask
52      skin = cv2.bitwise_and(img, img, mask=skinMask)
53      # Return the Skin image
54      return cv2.cvtColor(skin, cv2.COLOR_HSV2BGR)
55
56  def removeBlack(estimator_labels, estimator_cluster):
57
58      # Check for black
59      hasBlack = False
60      # Get the total number of occurance for each color
61      occurance_counter = Counter(estimator_labels)
62      # Quick lambda function to compare to lists
63      def compare(x, y): return Counter(x) == Counter(y)
64      # Loop through the most common occuring color
65      for x in occurance_counter.most_common(len(estimator_cluster)):
66
67          # Quick List comprehension to convert each of RBG Numbers to int
68          color = [int(i) for i in estimator_cluster[x[0]].tolist()]
69          # Check if the color is [0,0,0] that if it is black
70          if compare(color, [0, 0, 0]) == True:
71              # delete the occurance
72              del occurance_counter[x[0]]
73              # remove the cluster
74              hasBlack = True
75              estimator_cluster = np.delete(estimator_cluster, x[0], 0)
76              break
77      return (occurance_counter, estimator_cluster, hasBlack)
```

Figure 11 Enhanced Acne Removal Python Code Part 2



```
def getColorInformation(estimator_labels, estimator_cluster, hasThresholding=False):

    # Variable to keep count of the occurance of each color predicted occurance_counter = None
    # Output list variable to return
    colorInformation = []
    # Check for Black
    hasBlack = False

    # If a mask has be applied, remove th black
    if hasThresholding == True:
        (occurance, cluster, black) = removeBlack(
            estimator_labels, estimator_cluster)
        occurance_counter = occurance
        estimator_cluster = cluster
        hasBlack = black
    else:
        occurance_counter = Counter(estimator_labels)
    # Get the total sum of all the predicted occurances
    totalOccurance = sum(occurance_counter.values())

    # Loop through all the predicted colors
    for x in occurance_counter.most_common(len(estimator_cluster)):

        index = (int(x[0]))
        # Quick fix for index out of bound when there is no threshold
        index = (index-1) if ((hasThresholding & hasBlack)& (int(index) != 0)) else index
        # Get the color number into a list
        color = estimator_cluster[index].tolist()
        # Get the percentage of each color
        color_percentage = (x[1]/totalOccurance)
        # make the dictionay of the information
        colorInfo = {"cluster_index": index, "color": color,"color_percentage": color_percentage}
        # Add the dictionary to the list
        colorInformation.append(colorInfo)

    return colorInformation
```

Figure 12 Enhanced Acne Removal Python Code Part 3

```
118  def extractDominantColor(image, number_of_colors=5, hasThresholding=False):
119
120      # Quick Fix Increase cluster counter to neglect the black(Read Article)
121      if hasThresholding == True:
122          number_of_colors += 1
123
124      # Taking Copy of the image
125      img = image.copy()
126
127      # Convert Image into RGB Colours Space
128      img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
129
130      # Reshape Image
131      img = img.reshape((img.shape[0]*img.shape[1]), 3)
132
133      # Initiate KMeans Object
134      estimator = KMeans(n_clusters=number_of_colors, random_state=0)
135
136      # Fit the image
137      estimator.fit(img)
138
139      # Get Colour Information
140      colorInformation = getColorInformation(
141          estimator.labels_, estimator.cluster_centers_, hasThresholding)
142      return colorInformation
143
144
145  def plotColorBar(colorInformation):
146      # Create a 500x100 black image
147      color_bar = np.zeros((100, 500, 3), dtype="uint8")
148
149      top_x = 0
150      for x in colorInformation:
151          bottom_x = top_x + (x["color_percentage"] * color_bar.shape[1])
152
153          color = tuple(map(int, (x['color'])))
154
155          cv2.rectangle(color_bar, (int(top_x), 0),
156                        (int(bottom_x), color_bar.shape[0]), color, -1)
157          top_x = bottom_x
158      return color_bar
```

Figure 13 Enhanced Acne Removal Python Code Part 4

```
original_image =  cv2.imread("acnepaint.png")
image = original_image

# Resize image to a width of 250
image = imutils.resize(image, width=250)

# Apply Skin Mask
skin = extractSkin(image)

# Find the dominant color. Default is 1 , pass the parameter 'number_of_colors=N' where N is the specified
dominantColors = extractDominantColor(skin, hasThresholding=True)

# Show in the dominant color information
dominant_color = dominantColors[0]['color']

# Show in the dominant color as bar
colour_bar = plotColorBar(dominantColors)
plt.subplot(3, 1, 3)
plt.axis("off")
plt.imshow(colour_bar)
plt.title("Color Bar")

plt.tight_layout()
plt.show()

detect_small_blob(dominant_color, original_image)
```

Figure 14 Enhanced Acne Removal Python Code Part 5 (Main Execution Part)

These are the resulting pictures of the enhanced AcneRemoval Filter.

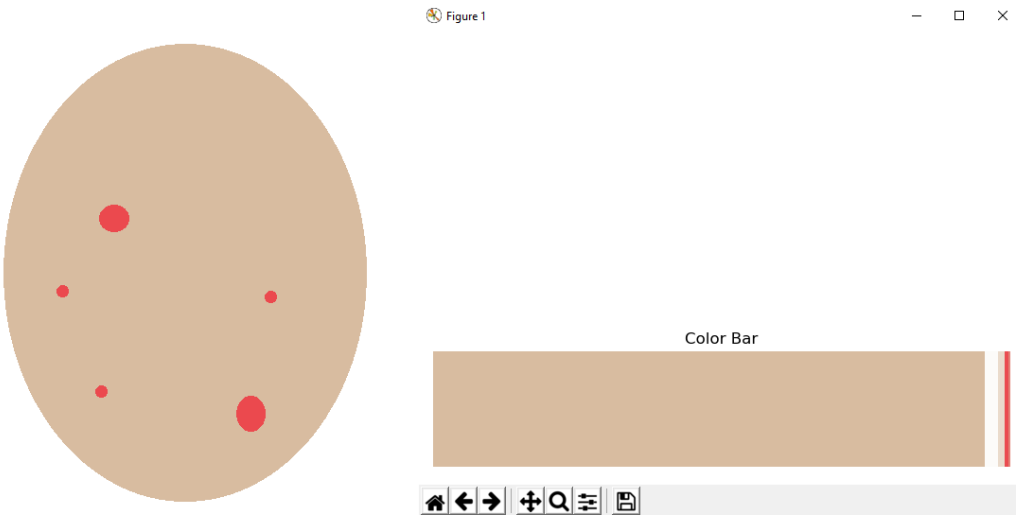


Figure 15 Original Image and Color Percentage Bar of the Image

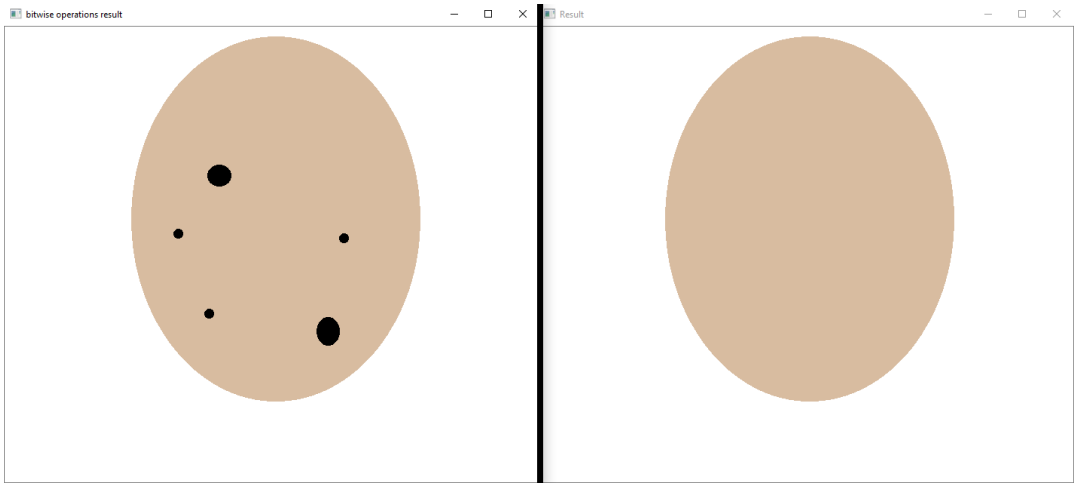


Figure 16 Blob Detection Mask Represented with Pure Black([0,0,0,]) Replaced with Maximum Percentage Color from Color Bar

4) Aşağıdaki kavramları araştırarak kısaca açıklayınız (20 p):

Answer :

a. SIFT (Scale-Invariant Feature Transform)

The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images.
SIFT keypoints of objects are first extracted from a set of reference images[and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors.
From the full set of matches, subsets of keypoints that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches. The determination of consistent clusters is performed rapidly by using an efficient hash table implementation of the generalised Hough transform. Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed model verification and subsequently outliers are discarded. Finally the probability that a particular set of features indicates the presence of an object is computed, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

Figure 17 On the Left Scale Space Extrema Detection, On the Middle SIFT Algorithm Discards Low Contrast Points, On the Right SIFT Algorithm Filters Out Dots on Edges

b. Corner Detection, Harris Corner Detector

A corner is a point whose local neighborhood stands in two dominant and different edge directions. In other words, a corner can be interpreted as the junction of two edges, where an edge is a sudden change in image brightness. Corners are the important features in the image, and they are generally termed as interest points which are invariant to translation, rotation and illumination. Although corners are only a small percentage of the image, they contain the most important features in restoring image information, and they can be used to minimize the amount of processed data for motion tracking, image stitching, building 2D mosaics, stereo vision, image representation and other related computer vision areas.

As Harris operator is the most simple, efficient and reliable for use in corner detection. This methodology is closely associated with and based on the local structure matrix. Harris corner detector provides good repeatability under changing illumination and rotation, and therefore, it is more often used in stereo matching and image database retrieval. Although there still exists drawbacks and limitations, the Harris corner detector is still an important and fundamental technique for many computer vision applications.

```python
import numpy as np
import cv2 as cv

filename = 'chessboard.png'
img = cv.imread(filename)
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

gray = np.float32(gray)
dst = cv.cornerHarris(gray,2,3,0.04)

#result is dilated for marking the corners, not important
dst = cv.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]

cv.imshow('dst',img)
if cv.waitKey(0) & 0xff == 27:
    cv.destroyAllWindows()
```

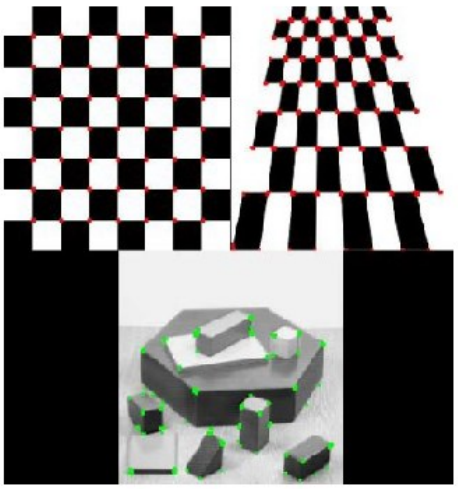


Figure 18 Corner Detection Harris Python Code and Resulting Image

c. Hough Transform

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.
The classical Hough transform was concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses.

d. Laplacian of Gaussian

The Laplacian is a two dimensional isotropic measure of the second spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter in order to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single graylevel image as input and produces another graylevel image as output

Self Evaluation Table:

Question Number	Question Point	Estimated Point	Explanation No.
Question 1	20	20	1
Question 2	20	20	2
Question 3	20	20	3
Question 4	20	20	4
Report	10	10	5
Self-Evaluation	10	10	6
Total	100	100	

Explanations:

1) Comprasion table is simple but gives full insight to choose between one of them for a person who has no prior information about the three programs before. It meets the requirements of the question.

2) As requested two different segmentations used one is Thresholding Binary and other is Watershed, their full Python source code and two different picture's before-after versions are added to the report to supply let the readers see the result of the codes.

3) An actual filtering idea implemented in a basic way and added to report. Then this basic version is enchanced to a version which is implemented and added to report as well. Before and after pictures along with Python codes are added.

4) All the questions are answered by explaining the algorithm's steps relatively shortly and example code, algorithm pictures are added to let readers visualize the algorithms.

5) Report contains all the neccesary answers, pictures and captions and styled in a way to ease readers reading it.

6) I have learned a lot by completing this take home. Firstly I have learned almost six or more methods about image processing. In the process of learning and completing this document I could practice with Python image processing which improved my coding abilities. Most importantly I have created a project from my personal idea. Thus is strenghten my creativity.