

# T.C. KÜTAHYA DUMLUPINAR ÜNİVERSİTESİ



Dönem Projesi  
Python Kullanılarak OpenCv İle Projeler

Hazırlayan  
Tuğrul Şahin Kök  
201851441053

# Python ile Görüntü İşleme (OpenCV)

## **OpenCV Nedir?**

OpenCV, gerçek zamanlı bilgisayar vizyonunu amaçlayan bir programlama fonksiyon kütüphanesidir. Kütüphane çapraz platformdur ve açık kaynaklı BSD lisansı altında kullanım için ücretsizdir. Kütüphane aynı zamanda makine öğrenmesinde desteklemektedir.

Cplusplus diliyle geliştirilmektedir. Kütüphane içerisinde 500'den fazla algoritma ve bu sayının yaklaşık 10 katı kadar da fonksiyon yer almaktadır.

TensorFlow, Torch/PyTorch , Caffe gibi derin öğrenme frameworklerini destekler. Linux, Windows, macOS, FreeBSD gibi masaüstü işletim sistemlerinde ve iOS , Android gibi mobil işletim sistemlerinde çalışabilmektedir.

## **OpenCV ile Neler Yapılabilmektedir?**

- Sabit görselleri , videoları veya webcam görüntülerini okuyup kaydetme
- Yüzleri ve yüz özelliklerini saptama
- Görseller üzerindeki yazıları saptama (plaka, tabela vb.)
- Nesne saptama , sayma ve takibi
- Yüzlerdeki duyguları anlama
- Hareket takibi gibi işlemler yapılabilmektedir.

## **OpenCV Hangi Yazılım Dillerini Destekler?**

- C++
- Java
- Python
- Android SDK

## Resim Açma , Resim Okuma , Resim Yazma

Python dosyası oluşturduktan sonra içerisine openCV kütüphanesini dahil ediyoruz.

```
import cv2
```

Resmimizi okutmak için bir tane resim değişkeni oluşturuyoruz.

```
resim = cv2.imread("Resimler/seinfeld.jpg")
```

Resmimizi bu şekilde okutursak resmimizi olduğu gibi yani renkli olarak açacaktır.Ancak ikinci parametre olarak “0” (sıfır) değerini verirse resmimiz gri olacaktır.Gri olmasının nedeni tüm pikseller 0-255 arasında bir değer alacak ve tek bir renk kanalından oluşacaktır.Bu konuya ileride daha detaylı değinilecektir.

Resmimizi ekranda göstermek için :

```
cv2.imshow("Seinfeld Resmi",resim)
```

Burada ilk parametre açılacak pencerenin başlığını ikinci parametre ise resmimizi okuttuğumuz resim değişkenimizi belirtir.

Resim açıldığında hemen kapanmaması için :

```
cv2.waitKey(0)
```

waitKey() fonksiyonu resim açıldığında ekranda ne kadar süre kalacağını belirten bir fonksiyondur.Parametreside delay yani bekleme süresidir.Ancak biz parametre olarak “0”(sıfır) değerini verirse, biz herhangi bir tuşa basmadan resim penceremiz açık kalacaktır.

Açılan tüm pencerelerin kapatılması için:

```
cv2.destroyAllWindows()
```

komutu kullanılır.

Program Çıktısı :



## OpenCV Resmimizi Nasıl Yorumlar?

OpenCV 'de resimlerimizi bir numpy dizisi olarak tutar bunu görmek için:

```
print(type(resim))
```

kodunu çalıştırdığımızda bize

```
<class 'numpy.ndarray'>
```

değerini dönmektedir. Buradan resimlerimizin bir numpy array'i olarak saklandığını görebiliriz

## Renk Sistemleri

Bilgisayarların anlamlandırdığı belirli renk sistemleri ve renk uzayları (color space) bulunmaktadır. Bunlardan bazıları RGB, BGR, HSV ve YCbCr'dır.

### RGB(Red, Green, Blue)

Çok yaygın kullanılan bir renk sistemidir. İngilizce anlamı olarak red(kırmızı), gree(yeşil), blue(mavi) renklerinin baş harflerinden oluşturulmuştur.

Bu sistem, bilgisayar ve televizyon gibi elektronik cihazların algılama ve sunum mekanizmalarında; ayrıca fotoğrafçılık alanında yaygın şekilde kullanılmaktadır. Birbirine çok yakın biçimde konumlandırılan temel renk kaynakları birbirini etkileyerek insan gözünün algılayabildiği renk dizilimindeki bileşimleri oluşturur.

Renkleri belirtmek için tamsayılar kullanılır. 8bitlik tam sayılar kullanıldığında aynı rengin 256, 16bitlik sayılar kullanıldığında 65536 farklı tonunu elde etmek mümkündür.

OpenCV'de genellikle 8bitlik renkler kullanılır.

## OpenCV ve BGR Mantığı

RGB renk sistemiyle temelde aynıdır. Sadece Kırmızı ve mavi renkler sıralamada birbirlerinin yerine geçmişlerdir.

OpenCV'nin ana renk sistemi BGR(blue, green, red)'dir.

OpenCV 'de resmimizi print fonksiyonu ile yazdırdığımızda bize bir matris döner. Bu matris piksellerin BGR renk modelindeki renk skalasını döner.

```
print(resim)
```

Blue	Green	Red
[ 12	15	20]
[ 12	15	20]
[11	14	19]
...		

Bize bu şekilde çok uzun bir matris değerleri dönmektedir. Her satır bir pikseli ifade etmektedir ve değerler 3 farklı kanalda gösterilir (R G B). Bu şekilde mavi, yeşil ve kırmızı kullanarak tüm renkler elde edilebilir. Her bir renk skalası için değerler 0 – 255 arasındadır. Yani her bir kanal için 256 farklı ton vardır.

**Peki bir piksel için kaç farklı renk kombinasyonu vardır?**

$256 * 256 * 256 = 16.777.216$  farklı renk kombinasyonu vardır.

## Resimler Neden Gri Tona Çevirilir?

OpenCV’de görüntü işleme çalışılırken resimler üzerinde gri ton kullanılır. Bunun nedeni resimlerimizin 3 farklı kanalda değil de tek bir kanaldan üretilmesidir. Bu hem resmimizin boyutunu küçültür. Hem de üzerinde çalışmamız gereken pikselleri azaltır. Resim renkli iken üzerinde işlem yapacağımız 3 farklı kanal olacağından iş yükümüz artmaktadır. Bunu önlemek için tek bir kanal üzerinden çalışılır ve 0 – 255 değerleri arasında çalışma yapılır. Burada 0 -> siyah, 255 -> beyazdır. Resim pikselleri sadece bu iki siyah ve beyaz skalası arasında değer alacağı için gri olur.

## Resim Verileri Hangi Türde Saklanıyor?

Bunu öğrenmek için :

```
print(resim.dtype)
```

komutunu çalıştırırız ve bize;

uint8(unsigned integer 8 bit ) değerini döner bu ne demektir:

$00000000 = 0(\text{min})$

$11111111 = 255(\text{max})$

## Resmimizin Boyutlarını Öğrenmek ?

Resmimizin yükseklik , genişlik ve renk kanalı değerlerini öğrenmek için :

```
print(resim.shape)
```

komutu çalıştırılır ve bize

(500, 390, 3) şeklinde değer döner.Burada birinci değer yükseklik ikinci değer genişlik üçüncü değer ise kaç tane renk kanalı kullanıldığıdır.Eğer resmimizi gri tonda açsaydık bize (500, 390) değerini dönecekti.

## OpenCv'de Yazı Yazmak

OpenCV kütüphanesinin doğal olarak desteklediği font sayısı kısıtlıdır.Üstelik bu fontlar sadece ASCII karakterleri desteklemektedir.Yani Türkçe karakter kullanarak yazı yazmamız durumunda desteklenmeyen , Türkçe'ye özgü karakterlerin yerinde soru işareti bulunacaktır.

OpenCV de yazılar putText() fonksiyonu ile yazılır.putText () fonksiyonun özelliklerini inceleyelim:

img=cv.putText(img, text, org, fontFace, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]]) parametre tanımları bu şekildedir.Şimdi de bunların anlamlarına bakalım

### Parametreler

<b>img</b>	Arkaplan görüntüsüdür.Hangi resmin üzerine yazı yazılacaksa o resmi ifade eder.
<b>text</b>	String türünde yazıyı ifade eder Türkçe karakter kullanılamaz.Kullanılırsa Türkçe karakterin yerine soru işareti gelecektir.
<b>org</b>	Resmin sol alt köşesini piksel cinsinden ifade eder.
<b>fontFace</b>	Sadece HERSHEYFONT ismindeki fontları tanır.Bu fontlar 8 adettir.
<b>fontScale</b>	Yazının büyüklüğünü ifade eder.Çarpan olarak ifade edilebilir.Normal ve varsayılan değeri 1'dir.
<b>color</b>	Yazının rengini belirtir.BGR renk uzayına göre olan renkleri içerir.
<b>thickness</b>	Harflerin çizgi kalınlığını ifade eder.
<b>lineType</b>	Yazı tipini ifade eder.
<b>bottomLeftOrigin</b>	Değer eğer False ise sol alt köşeye göre yazıyı düzgün bir biçimde yazar.True değerini verirse yazıyı ters(tepetaklak) yazar.

HERSHEY Fontları Nelerdir ?	
<b>FONT_HERSHEY_SIMPLEX</b> Python: cv.FONT_HERSHEY_SIMPLEX	normal boyutta sans-serif yazı tipi
<b>FONT_HERSHEY_PLAIN</b> Python: cv.FONT_HERSHEY_PLAIN	küçük boyutta sans-serif yazı tipi
<b>FONT_HERSHEY_DUPLEX</b> Python: cv.FONT_HERSHEY_DUPLEX	normal boyutta sans-serif yazı tipi (FONT_HERSHEY_SIMPLEX'ten daha karmaşıktır.)
<b>FONT_HERSHEY_COMPLEX</b> Python: cv.FONT_HERSHEY_COMPLEX	normal boyutta sans-serif yazı tipi
<b>FONT_HERSHEY_TRIPLEX</b> Python: cv.FONT_HERSHEY_TRIPLEX	normal boyutta sans-serif yazı tipi (FONT_HERSHEY_COMPLEX'ten daha karmaşıktır.)
<b>FONT_HERSHEY_COMPLEX_SMALL</b> Python: cv.FONT_HERSHEY_COMPLEX_SMALL	FONT_HERSHEY_COMPLEX'in küçük versiyonudur.
<b>FONT_HERSHEY_SCRIPT_SIMPLEX</b> Python: cv.FONT_HERSHEY_SCRIPT_SIMPLEX	El-yazı stili yazı tipi
<b>FONT_HERSHEY_SCRIPT_COMPLEX</b> Python: cv.FONT_HERSHEY_SCRIPT_COMPLEX	FONT_HERSHEY_SCRIPT_SIMPLEX'in daha karmaşık bir şekli
<b>FONT_ITALIC</b> Python: cv.FONT_ITALIC	Fontların italik yazılıp yazılmayacağını ifade eder sayısal değeri 16'dır.

## Yazı Tipleri Nelerdir ?(lineType)

Enumerator	
<b>FILLED</b> Python: cv.FILLED	
<b>LINE_4</b> Python: cv.LINE_4	4-bağlı hat
<b>LINE_8</b> Python: cv.LINE_8	8-bağlı hat
<b>LINE_AA</b> Python: cv.LINE_AA	Kenar yumuşatıcı hat

### 3-YazıYazmak.py

```
import cv2
import numpy as np

fontlar = [
    'FONT_HERSHEY_SIMPLEX', 'FONT_HERSHEY_PLAIN',
    'FONT_HERSHEY_DUPLEX', 'FONT_HERSHEY_COMPLEX',
    'FONT_HERSHEY_TRIPLEX', 'FONT_HERSHEY_COMPLEX_SMALL',
    'FONT_HERSHEY_SCRIPT_SIMPLEX',
    'FONT_HERSHEY_SCRIPT_COMPLEX'] #8 tane fontumuzla yazı yazmak için
8 tane yazı oluşturduk
imaj = np.ones((720,780,3),np.uint8)*255#beyaz imaj (resim)
for j in range(8):#font numaraları 8 tane olduğu için sırasıyla 0'dan
başlayarak farklı fontlar oluşturacak.Bu değerler 0-7 arasında toplam
8 tanedir.
    cv2.putText(imaj,fontlar[j],(20,40+j*40),j,1.1,(0,0,0),1)
'''
    putText fonksiyonun parametreleri
    birincisi Hangi görsele ekleyeceğimizi
    ikincisi Ne yazacağımızı
    üçüncüsü Görselin hangi kısmına ekleyeceğimizi
    dördüncüsü Hangi formatta yazacağımızı
    beşincisi Skalamızı ekliyoruz (boyut)
    altıncısı Hangi renk ile yazacağımızı
    yedincisi de kalınlık belirtir
    biz burada opencv'nin bize sunduğu tüm fontlar ile yazı yazdık
bunu for döngüsü içerisinde yaptık
'''

italik = 16 #yazıyı italik yapmak için fontun değerine 16
ekliyoruz.Bazı font stilleri italik
    # yazıyı desteklememektedir.
for j in range(8):

cv2.putText(imaj,fontlar[j]+'(italik)',(20,400+j*40),j+italik,1.1,(0,0
,0),1)

cv2.imshow('imaj',imaj)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



## Program Çıktısı :



## Türkçe Karakter Destekli Yazı Yazmak

OpenCV'nin desteklediği fontlar sadece ASCII karakterlerine izin vermekte ama Türkçe karakter yazdırabilmek için pillow kütüphanesi kullanarak bunu çözebiliyoruz. Pillow kütüphanesi açık kaynak kodlu bir grafik işleme kütüphanesidir. Aslında burada yaptığımız pillow kütüphanesiyle yazı şeklinde grafik çizmek bu yüzden Türkçe karakterleri oluşturabiliyoruz.

### 4-Utf8-DestekliYaziYazmak.py

```
import cv2
import numpy as np
from PIL import Image, ImageDraw, ImageFont

# Türkçe karakterleri de yazabilmek için pillow kutuphanesini kullanıyoruz
# pillow kutuphanesi bir görüntü işleme kutuphanesidir. Python da görüntü işlemleri
# yapılabilmesi için oluşturulmuştur.
def print_utf8_text(image, text, fontName='DejaVuSerif',
                    color=(0,0,0), yer=(10,10), boy=48):
    font = ImageFont.truetype(fontName, boy) # font adı ve boyutu
    pilImg = Image.fromarray(image) # imajı pillow moduna çevir
    #Image opencv de BGR renk uzayını kullanıyor ancak PIL RGB uzayını kullandığı için onun
    #cevirimini yapıyoruz.
    draw = ImageDraw.Draw(pilImg) # imajı hazırla
    #ImageDraw basit iki boyutlu grafik oluşturulmasını sağlar
    #burada yapılan pilimg'i çizmek
```

```

draw.text((yer[0], yer[1]), text,
          fill=(color[0], color[1], color[2],0),font = font)
image = np.array(pilImg) # resmi tekrar opencv'nin kullanabilecegi moda (BGR) ceviriyoruz
return image
...

PIL.ImageDraw.Draw.text(xy, text, fill=None, font=None)
Parameters:
    xy -metnin sol ust kosesi
    text - cizilecek metin
    fill - metin icin kullanılacak renk
    font - ImageFont ornegi
...

if __name__ == "__main__":

    imaj = np.ones((400, 700, 3), dtype=np.uint8) * 255#beyaz bir zemin olusturak icin

    fontName='verdana.ttf'; renk = (0, 0, 0); yer = (10,10); boy = 64#sirayla deger atama
    islemleri.
    #ayni satirda yazdigimiz icin aralara ; koyduk
    imaj = print_utf8_text(imaj, "Ağaç ", fontName,
                          renk, yer, boy)
    #fonksiyonumuzun icerisine parametrelerimizi gonderiyoruz
    # ve ayni islemleri farkli yazilar yazmak icin tekrarliyoruz
    fontName = 'tahoma.ttf'
    renk = (0, 0, 255); yer = (100,120); boy = 36
    imaj = print_utf8_text(imaj, "Tuğrul Şahin Kök ", fontName,
                          renk, yer, boy)
    fontName = 'times.ttf'; renk = (128, 0, 0)
    yer = (30,200); boy = 48
    imaj = print_utf8_text(imaj,
                          "LACİVERT ", fontName, renk, yer, boy)

    fontName = 'times.ttf'; renk = (0, 255, 0)
    yer = (10,300); boy = 42
    imaj = print_utf8_text(imaj, "Ömer ", fontName, renk, yer, boy)

    cv2.imshow('beyaz fon', imaj)
    cv2.moveWindow('beyaz fon', 10, 10)# pencerenin x,y duzleminde nerede acilacagini belirtir

    cv2.waitKey(0)
    cv2.destroyAllWindows()

```

Program Çıktısı:



## Görseller ve Matris İşlemleri

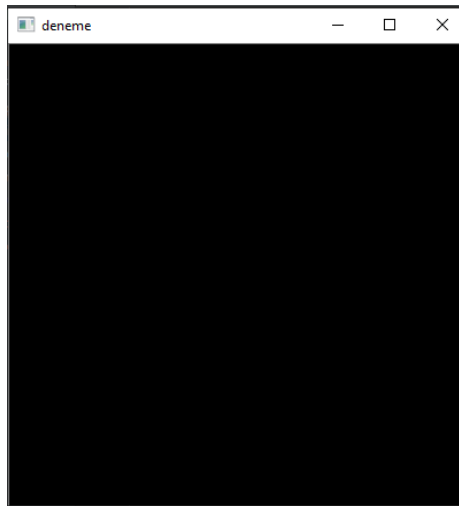
OpenCV’de resimler birer matris olarak görülür. Matris işlemleri numpy kütüphanesi ile yapılır. Oluşturacağımız görselleri matris ile oluşturur

5-bos\_resim.py

```
import numpy as np
import cv2

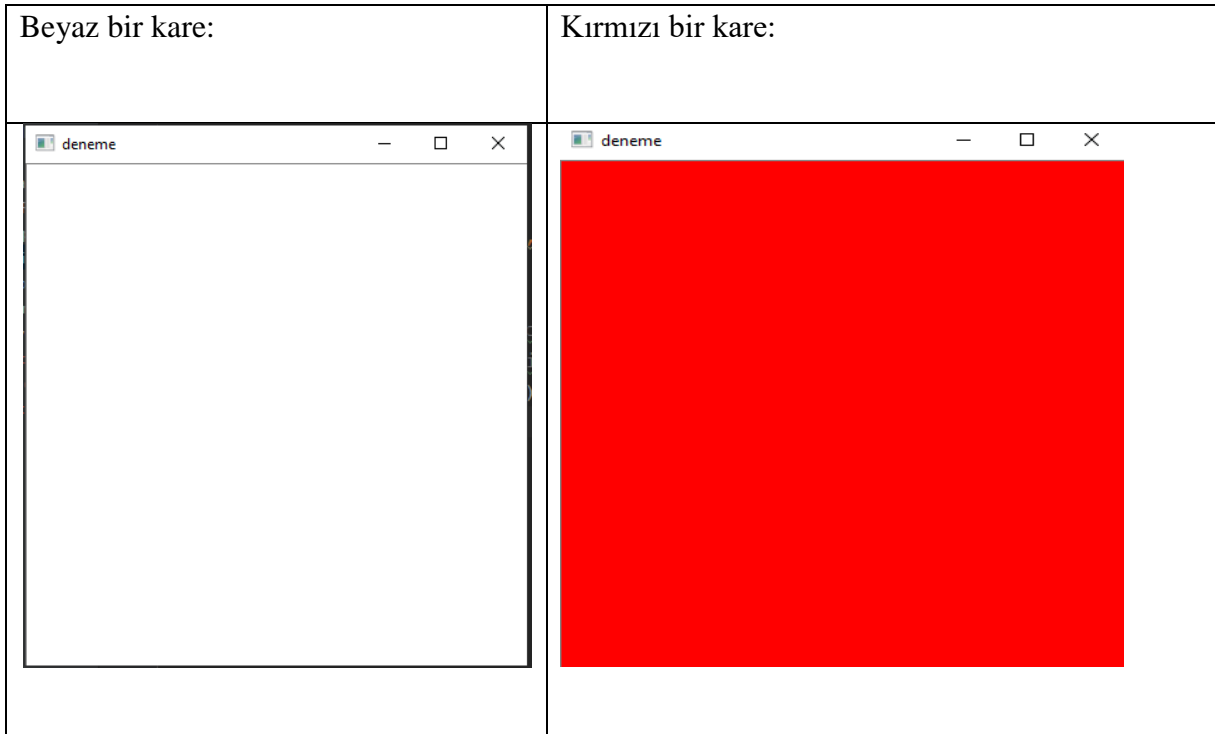
deneme = np.zeros((400,400,3),dtype=np.uint8)
#deneme[:] = (255,255,255) beyaz piksel degerleri
#deneme[:] = (0,0,255) kirmizi piksel degerleri
#400x400 piksel boyutlarında 3 renk kanalına sahip bir siyah pencere olusturur.
#bu resmi bir numpy matrisiyle olusturup icerisini sifir ile
doldurduk.(np.zeros)
cv2.imshow('deneme',deneme)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



Bu 400x400 piksellik 3 renk kanalına sahip siyah bir penceredir. Bu görüntüyü bir numpy matrisiyle oluşturup içeriğini 0(sıfır)’larla doldurduk.(np.zeros)

Bu siyah görüntüyü eğer beyaza çevirmek istersek o zaman görselimizin her bir pikselindeki renk bilgisini beyaza karşılık gelen (255,255,255) demetiyle doldurmamız gerekir.



## Renk Kanalları

BGR renk uzayına sahip olan OpenCV’de görüntümüzün 3.boyutu renk kanallarını tutar.Herhangi bir resmi oluşturmak için bu renk kanallarının birlikte kullanılması gerekmektedir.Renk kanallarını birer filtre olarak düşünürsek 3 tane renk kanalımız(BGR) üst üste gelince resmimizi renkli biçimde görürüz.Bu renk kanallarını ayrı ayrı görüntülemekte mümkün.Yani resmimizi mavi , yeşil ve kırmızı filtreler ile gösterebilir.Bunu yapmak için bir renk kanalını göstermek için diğer renk kanallarının tüm değerlerini sıfırlamak olacaktır.

6-renk\_kanallari.py

```
import cv2

deltax=0
deltay=0

img = cv2.imread('../Resimler/monalisa.jpg')
m = img.copy()
m[:, :, 1]=0
m[:, :, 2]=0
# mavi renk filtresini elde etmek için yeşil ve kırmızı renk kanallarını sıfıra
esitliyoruz

y = img.copy()
y[:, :, 0]=0
y[:, :, 2]=0
# Yeşil renk filtresini elde etmek için mavi ve kırmızı renk kanallarını sıfıra
esitliyoruz
```

```

k = img.copy()
k[:, :, 0] = 0
k[:, :, 1] = 0
# Kırmızı renk filtresini elde etmek için yeşil ve kırmızı renk kanallarını
sıfıra eşitliyoruz
print(img.shape) # resmin boyutlarını verir -> yükseklik, genişlik, renk kanalları

# renk kanallarının hepsinin aynı anda olması resmimizin orijinal halinin ortaya
çıkmasını sağlıyor
cv2.imshow("Orijinal", img); cv2.moveWindow('Orijinal', 10, 10)
...

    Yükseklik, genişlik ve kanal sayısına img.shape
    ile erişebiliriz: Yükseklik indeks 0'da,
    Genişlik indeks 1'de; ve indeks 2'deki kanal sayısı.
...

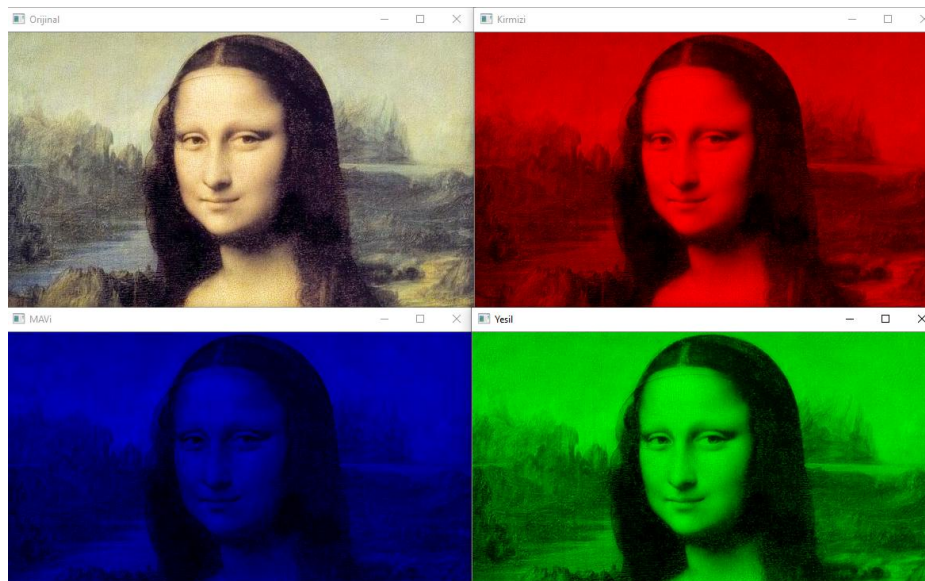
cv2.imshow('MAVi', m)
cv2.moveWindow('MAVi', 10, img.shape[0] + deltay)
# moveWindow(x = 10, ya = yukseklik + deltay)

cv2.imshow('Kirmizi', k)
cv2.moveWindow('Kirmizi', img.shape[1] + deltax, 10)
# moveWindow(x = genislik + deltax , y = 10)

cv2.imshow('Yesil', y)
cv2.moveWindow('Yesil', img.shape[1] + deltax, img.shape[0] + deltay)
# moveWindow(x = genislik + deltax , yukseklik + deltay)
# moveWindow() ile pencerelerin acilma konumlarını ayarlıyoruz
cv2.waitKey(0)
cv2.destroyAllWindows()

```

Program Çıktısı:



## Yeniden Boyutlandırmak

OpenCV kullanırken bazen resimleri ihtiyacımız çerçevesinde yeniden boyutlandırmamız gerekebilir. Bunu en-boy oranını koruyarak yapabileceğimiz gibi bu bir zorunluluk değildir. Resimlerimizi istediğimiz boyutlarda tekrar boyutlandırabiliriz.

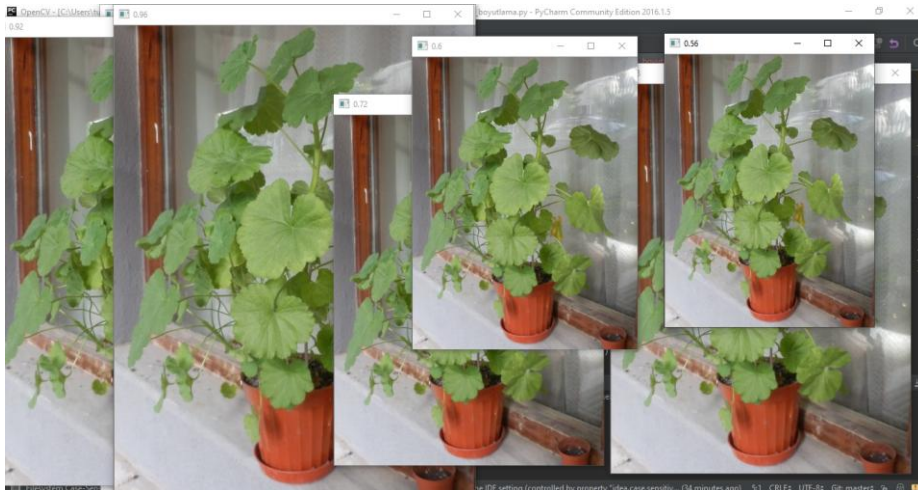
7-yeniden\_boyutlama.py

```
import cv2
import random

imaj = cv2.imread('../Resimler/sardunya2.jpg')
cv2.imshow('imaj', imaj)

oran = 0.8
imajlar=[]#resimleri bir liste icerisinde tutucuz
for j in range(10):
    oran = random.randint(1,25)/25 # 0 ile 1 arasında bir sayi olusturur
    rx = int(imaj.shape[1]*oran) #Yeni imajın genisligi
    ry = int(imaj.shape[0]*oran) #yeni imajın yukseligi
    x = random.randint(100,1600)#100-1600 arasında rastgele x koordinati
    y = random.randint(100,800)#100-800 arasında rastgele y koordinati
    imajlar.append((str(oran),cv2.resize(imaj,(rx,ry))))
    #listeye ekleme fonksiyonu,
cv2.resize(orjinal_imaj,(yeni_yukseklik,yeni_genislik))
cv2.imshow(imajlar[j][0],imajlar[j][1])
    #pencere ismi olarak kullanılan orani ve yeni olusturan imajın kendisini
    kullanıyoruz
    cv2.moveWindow(imajlar[j][0],x,y)
    #bir ust satırda kullandığımız isimdeki pencerenin x ve y random
    komutlarını alıyoruz
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



## Maskeleme

Maskeleme resim birleştirme işlemlerinde kullanılan bir araçtır. Maske siyah ve beyazdan oluşan bir görüntüdür. Maske siyah ve beyazdan oluşan bir görüntüdür. Uygulama işlemi gerçekleştirildiğinde beyaz olan pikseller işleme sokulurken siyah olan pikseller işleme alınmazlar.

Örneğimiz de monalisa.jpg dosyamızın maskesini yarısı beyaz diğer yarısı siyah olacak şekilde oluşturduk. Resmimizi ve maskemizi `bitwise_and()` komutu ile işleme soktuğumuzda resmimizin yarısı monalisa diğer yarısı ise siyah piksellerden oluşturduk.

8-maskeleme.py

```
import cv2
import numpy as np

deltax = 0
deltay = 0

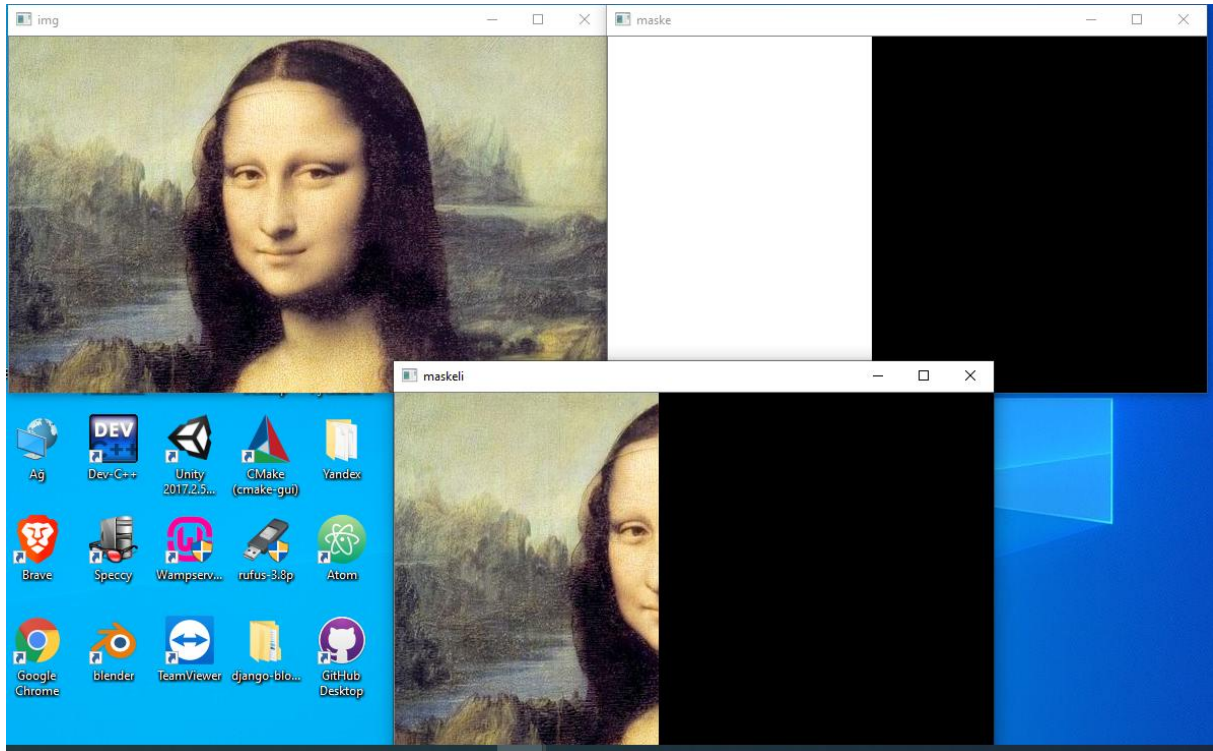
img = cv2.imread("../Resimler/monalisa.jpg")
maske = np.ones(img.shape, dtype="uint8")*255 #beyaz bir resim olusturduk
#img.shape resmin boyutu kadar bir pencere olusturmamizi sagliyor
maske[:,260:] = [0,0,0]#resmin ortasina kadar siyah alan olusturmak

maskeli = cv2.bitwise_and(img,maske)#maskenin siyah bolgeleri siyah kalir beyaz
kisimlari diger resim ile doldurulcak
#cv2.bitwise_and(ilk girdi dizisi,ikinci girdi dizisi)
#ilk parametre okuttugumuz resmimiz ikincisi ise olusturdugumuz siyah maske
#bitwise_and iki resmin bitsel birlesimini saglar
cv2.imshow('img',img)#resim gosterme islemi
cv2.moveWindow('img',10,10)#resmin nerede cikagi
cv2.imshow('maske',maske)#maske gosterimi
cv2.moveWindow('maske',img.shape[1]+deltax,10)#resmin nerede cikacagi
cv2.imshow('maskeli',maskeli)#maske gosterimi
cv2.moveWindow('maskeli',380,img.shape[0]+deltay)#maskeli resmin nerede cikacagi

cv2.waitKey(0)
cv2.destroyAllWindows()
```



## Program Çıktısı:



## Resim Ekleme

OpenCV’de herhangi bir resim üzerine başka bir resim ekleme işlemi yapabiliyoruz. Uygulama olarak yapmak istediğimiz maske ve ters maske oluşturarak arkaplan resmimizin ortasına monalisa.jpg dosyamızı oturtmak. Kodlarımızı inceledikten sonra bunu adım adım gözlemleyelim.

9-resim\_ekleme.py

```
import cv2
import numpy as np
bekle = True

#tusa bastigimizda false 'a doner
arkaplan = np.ones((800,800,3),dtype=np.uint8)*255#800x800 boyutlarında beyaz
arkaplan
if bekle:
    cv2.imshow('arkaplan',arkaplan)#arkaplani goster
    cv2.waitKey(0)
#Arkaplanın ortasına 360piksel yarıçaplı siyah bir daire çizimi
maske = cv2.circle(arkaplan,(400,400),360,(0,0,0),-1)
#beyaz arkaplanın üzerine 400x400 piksel ve çapı 360px olan bir siyah cember
cizdiriyoruz
```



```

if bekle:
    cv2.imshow('maske',maske)
    cv2.waitKey(0)

#ters maske islemi
tersmaske = (255-maske)#bu islem maskedeki 0'lari 255'e , 255'leri 0'a ceviriir yani
beyazlar siyah,siyahlar beyaz olur
if bekle:
    cv2.imshow('tersmaske',tersmaske)
    cv2.waitKey(0)
#resim yukleme
img = cv2.imread('../Resimler/monalisa(800x800).jpg')
if bekle:
    cv2.imshow('img',img)
    cv2.waitKey(0)
#ters maskeyi img ye uygula

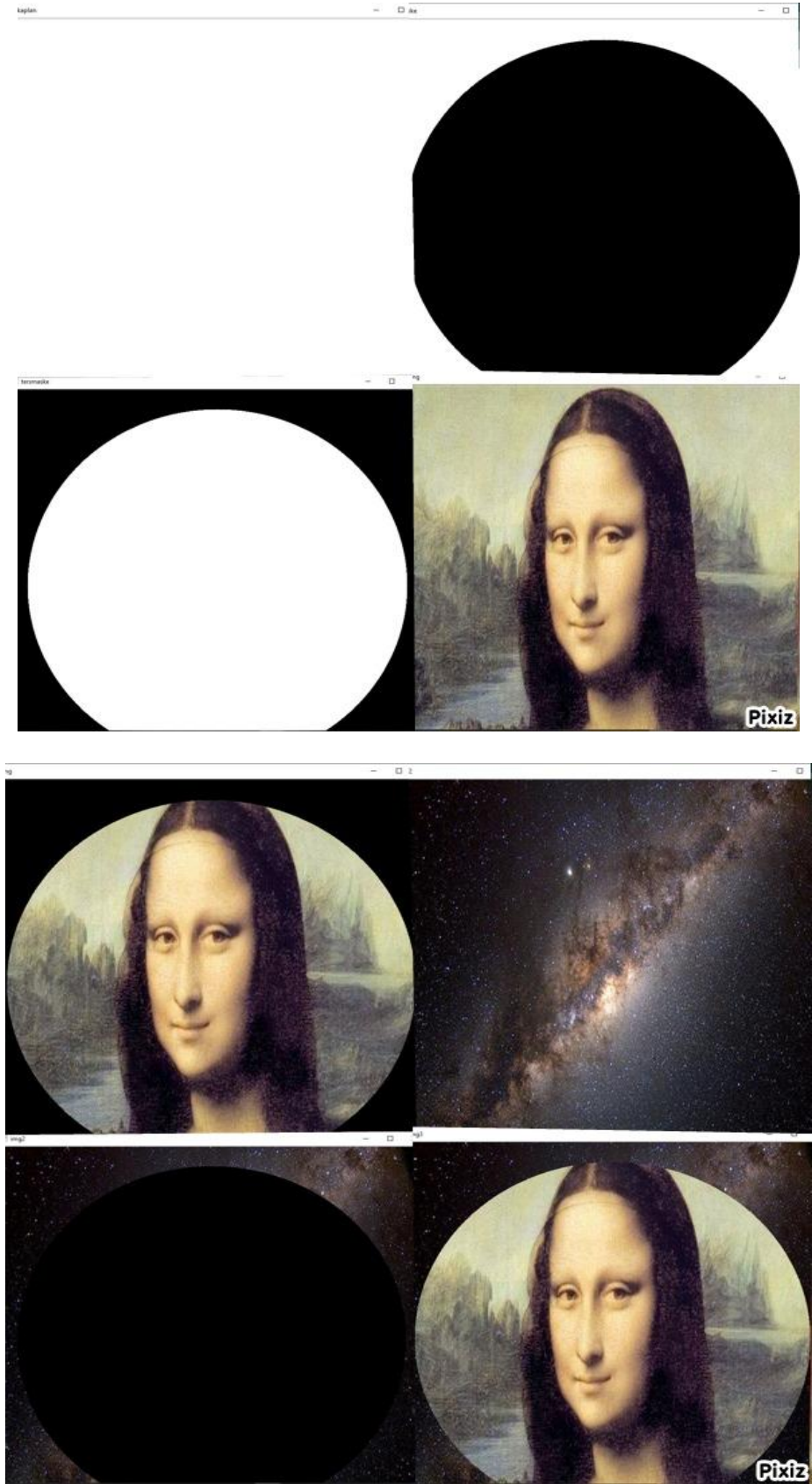
img = cv2.bitwise_and(img,tersmaske)#resim uzerine tersmaskeyi uyguluyoruz
#resmin orta kismina monalisa (maskenin beyaz yerleri) dis kismina siyah alan
olusacak
if bekle:
    cv2.imshow('img',img)
    cv2.waitKey(0)

#diger resmi yukle ve maske uygula
img2 = cv2.imread('../Resimler/uzay(800x800).jpg')
if bekle:
    cv2.imshow('img2',img2)
    cv2.waitKey(0)
img2 = cv2.bitwise_and(img2,maske)#uzay resmimize maskeyi uyguluyoruz
#resmin ortasi siyah kenarlari beyaz oluyor.Beyaz(dis) kisma uzay resmimiz gelmis
oluyor.
if bekle:
    cv2.imshow('img2',img2)
    cv2.waitKey(0)
#iki resim birlestirme islemi
img3 = img + img2#maske ve tersmaske uygulanmis resimleri birlestiriyoruz
cv2.imshow('img3',img3);cv2.waitKey(0)
cv2.destroyAllWindows()

...
İşlem Asamaları:
1-arkaplan olusacak
2-maske olusacak
3-tersmaske olusacak
4-monalisa resmini yukledik
5-monalisaya ters maske uyguladik
6-uzay resmini yukledik
7-uzay resmine maske uyguladik
8-son olarak iki maske uygulanmis görüntüyü birlestirme islemini
gerçekleştirdik
...

```

## Program Çıktısı:



## Görsellerdeki Keskinlikleri Yumuşatma (Blurring)

“Blurring” terimi bir resmin bulanıklaştırılmasını, yumuşatılmasını ifade eder.”Blurring” işlemi bir resme uygulandığında resmin keskin hatları, kenarları belirginliğini kaybeder.Aynı zamanda detaylar baskın renklerin içerisinde eriyerek kaybolur.Görüntülerin gürültülerini(detaylarını) gidermede etkin bir yoldur.

İhtiyaç durumuna göre birçok bulanıklaştırma(yumuşatma) yöntemi vardır.

- cv2.filter2D(imaj,derinlik,kernel)
- cv2.blur(imaj,kernel)
- cv2.GaussianBlur(imaj,kernel,sigma)
- cv2.medianBlur(imaj,kernel\_boyutu)
- cv2.bilateralFilter(imaj,komsuluk\_capi,sigma\_renk,sigma\_uzay)

### cv2.filter2D(imaj,derinlik,kernel)

Bu yöntem, resim matrisindeki değerlerin çekirdek (kernel) boyutlarına göre ortalamalarını bulurak çalışır.Örneğin aşağıdaki örnekte 11x11 boyutlu bir kernel(çekirdek) kullanıyoruz. Bu nedenle her pikselin rengi, içinde bulunduğu 11x11’lik çekirdek matrisin içerisinde yer alan ve birbirine komşu olan 121 pikselin renk ortalamasıyla değiştirilir. Dolayısıyla renkler keskinliğini kaybederek ortalama değerlere doğru kayar.Kernel boyutları büyüdükçe bulanıklık da yoğunlaşır.

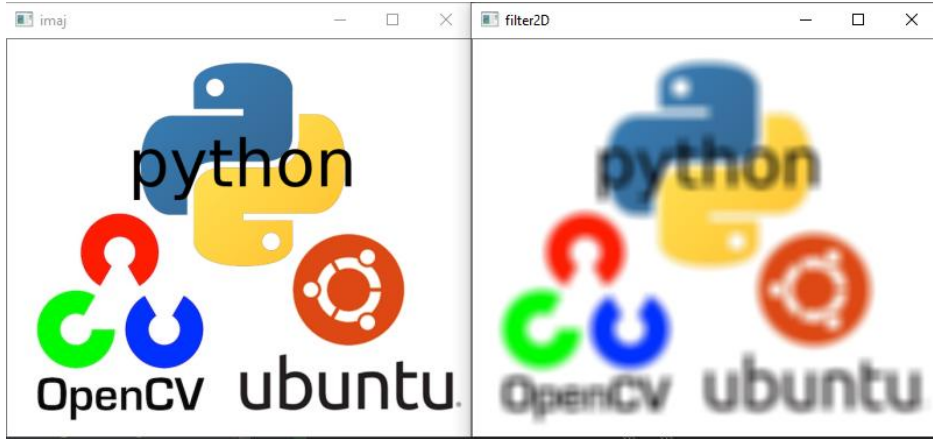
#### 10-blurring\_filter2D.py

```
import cv2
import numpy as np

deltax = 0
deltay = 0

imaj = cv2.imread('../Resimler/pou400.png')
n = 11
kernel = np.ones((n,n),np.float32) / (n*n*1.0) #matris isleme
girdiginde en fazla 0-1 arasında deger almasi saglaniyor
#(kaynak, derinlik, kernel, capa, delta, sinirtipi)
blur = cv2.filter2D(imaj,-1,kernel)
                                #-1 orjinal resmin boyutlarına göre işlem
yapilmasini saglar
cv2.imshow('imaj',imaj)
cv2.imshow('filter2D',blur)
cv2.moveWindow('imaj',10,10)
cv2.moveWindow('filter2D',imaj.shape[1]+deltax,10)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



### cv2.blur(imaj,kernel)

Bu yöntemde her piksel değerinin komşu piksel renklerinin ortalama değeriyle değiştirilmesi esasına dayanır.

11-bluring\_blur.py

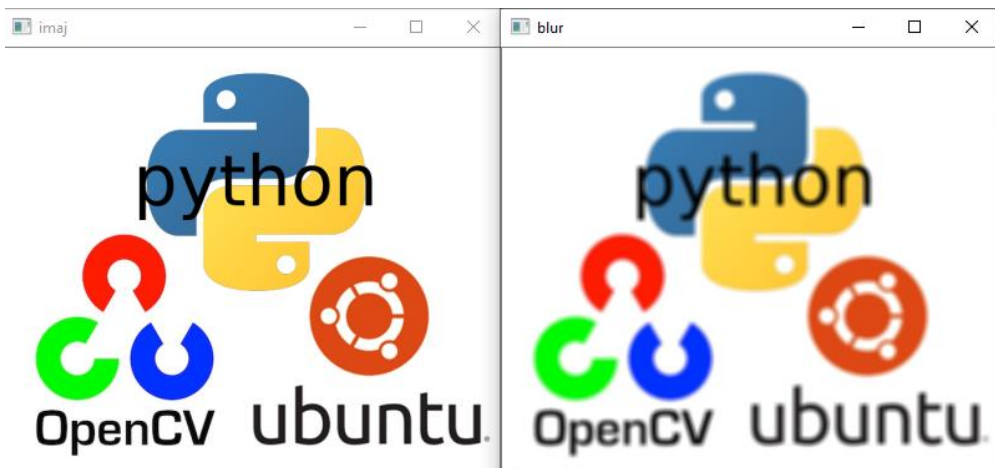
```
import cv2

deltax = 0
deltay = 0

imaj = cv2.imread('../Resimler/pou400.png')
# (imaj, kernel)
blur = cv2.blur(imaj,(5,5))
#kernel (cekirdek) boyutu arttilirdikca detaylarin kaybolmasi durumu artar
cv2.imshow('imaj',imaj)
cv2.imshow('blur',blur)
cv2.moveWindow('imaj',10,10)
cv2.moveWindow('blur',imaj.shape[1]+deltax,10)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



## **cv2.GaussianBlur(imaj,kernel,sigma)**

GaussianBlur filtresi görüntü üzerinde düzleştirme işlemi uygular.

12-blurring\_GaussianBlur.py

```
import cv2

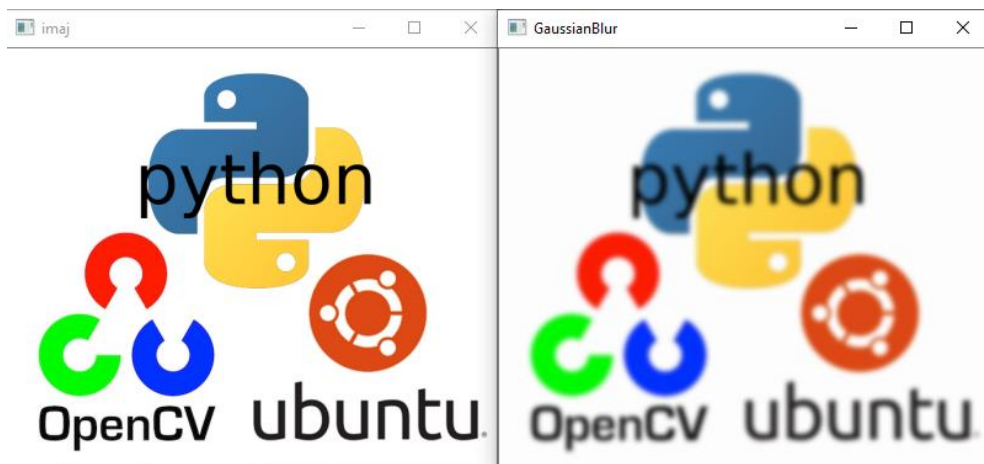
deltax = 0
deltay = 0

imaj = cv2.imread('../Resimler/pou400.png')

#(imaj,kernel,standart sapma)
blur = cv2.GaussianBlur(imaj,(11,11),0) #pozitif tek sayi

cv2.imshow('imaj',imaj)
cv2.imshow('GaussianBlur',blur)
cv2.moveWindow('imaj',10,10)
cv2.moveWindow('GaussianBlur',imaj.shape[1]+deltax,10)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



### cv2.medianBlur(imaj,kernel\_boyutu)

Fonksiyon çekirdek alanının altındaki tüm piksellerin medyanını alır ve merkezi eleman bu medyanın değerleriyle değiştirilir.

13-blurring\_medianBlur.py

```
import cv2

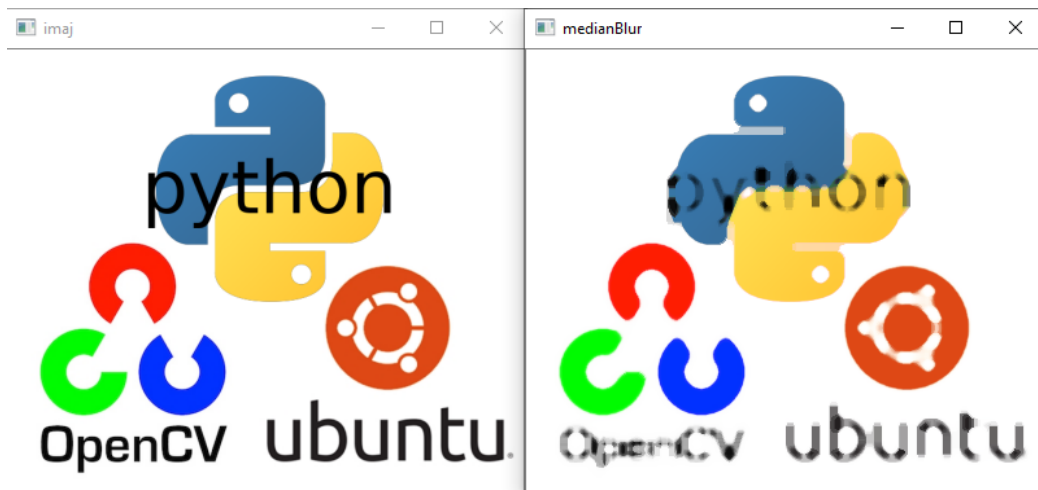
deltax = 0
deltay = 0

imaj = cv2.imread('../Resimler/pou400.png')
# (imaj,kernel_boyutu)
blur = cv2.medianBlur(imaj,11)# tek sayi olmak zorunda

cv2.imshow('imaj',imaj)
cv2.imshow('medianBlur',blur)
cv2.moveWindow('imaj',10,10)
cv2.moveWindow('medianBlur',blur.shape[1]+deltax,10)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



### cv2.bilateralFilter(imaj,komsuluk\_capi,sigma\_renk,sigma\_uzay)

Uyguladığımız filtreler resimler üzerinde bazı bozulmalara sebep olabilir. Örneğin GaussianBlur işleminde resmin piksellerini ortadaki piksellerin ortalama renklerine göre yumuşatma işlemi yaptığı için resmin kenarlarında bozulmalara yol açabilir. Ancak bilateralFilter işlemi bunu önler resim içerisinde kenarları korur.

## 14-blurring\_bilateralFilter.py

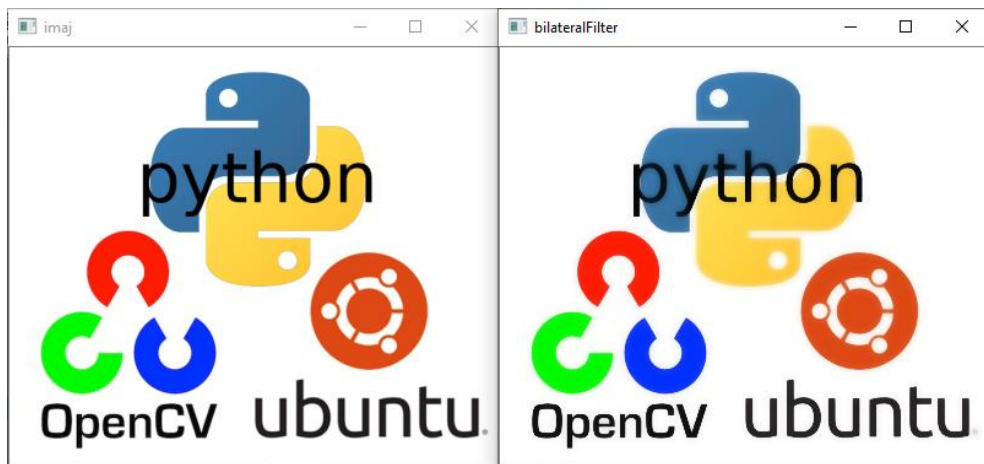
```
import cv2

deltax = 0
deltay = 0
imaj = cv2.imread('../Resimler/pou400.png')
# (imaj,komsuluk_capi,sigma_renk,sigma_uzay)
blur = cv2.bilateralFilter(imaj,11,175,175)

cv2.imshow('imaj',imaj)
cv2.imshow('bilateralFilter',blur)
cv2.moveWindow('imaj',10,10)
cv2.moveWindow('bilateralFilter',blur.shape[1]+deltax,10)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



## Görüntüleri Daraltmak ve Genişletmek

Siyah beyaz binary bir resmin ön plan(beyaz) görüntülerini daraltmak veya genişletmek için kullanabileceğimiz 2 tane fonksiyonumuz var: `cv2.erode()` ve `cv2.dilate()`

### **cv2.erode():**

**cv2.erode(imaj, kernel, iterasyon)** parametrelerini alır.Burada imaj siyah beyaz görüntümüzdür.Kernel çekirdeğimizi ifade eder.İterasyon inceltme işleminin kaç defa tekrar edeceğini ifade eder.Ön plan beyaz görüntümüzün sınırları verilen çekirdek boyutuna aşındırılır.Görüntünün siyah beyaz olması önemlidir.Çekirdek alandaki siyah pikseller beyaza, beyaz pikseller ise siyah piksele dönüştürülme işlemi yapılır.Bu fonksiyon beyaz görüntünün gürültüsünün giderilme işlemini yapar.



## cv2.dilate():

**cv2.dilate(imaj, kernel, iterasyon).**Parameterini alır.İmaj siyah beyaz resmi ifade eder.Kernel çekirdek boyutumuzdur.İterasyon ise kalınlaştırma işleminin kaç defa gerçekleştirileceğini ifade eder.Fonksiyon çekirdek içerisinde bir tane bile beyaz piksel varsa geriye beyaz, aksi halde siyah döndürür.Dolayısıyla beyaz görüntünün sınırları genişletilmiş olur.

Erode ve dilate işlemleri ard arda uygulanırsa sınır çizgileri daha düzgün bir hale gelir.

15-erode\_dilate.py

```
import cv2
import numpy as np

imaj = cv2.imread('../Resimler/1.png',0)
kernel = np.ones((5,5),np.uint8)

#sinirlari inceltme
erosion = cv2.erode(imaj, kernel, iterations = 1)
#sinirlari kalinlastirma
dilation = cv2.dilate(imaj, kernel, iterations = 1)
...
    1-imaj : siyah beyaz resim
    2-kernel : cekirdek
    3-iterations : inceltme ve kalinlastirma isleminin tekrarlanma sayisi
...

#uc goruntuyu dusey olarak birlestirme
dusey = np.vstack((imaj, erosion, dilation))
#vstack 3 resmi alt alta birlestirme yapar
cv2.imshow('dusey',dusey)
cv2.moveWindow('dusey', 600, 50)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

**Program Çıktısı :**





## Sınır Çizgilerinin (Konturların) Belirlenmesi ve İşlenmesi

Sınır çizgileri sınır çizgilerindeki aynı renk veya yoğunluğa sahip noktaları birleştiren eğrilerdir. Konturlar biçim analizi, nesne saptama ve tanıma işlemlerinde kullanılır.

Kontur belirlemede siyah beyaz görüntüleri kullanmak daha iyi sonuçlar vermektedir. Bu yüzden threshold veya canny uygulamalarından sonra konturların bulunması daha kolay olur.

OpenCV açısından bulunacak nesnenin beyaz arkaplanın siyah renkte olması gerekmektedir.

16-esikler.py

```
import cv2

deltax = 0
deltay = 0

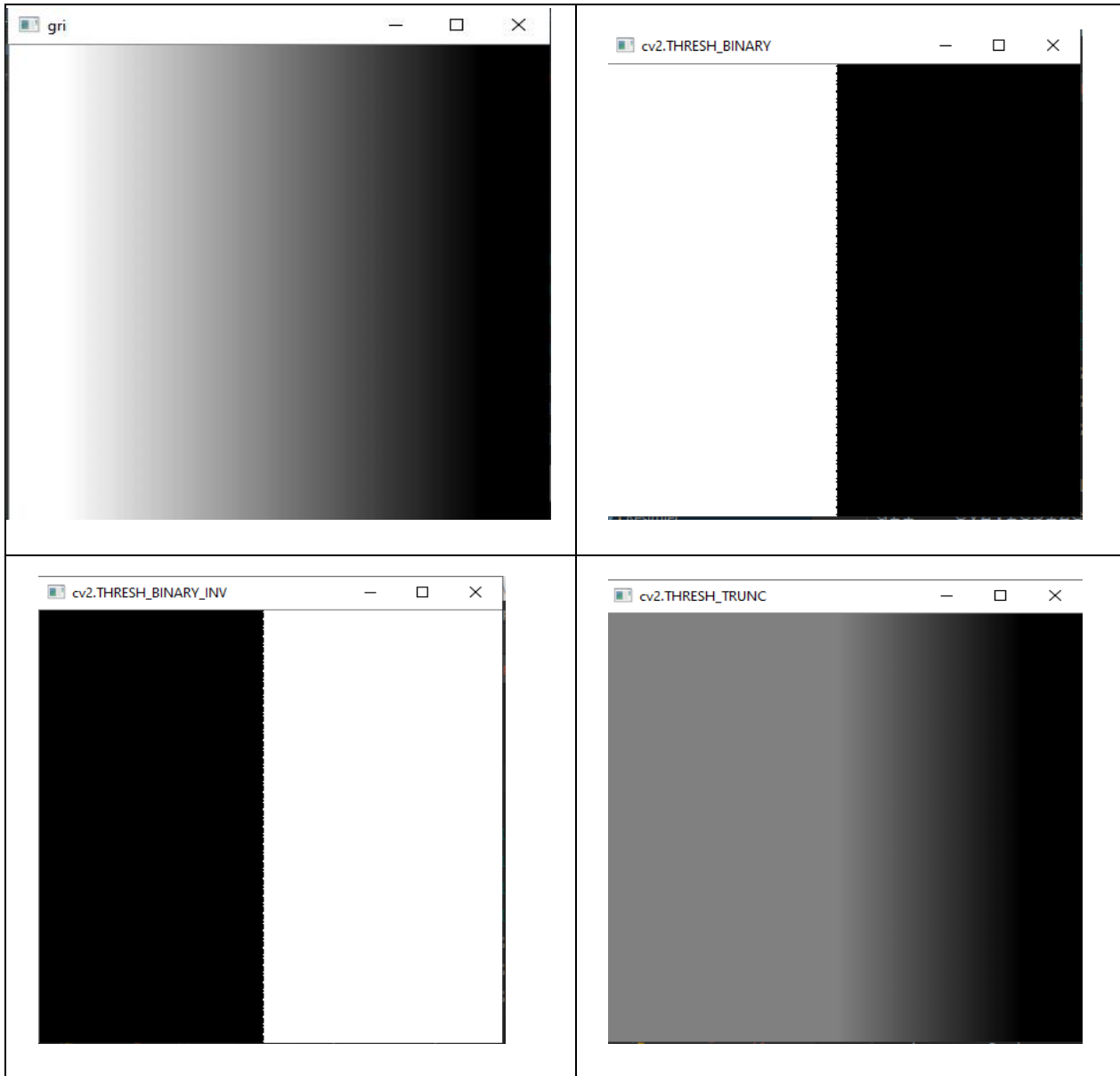
sthres = ['cv2.THRESH_BINARY', 'cv2.THRESH_BINARY_INV',
          'cv2.THRESH_TRUNC', 'cv2.THRESH_TOZERO',
          'cv2.THRESH_TOZERO_INV', 'cv2.THRESH_MASK',
          'cv2.THRESH_OTSU', 'cv2.THRESH_TRIANGLE']
thrs = [cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC,
        cv2.THRESH_TOZERO, cv2.THRESH_TOZERO_INV, cv2.THRESH_MASK,
        cv2.THRESH_OTSU, cv2.THRESH_TRIANGLE]
print(thrs)
gri = cv2.imread('../Resimler/gradient.jpg', cv2.IMREAD_GRAYSCALE)
gri = cv2.resize(gri, (int(gri.shape[1]/gri.shape[0]*400), 400))

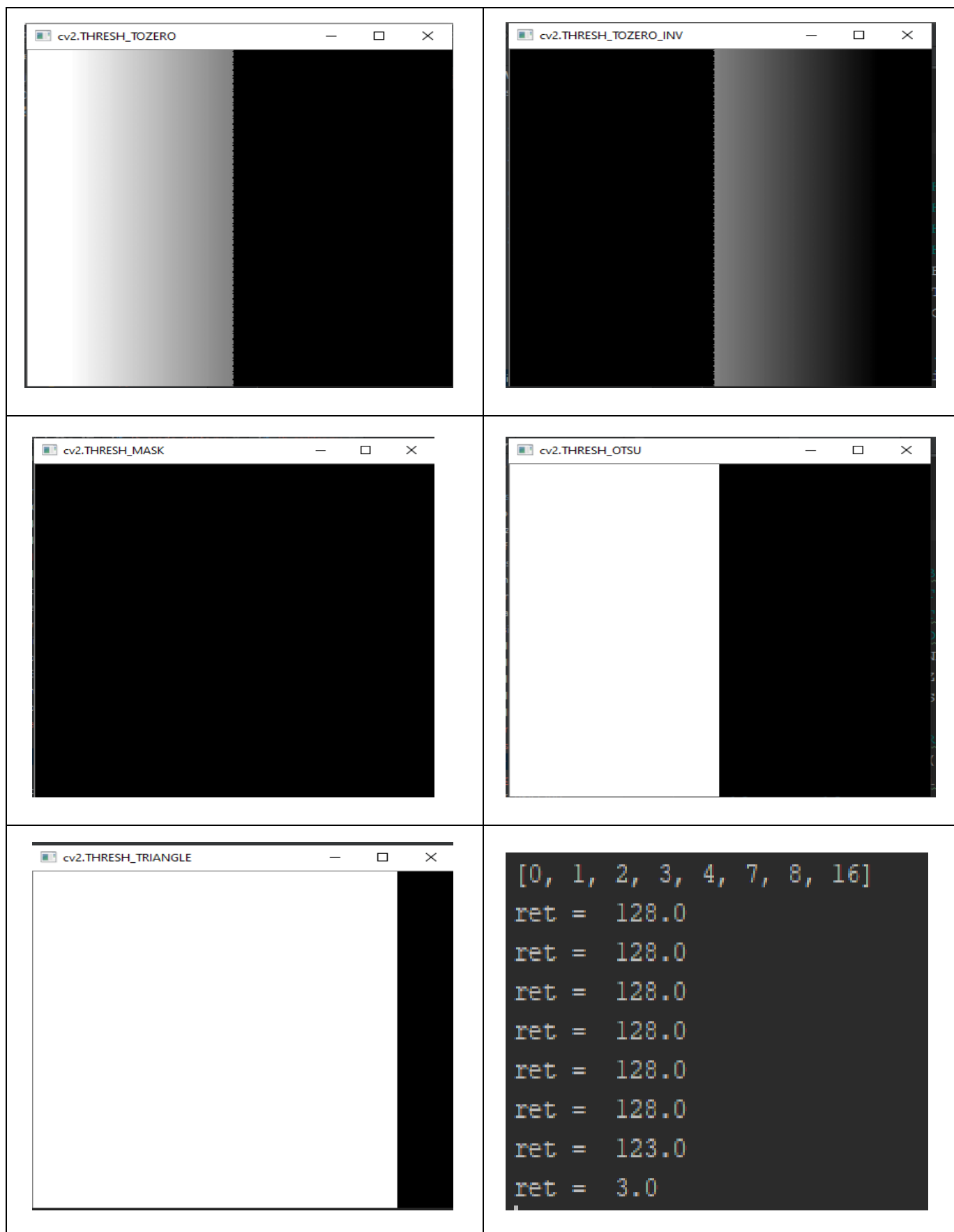
#
x = 10; y = 10
cv2.imshow('gri', gri)
cv2.moveWindow('gri', x, y)
x = 45;
for j in range(len(thrs)):
    #burada yapılan işlem acılan pencereler ekran boyutuna ulastiginda alt satira
    #gecmesini sağlamak
    x += gri.shape[1] + deltax
    if x > 1600: #1600 ekran genisligi
        x = 45; y += gri.shape[0] + deltay
    ret, esik = cv2.threshold(gri, 128, 255, thrs[j])
    #(gri_imaj, esik_degeri, max_degeri, esik_tipi)
    #esik_degeri : 0 - 255
    #max_deger : 0 - 255
    #geriye iki deger dondurur birincisi esik degeri ikincisi yeni olusan esik
    #goruntusudur.
    #ret : cv2.THRESH_OTSU kullanilirsas hesaplanan esik degeri, yoksa verilen esik
    #degeri
```

```
'''
    Parameters
        src      : giriş dizisi (çok kanallı, 8 bit veya 32 bit kayma noktası).
        thresh   : esik değeri.
        maxval   : THRESH_BINARY ve THRESH_BINARY_INV ile kullanılacak esikleme
değeri.
        type    : esik turu.
    '''
    print('ret = ',ret)
    cv2.imshow(sthrs[j],esik)
    cv2.moveWindow(sthrs[j],x,y)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:





## Sınır Çizgileri

### **cv2.findContours():**

Konturlar şekil analizi, nesne algılama ve tanıma için kullanışlı bir araçtır. cv2.findContours(orijinal imaj, mod, metod) parametrelerini alır.

Mod: Kontur bulma yöntemini ifade eder.

Metod: Kontur yaklaşım algoritmasını ifade eder

### **cv2.drawContours():**

Kontür hatları veya dolgulu konturlar çizer.

Cv2.drawContours(image ,contours,contourIdx, color, thickness)

Parametreleri:

**image** – Hedef görüntü.

**contours** – Tüm giriş konturları. Her kontur bir nokta vektörü olarak saklanır.

**contourIdx** – Çizilecek konturu gösteren parametre. Negatifse, tüm konturlar çizilir.

**color** - kontur rengi

**thickness** – konturların çizgi kalınlığı

17-kontur01.py

```
import cv2
import numpy as np

imaj = cv2.imread('../Resimler/python.png')

# resmin boyutunu buyutme
y = imaj.shape[0]*2
x = imaj.shape[1]*2

imaj = cv2.resize(imaj,(x,y))

gri = cv2.cvtColor(imaj,cv2.COLOR_BGR2GRAY)

...
cv2.cvtColor()
    Parameters:
        src: rengi degistirilecek goruntu
        code: renk alani donusturme kodu
...
```

```

_,sb = cv2.threshold(gri,127,255,cv2.THRESH_BINARY)
#_ = esik degeri , sb = yeni olusan esik görüntüsüdür
konturlar = cv2.findContours(sb,cv2.RETR_EXTERNAL,
                             cv2.CHAIN_APPROX_SIMPLE)[-2]

# cv2.findContours(orjinal imaj = imaj, mod = cv2.RETR_EXTERNAL, metod =
cv2.CHAIN_APPROX_SIMPLE)
#mod : kontur bulma yontemi (RETR_LIST,RETR_EXTERNAL,RETR_CCOMP,RETR_TREE)
#metod : kontur yaklasim yontemi (approximation)

imaj2 = cv2.drawContours(imaj.copy(),konturlar,-1,(0,0,255),2)

...
cv2.drawContours()
    Parameters:
        image - Hedef goruntu.
        contours - Tüm giriş konturları. Her kontur bir nokta vektörü olarak
saklanır.
        contourIdx - Çizilecek konturu gösteren parametre. Negatifse, tüm konturlar
çizilir.
        color - kontur rengi
        thickness - konturlarin cizgi kalinligi

...

#iki resmi dusey olarak birlestirme
imaj3 = np.vstack((imaj,imaj2))
cv2.imshow('imaj3',imaj3)
cv2.moveWindow('imaj3',10,10)

cv2.waitKey(0)
cv2.destroyAllWindows()

```

Program Çıktısı:



## cv2.Canny()

Canny algoritmasını kullanarak görüntüdeki kenarları bulur. Fonksiyon, giriş görüntüsünde kenarları bulur ve Canny algoritmasını kullanarak bunları çıktı haritası kenarlarında işaretler. Eşik1 ve eşik2 arasındaki en küçük değer kenar bağlantısı için kullanılır. En büyük değer, güçlü kenarların başlangıç segmentlerini bulmak için kullanılır

18-kenar01.py

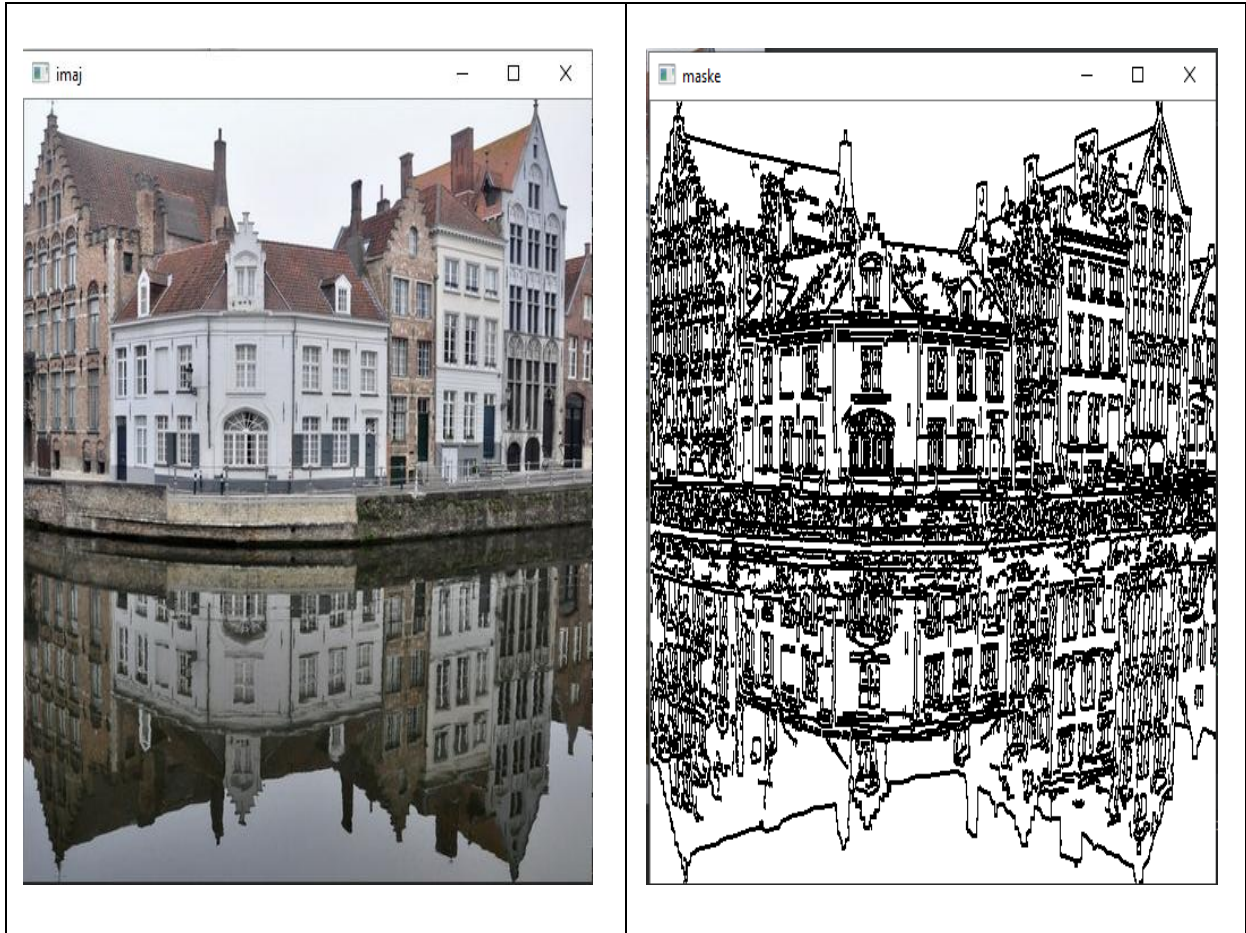
```
import cv2
#Nesne kenarlarının saptanmasi
deltax = 0
deltay = 0
imaj1 = cv2.imread('../Resimler/bina.jpg')
imaj = cv2.resize(imaj1,(500,500))
kenarlar = cv2.Canny(imaj, 50, 150)
#geri dondurulen kenar haritasi
'''
cv2.Canny()
Parameters:
    image - tek kanalli 8 bit giris goruntusu.
    threshold1 - kesiklik islemi icin ilk esik
    threshold2- kesiklik islemi icin ikinci esik
'''
maske = cv2.bitwise_not(kenarlar)
# bitwise_not() Bir dizinin her bitini tersine çevirir.
# parametresi giris arrayi
maske = cv2.erode(maske,(5,5),iterations = 2)
# erode kenarlari inceltme islemi

#maske = kenarlar

imaj2 = cv2.bitwise_and(imaj,imaj,mask = maske)
#bitwise_and iki resmin (maske ve imaj) bitsel birlesimini saglar
cv2.imshow('imaj',imaj)
cv2.imshow('maske',maske)
cv2.imshow('imaj2',imaj2)
cv2.moveWindow('imaj',10,10)
cv2.moveWindow('maske',imaj.shape[1] + deltax,10)
cv2.moveWindow('imaj2',imaj.shape[1] + maske.shape[1] + deltax, 10)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Program Çıktısı:



## Kamera Görüntülerini Okumak

OpenCV’de videodan görüntü yakamak için videoCapture fonksiyonu kullanılır. Bunun için VideoCapture nesnesi oluşturmamız gerekir. Parametresi, aygıt dizini veya bir video dosyasının adı olabilir. Cihaz dizini yalnızca hangi kamerayı belirleyeceğiniz sayıdır. Bu sayılar dahili kamera veya takılı olan tek bir kameramız var ise 0’dır. Dışarıdan ikinci bir kamera takıyorsak bunun değeri 1’dir.

19-kamera\_oku.py

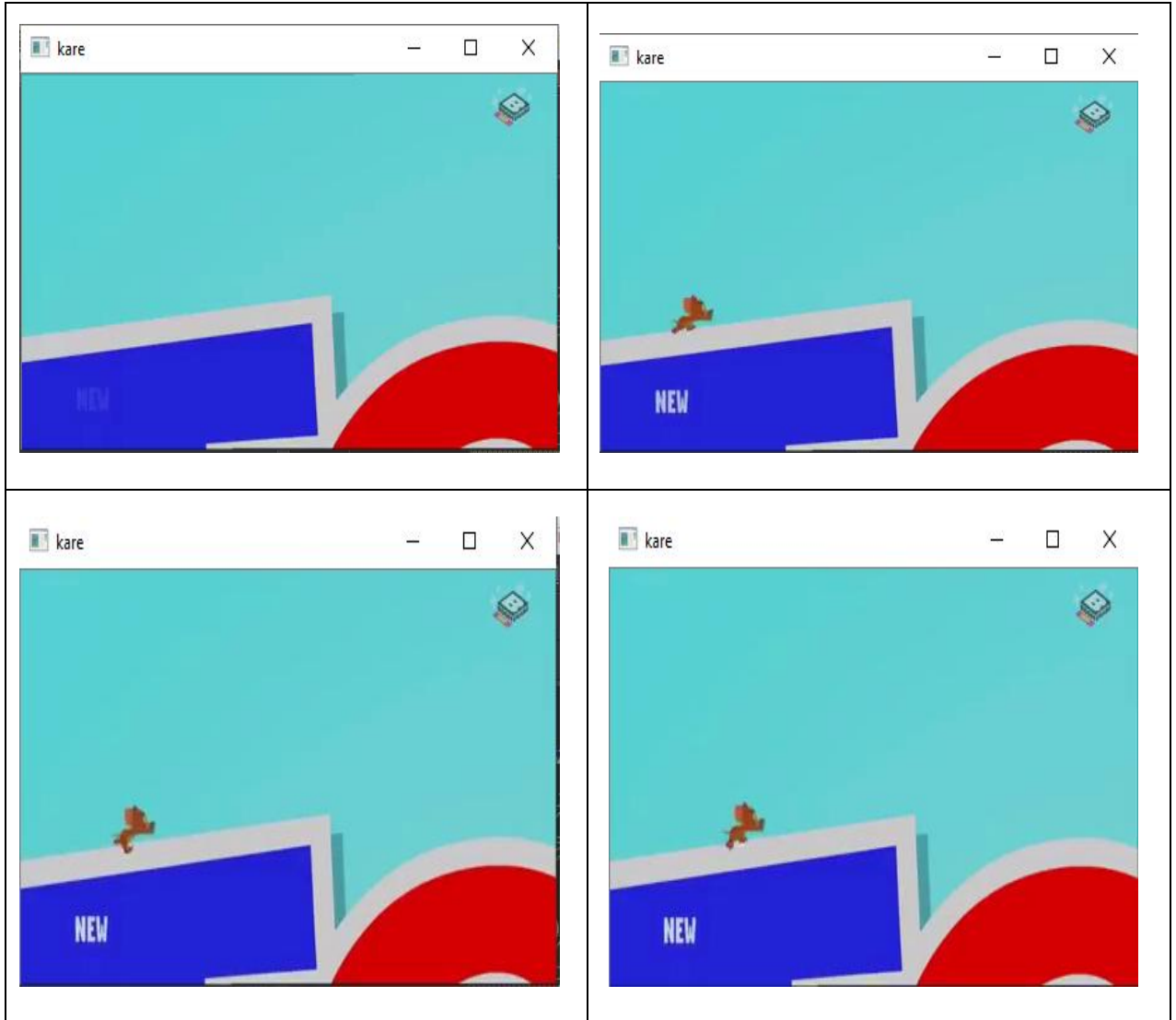
```
import cv2
def ana():
    # kamera = cv2.VideoCapture(0)
    kamera = cv2.VideoCapture("../Videolar/tomandjerry.mp4")
    #VideoCapture fonksiyonuna istersek video dosyamızı istersekte kamera index
    imizi verebiliriz.
    # 0. kamera index i varsayılan kameramızdır.

    while(True):
        #videodan goruntu alma islemi
        ret, kare = kamera.read()

        #goruntu alindimi kontrol et alinmadiysa durdur
        if not ret: break
        cv2.imshow('kare',kare) #goruntuyu gosterme islemi
        cv2.moveWindow('kare',10,10)
        cv2.waitKey(0) #degerini sifir yaparsak kare kare yakalama islemini
        gerceklestirebiliriz.
        #cv2.waitKey(25) video oynamaya devam eder
        #kameranın acilip acilmadigin kontrol eder
        if kamera.isOpened():
            #VideoCapture Sinifini kapatir
            kamera.release()
            cv2.destroyAllWindows()
if __name__ == '__main__':
    ana()
```



Program Çıktısı:



## Kamera Çözünürlükleri

OpenCV aracılığıyla kameramızın çözünürlükleri üzerinde değişiklik yapabiliriz. Biz istediğimiz çözünürlükleri verebilirken kameramız sadece desteklediği çözünürlükleri uygulayacaktır. Bunu kontrol etmek için.

```
import cv2

#cozunurlukler
cozwh = [(1920,1080),(1600,900),(1366,768),(1280,720),
         (1024,576),(960,544),(640,480),(320,240)]

def ana():
    kamera = cv2.VideoCapture(0)
    for j in range(len(cozwh)): #cozunurlukler
        w0 = int(cozwh[j][0])#genislik
        h0 = int(cozwh[j][1])#yukseklık degerleri sirasiyla hep denenecek
```

## 20-kamera\_cozunurlukleri.py

```
kamera.set(3,w0) #genislik w0 genislik
kamera.set(4,h0) #yukseklık h0 yukseklik

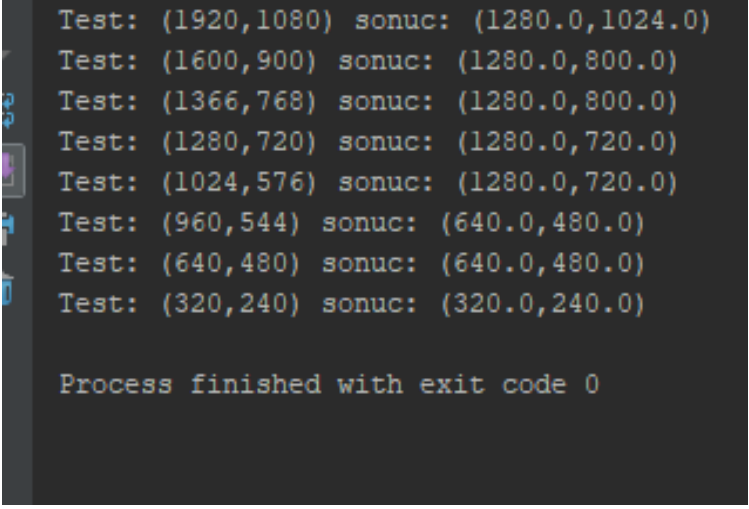
#kameranın olusan cozunurluk degerleri (VideoCapture ozellikleri )
#desteklenen boyutlara gore degisim saglar
w1 = kamera.get(3)
h1 = kamera.get(4)

#print(f"Test: ({w0},{h0}) sonuc: ({w1},{h1})")
print("Test: ({},{}) sonuc: ({},{})".format(w0,h0,w1,h1))

if kamera.isOpened():
    kamera.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    ana()
```

Program Çıktısı :



```
Test: (1920,1080) sonuc: (1280.0,1024.0)
Test: (1600,900) sonuc: (1280.0,800.0)
Test: (1366,768) sonuc: (1280.0,800.0)
Test: (1280,720) sonuc: (1280.0,720.0)
Test: (1024,576) sonuc: (1280.0,720.0)
Test: (960,544) sonuc: (640.0,480.0)
Test: (640,480) sonuc: (640.0,480.0)
Test: (320,240) sonuc: (320.0,240.0)

Process finished with exit code 0
```

Test kısmında biz istediğimiz çözünürlüğü yolladık sonuç kısmında kameramız desteklediği en yakın değere çevirdi.

## Kamera Kayıt İşlemleri

OpenCv’de bilgisayarımızdaki kameramızı okuyabildiğimiz gibi okuduğumuz kameramızı da kaydedebiliyoruz.

OpenCv bunu yapmak için Codec yani kısaca kod çözücü işlemini kullanmaktadır. Dijital ortamda video dosyalarını, görüntü dosyalarını, ses dosyalarını sıkıştırma ve yeniden açma işlemlerinde kullanılmaktadır.

FourCC (Four character code) medya dosyalarında kullanılan codec’ler için pixel formatlarını, renk formatlarını, sıkıştırma formatlarını standart bir biçimde tanımlarlar. Bu tanımlamayı sadece 4 karakter kullanarak yaptığı için 4CC ya da FourCC adını almıştır. Tanımlamaları sadece ASCII tablosu üzerindeki karakterleri kullanarak yapmaktadır.

OpenCv video dosyalarını .avi, .jpeg, .mp4 uzantılı olarak kaydedebilir. Biz kameramızı okuma işlemi yapıcaz daha sonra okuduğumuz kameramızı .mp4 formatında kayıt işlemini gerçekleştireceğiz.

21-kamera\_kayit.py

```
import cv2

kamera = cv2.VideoCapture(0)
# fourcc = cv2.VideoWriter_fourcc(*"XVID") # .avi
# fourcc = cv2.VideoWriter_fourcc(*"MJPG") # .jpeg
fourcc = cv2.VideoWriter_fourcc(*'m','p','4','v') # .mp4 uzantili. videonun
kaydedilme algoritmasi fourcc

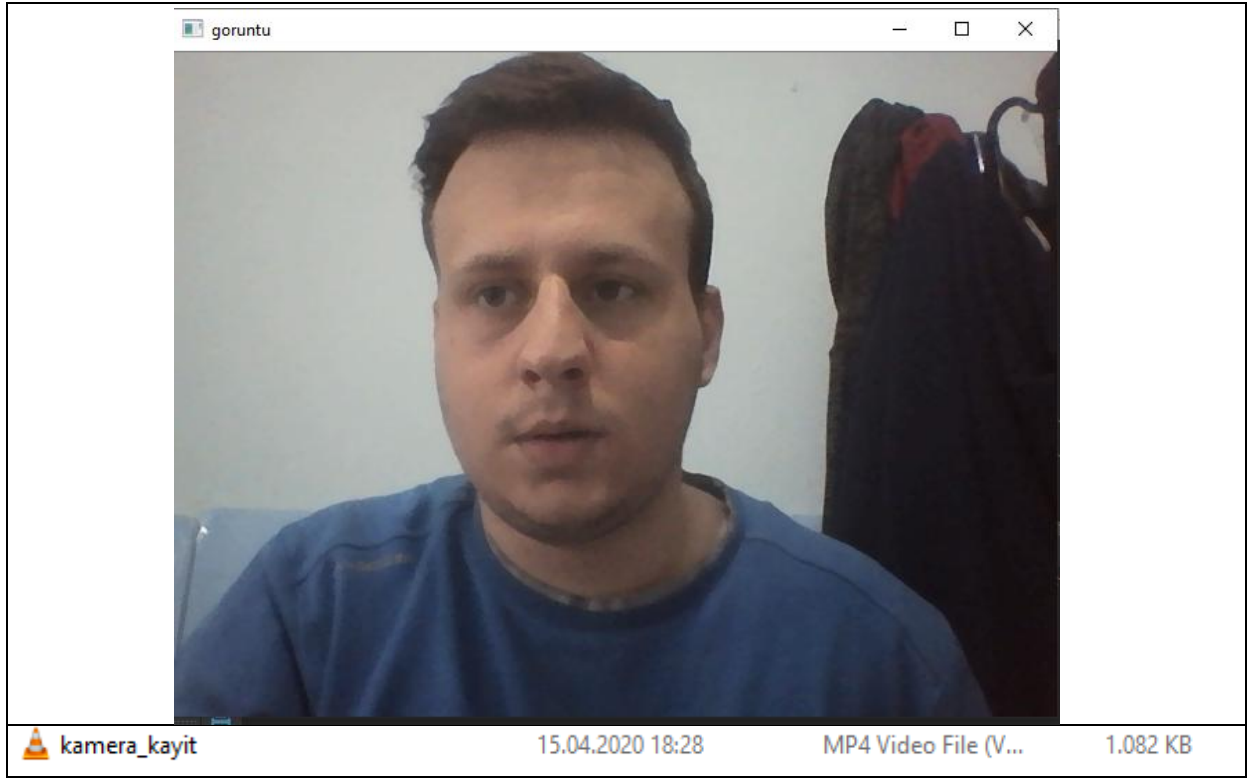
kayit = cv2.VideoWriter('kamera_kayit.mp4',fourcc,24.0,(640,480))
#cv2.VideoWriter(video adi,kaydedilme algoritmasi, saniyede alinan kare
sayisi,cozunurluk boyutlari)

#kayit ne zaman baslayacak kamera acildi ise baslayacak
while kamera.isOpened():
    ret,video = kamera.read()
    if ret == True: # video varsa true yoksa false
        #video = cv2.flip(video,0)#kameray dondurme islemi ters video 0 ters 1 duz
        kayit.write(video)
        cv2.imshow('goruntu',video)

        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

kamera.release()
kayit.release()
cv2.destroyAllWindows()
```

## Program Çıktısı:



# BÖLÜM-2

# UYGULAMALAR

## Yüz Algılama

Çeşitli yüz ve beden parçalarının tespitinde yaygın olarak CascadeClassifier() filtreleri kullanılır. Bunlar eğitilmiş sınıflandırıcılardır .xml uzantılıdır. Bu dosyalara <https://github.com/opencv/opencv/tree/master/data/haarcascades> adresinden ulaşılabilir.

## OpenCv Yüzleri Nasıl Tanır?

Bilgisayar yüz tanıma sistemlerine benzer sistemlerde geometrik açıdan tespiti gerçekleştirebilmektedir. İnsan yüzü tanımak bunun gibi bir şeydir. Yüz burnun tam ortasından ikiye bölündüğünde oluşan parçalar neredeyse birbirinin simetriğidir. Burun iki gözün altında yer alır, iki adet göz bulunur. Burun ve ağız iki gözün ortasındadır gibi temel özelliklerden ve bilinenden yararlanır.

OpenC V içerisinde kullandığımız haarcascade algoritması buna bir örnektir. Bu algoritmalar ile görüntü içerisindeki yüzler, yüzün içerisindeki göz ve ağız tespit edilebilir.

Yüz tespit sistemlerinden farklı olarak yüz tanıma sistemlerinde ise çok fazla veriye ihtiyaç duyulur. Tanınması istenen yüzün verileri eğitilmek üzere sinir ağlarına eğitim verisi olarak verilir. Sinir ağı ise yüzdeki baskın özelliklere göre bir sınıflandırma yapar. Girdi olarak verilen yüzün kendine has özellikleri sinir ağı tarafından sınıflandırılır. Bu şekilde bu veri tekrar önüne geldiğinde kolay bir şekilde yüzü tanıyabilir. Her yüz için ayrı sınıflandırma oluşturulur. Burada önemli olan sisteme girilen yüzün hangi sınıfa ait olduğudur.

## Video Kamera Üzerinden Yüz Tespiti

OpenCv' nin açık kaynak olarak kullandığı haarcascade sınıflandırıcısını kullanarak bu işlemleri gerçekleştireceğiz.

```

import cv2

#siniflandirici yuklenmesi
yuzCascade = cv2.CascadeClassifier(
    '../Cascades/haarcascade_frontalface_default.xml')
kamera = cv2.VideoCapture(0)

while True:
    _, kare = kamera.read() #kameradan okuma islemi
    #ilk deger _ frame in dogru okunup okunmadigini kontrol eder (true-false)
    #ikinci deger kare yakalanan kareyi ifade eden ndarraydir
    kare = cv2.flip(kare,1)#kamera ters ise -1 , aynalama icin 1
    gri = cv2.cvtColor(kare, cv2.COLOR_BGR2GRAY)#griye cevirme islemi
    #yuz saptama isleminde daha iyi bir sonuc icin gri tonlamada yapiyoruz

    # detectMultiScale goruntu icerisinde birden fazla yuz varsa onlari yakalar
    yuzler = yuzCascade.detectMultiScale(
        gri,
        scaleFactor = 1.2,#Her görüntü ölçeğinde görüntü boyutunun ne kadar
        azaltılacağını belirten parametre.
        minNeighbors = 5,#Her aday dikdörtgenin kaç tane komşu tutması gerektiğini
        belirten parametre.
        minSize = (20, 20)# mumkun olan en kucuk nesne boyutu bundan kucukleri
        gozardi edilir
    )
    #detectMultiScale Giriş görüntüsünde farklı boyutlardaki nesneleri algılar.
    # Algılanan nesneler dikdörtgenler listesi olarak döndürülür.

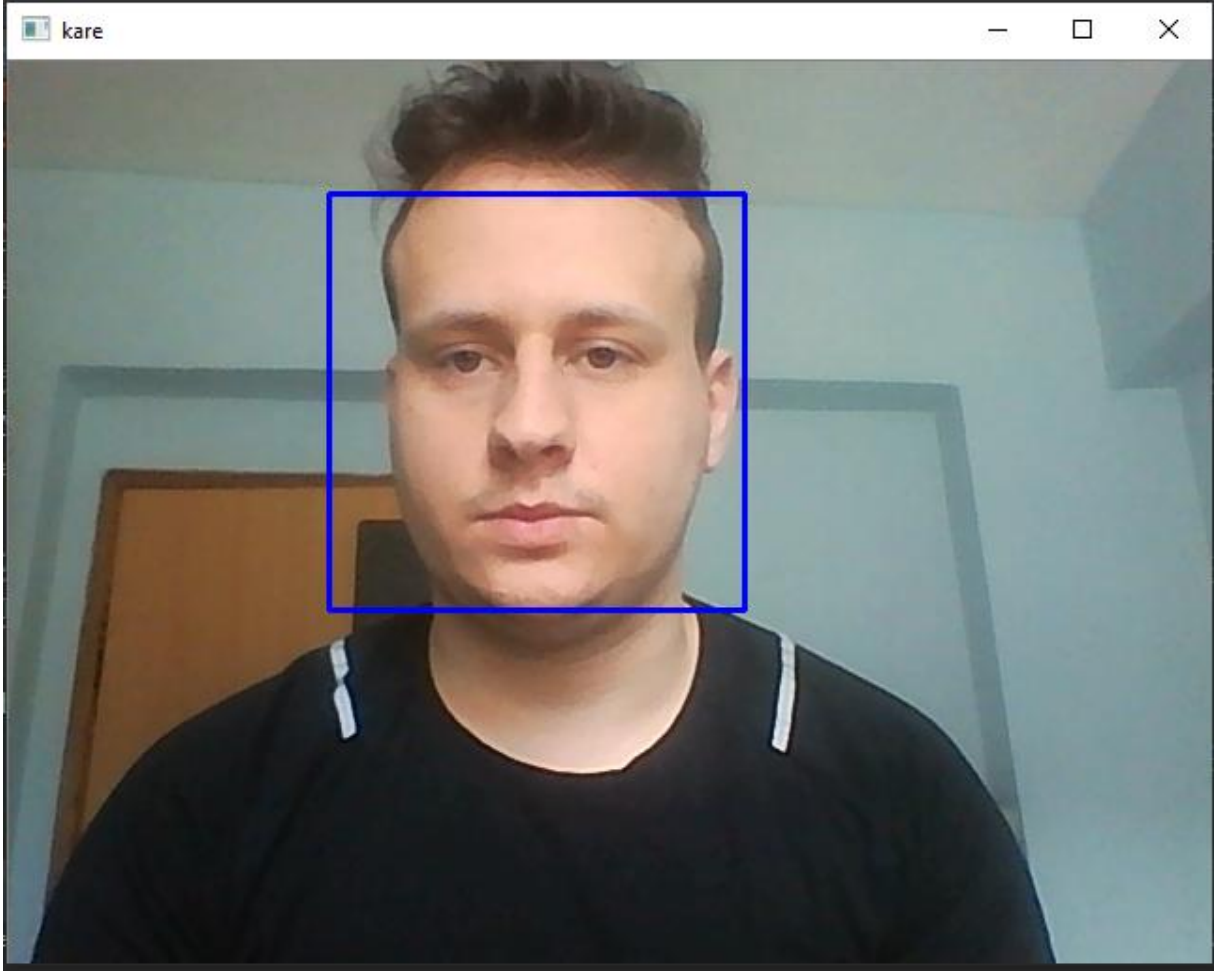
    #(x,y) => sol ust kose koordinalari (w,h) => genislik ve yukseklik
    for (x,y,w,h) in yuzler:
        # rectangle dikdortgen cizme islemi yapar
        cv2.rectangle(kare,(x,y),(x+w,y+h),(255,0,0),2)
        #cv2.rectangle(dikdortgeni cizecegi alan,yuzun basladigi sol ust kosesi,
        yukseklik ve genisligi,
        #dikdortgen rengi, cizgi kalinligi)

        cv2.imshow('kare',kare) # goruntuleme islemi
        k = cv2.waitKey(1) & 0xff # q tusuna basildiginda pencerenen kapanmasi icin
        if k == 27 or k==ord('q'):break

    kamera.release() #is bittiginde kamerayi serbest birak
    cv2.destroyAllWindows()

```

Program Çıktısı:



## Resim Üzerinden Yüz Tespit İşlemi

Yaptığımız işlem aynı farklı olarak video kamerayı okumak yerine programa bir tane resim veriyoruz ve resim içerisindeki yüzleri verdiğimiz parametreler doğrultusunda sınıflandırmasını istiyoruz.



resimden\_yuz\_tanima.py

```
import cv2

foto = cv2.imread('../Resimler/millitakim.jpg')
gri = cv2.cvtColor(foto, cv2.COLOR_BGR2GRAY)

#siniflandirici yuklenmesi
yuz_belirleme = cv2.CascadeClassifier(
    '../Cascades/haarcascade_frontalface_default.xml')

yuzler = yuz_belirleme.detectMultiScale(gri,1.5,5)

for (x,y,w,h) in yuzler:
    cv2.rectangle(foto,(x,y),(x+w,y+h),(255,0,0),2)

cv2.imshow('foto',foto)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program Çıktısı:



## Göz Saptama İşlemi

Göz tespiti yapmak için yine haarcascade sınırlandırıcılarını kullanıyoruz. Yaptığımız işlem yüz tespiti ile aynı aslında, farklı olarak göz tespiti yaparken gözleri yüzün içerisinde aramasını söylüyoruz.



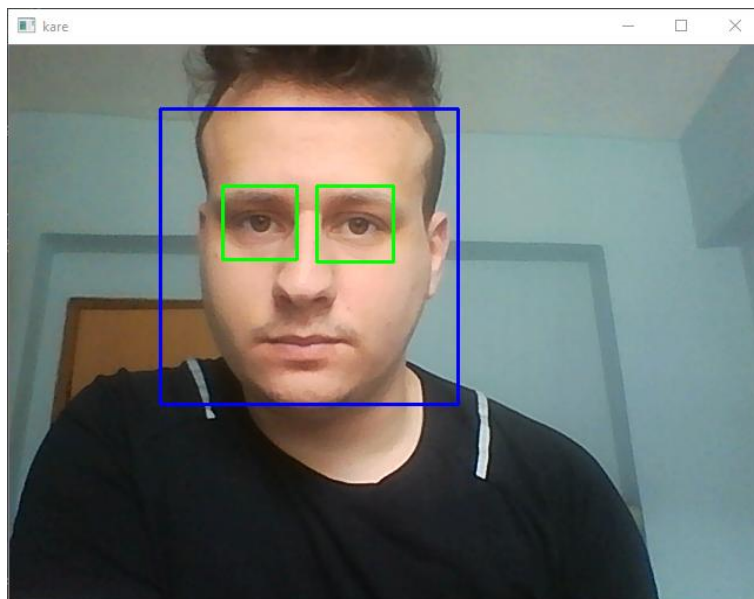
goz\_saptama.py

```
import cv2
yuzCasCade = cv2.CascadeClassifier(
    '../Cascades/haarcascade_frontalface_default.xml'
)
gozCascade = cv2.CascadeClassifier(
    '../Cascades/haarcascade_eye.xml'
)
kamera = cv2.VideoCapture(0)
while True:
    _, kare = kamera.read()
    kare = cv2.flip(kare,1)
    gri = cv2.cvtColor(kare,cv2.COLOR_BGR2GRAY)
    yuzler = yuzCasCade.detectMultiScale(
        gri,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(20,20)
    )
    for (x,y,w,h) in yuzler:
        cv2.rectangle(kare, (x, y), (x + w, y + h), (255, 0, 0), 2)
        gri_kutu = gri[y:y+h,x:x+w] # y'den (y+h)'ye kadar ve x'den (x+w)'ye kadar
        renkli_kutu = kare[y:y+h, x:x+w]

        gozler = gozCascade.detectMultiScale(gri_kutu)
        for (ex,ey,ew,eh) in gozler:
            cv2.rectangle(renkli_kutu, (ex,ey),(ex+ew,ey+eh),(0,255,0),2)
    cv2.imshow('kare',kare)

    k = cv2.waitKey(10) & 0xff
    if k == 27 or k==ord('q'):break
kamera.release()
cv2.destroyAllWindows()
```

Program Çıktısı:



## Gülümseme Tespiti

Haarcascade'in gülümseme tespiti için oluşturulmuş bir sınıflandırıcısı var.Bunu kullanarak yine yüzün içerisinde gülümseme tespiti yapacağız.

gulus\_saptama.py

```
import cv2

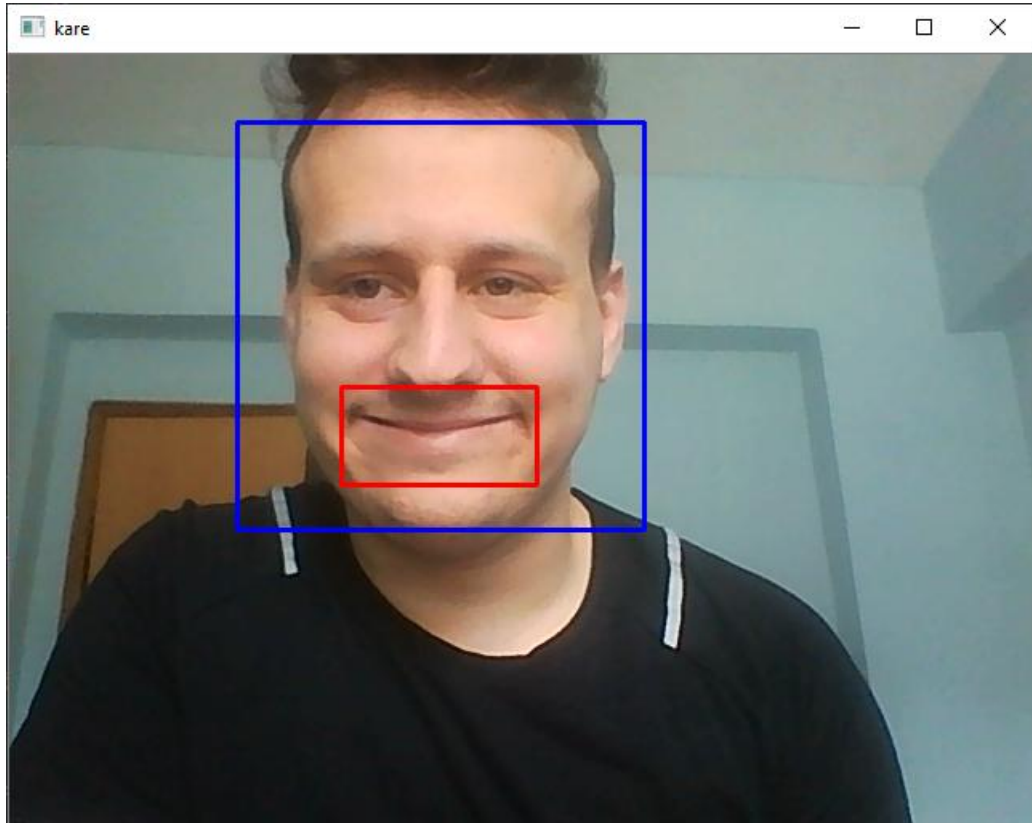
yuzCascade = cv2.CascadeClassifier(
    '../Cascades/haarcascade_frontalface_default.xml'
)

gulCascade = cv2.CascadeClassifier(
    '../Cascades/haarcascade_smile.xml'
)

kamera = cv2.VideoCapture(0)
kamera.set(3,640)
kamera.set(4,480)

while True:
    _, kare = kamera.read()
    kare = cv2.flip(kare,1)
```

Program Çıktısı:



## Video Kamera Üzerinden Kenar Tespiti ve Çizim Yapmak

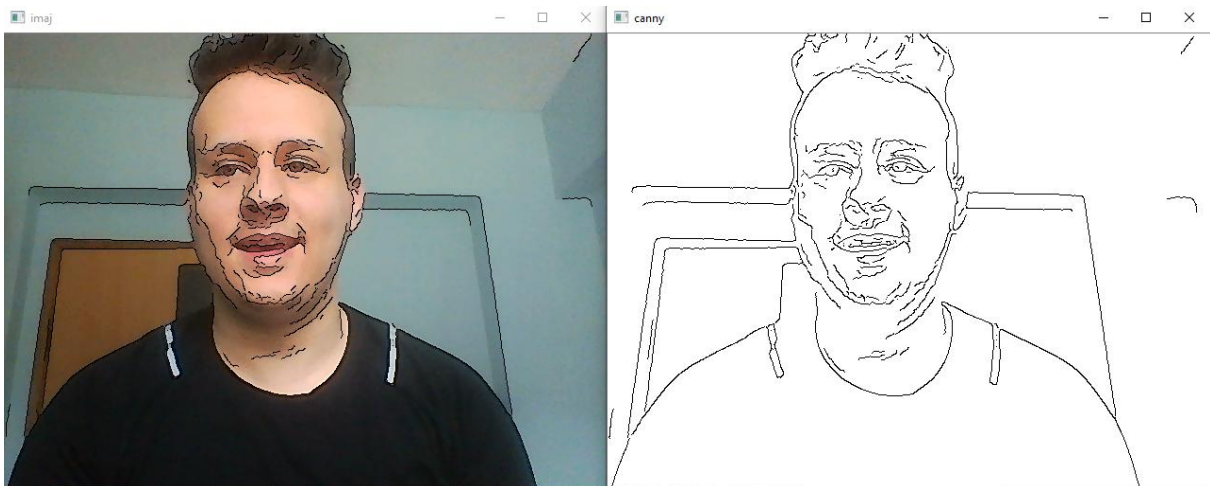
Burada yapacağımız işlem videodaki sınırları bularak bunları çizime çevirmek. Tekrar bak açıklaması için

karakalem01.py

```
import cv2
deltax = 0
deltay = 0
kamera = cv2.VideoCapture(0)
kamera.set(3,640) #3 genişlik
kamera.set(4,480) #4 yüksekliği ifade eder
while True:
    ret, kare = kamera.read()
    kare = cv2.flip(kare,1)
    gri = cv2.cvtColor(kare,cv2.COLOR_BGR2GRAY) #griye çevirme işlemi
    blur = cv2.GaussianBlur(gri,(7,7), 0 ) #bulanıklaştırma işlemi-ayrıntıları
    #azaltmak için ve canny'nin daha iyi sonuç vermesi için
    canny = cv2.Canny(blur,30,50)#kenarlar tespiti için
    canny = cv2.bitwise_not(canny)
    #bitwise_not() dizinin herbir elemanını terse çevirir
    imaj = cv2.bitwise_and(kare,kare,mask = canny)
    #iki resmin bitisel olarak birleşimini sağlar
    cv2.imshow('imaj',imaj)
    cv2.moveWindow('imaj',10,10)
    cv2.imshow('canny',canny)
    cv2.moveWindow('canny',imaj.shape[1]+deltax,10)
    cv2.waitKey(25)

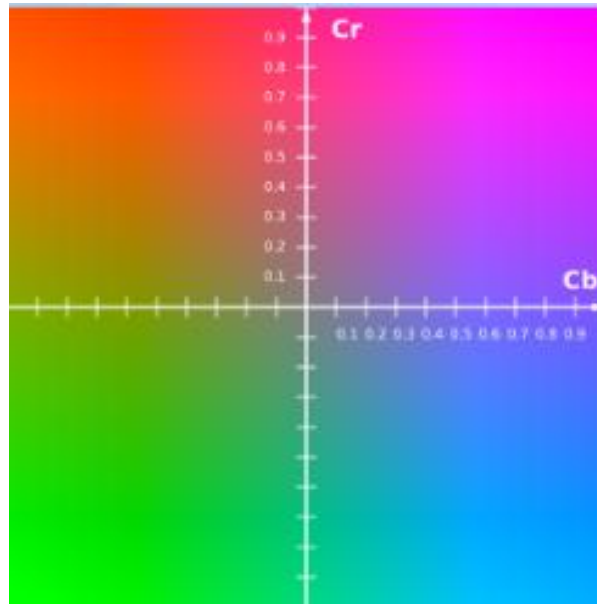
kamera.release()
cv2.destroyAllWindows()
```

Program Çıktısı:



## Ten Rengini Göstermek

Ten rengini tespit etmek için YCrCb renk sistemini kullanacağız.YCrCb renk sistemi dijital komponent videolarda kullanılan renk uzayıdır.Y – parlaklık (Luminance), Cb – Chroma (blue minus luma), Cr – Chroma (red minus luma) değerlerini ifade etmektedir.Dünya çapında sayısal video standardı oluşturmak için ortaya çıkmıştır.Renk bilgisinin sayısal olarak kodlanması için kullanılır.



(YCrCb Renk Uzayı)

Uygulamamızda ilk olarak BGR2YCrCb dönüşümü gerçekleştiriyoruz.Daha sonra renk aralıklarımı veriyoruz.Sonrasında ise morfolojik filtre uyguluyoruz.Morfolojik işlemler genelde iki temel işlemten türetilmiştir.Bunlar erosion(aşındırma) ve dilation(genişletme) işlemleridir.Aşındırma ikili bir görüntüde bulunan nesnelerin boyutunu seçilen yapısal elemente bağlı olarak küçültürken, genişletme nesnenin alanını artırır.Bu işlemlerden erosion işlemi birbirine ince bir gürültü ile bağlanmış iki veya deha fazla nesneyi birbirinden ayırmak için kullanırken,dilation işlemi ise aynı nesnenin bir gürültü ile ince bir şekilde bölünerek ayrı iki nesne gibi görünmesini engellemek için kullanılır.

tenrengi.py

```
import cv2
import numpy as np

deltax = 0
deltay = 0
kamera = cv2.VideoCapture(0)
kamera.set(10, 0.8)#parlaklik

while True:

    ret, kare = kamera.read()
    kare = cv2.flip(kare,1)
    ycrcb = cv2.cvtColor(kare, cv2.COLOR_BGR2YCrCb)#bgr renk uzayinin YCrCb'ye
donusturuyoruz.
    ycrcb = cv2.inRange(ycrcb, (0,137,85),(255,180,135))#renk araliginin
tanimlanmasi
    #morphologyEx() fonksiyonu kendi icerisinde hem erode hem dilate islemini yapar
    ycrcb = cv2.morphologyEx(ycrcb,cv2.MORPH_OPEN,np.ones((3,3),np.uint8))

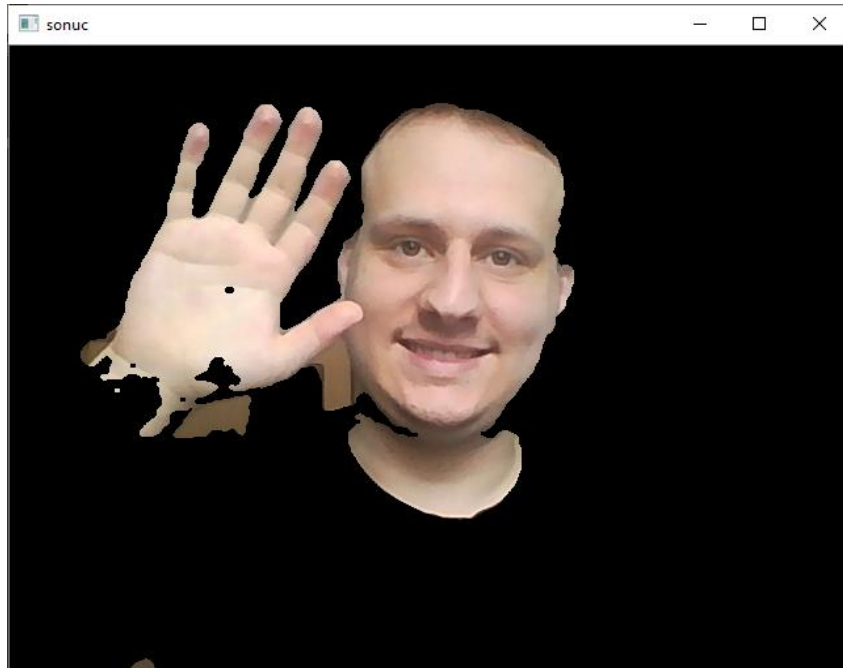
    ...

    cv2.morphologyEx()
Parameters
    src      = Kaynak goruntu
    op       = Morfolojik operasyon tipi
    kernel   = Yapilandirma elemani, morfolojik islemlerin yapilacagi boyut
    ...

    ycrcb = cv2.medianBlur(ycrcb, 5)
    #kucuk ayrintilarin kaybolmasi icin bulaniklastirma islemini yapiyoruz
    sonuc = cv2.bitwise_and(kare,kare,mask = ycrcb)
    #burada bitwise_and arkaplanin siyah olmasi icin kullaniliyor(maskeleme islemi)
    #cv2.imshow('kare',kare)
    #cv2.imshow('maske',ycrcb)
    cv2.imshow('sonuc',sonuc)
    #cv2.moveWindow('kare',10,10)
    #cv2.moveWindow('maske',10,kare.shape[0])
    cv2.moveWindow('sonuc',10,10)
    cv2.waitKey(25)

kamera.release()
cv2.destroyAllWindows()
```

Program Çıktısı:



## Nesne Takibi

OpenCV içerisinde nesne takibi yapabilmek için öncelikle nesnenin ayırt edici özelliği olması gerekir. Biz burada ayırt edici olarak renk kullanacağız. Belirli renk aralıklarımız ile mavi, kırmızı, sarı vb renkleri tespit ederek maskeleme işlemi yapacağız.

ax\_top\_izleme.py

```
import cv2
import imutils

deltax = 0
deltay = 0

GENISLIK = 600
SADECE_MAX = False
#renk araliklarimiz
YESIL = ((29,86,6),(64,255,255))
KIRMIZI = ((139,0,0),(255,160,122))
MAVI = ((110,50,50),(130,255,255))
TURUNCU = ((160,100,47),(179,255,255))
SARI = ((10,100,100),(40,255,255))
altRenk, ustRenk = MAVI# rengimizi mavi olarak aldik
#dahili kamerayi okuma islemi
kamera = cv2.VideoCapture(0)
```

```

cv2.namedWindow('kare')
cv2.moveWindow('kare',10,10)
cv2.namedWindow('maske')
cv2.moveWindow('maske',GENISLIK+deltax,10)
while True:
    (ok,kare) = kamera.read()

    kare = imutils.resize(kare,GENISLIK) #en boy oraninin korunmasini saglar
    genislik degerine gore yuksekligi oranliyor
    hsv = cv2.GaussianBlur(kare,(25,25),0) # detaylari azaltmak icin bulaniklastirma
    hsv = cv2.cvtColor(hsv, cv2.COLOR_BGR2HSV)#bgrdan HSV'ye donusturma islemi

    maske = cv2.inRange(hsv,altRenk,ustRenk)#deger araliklari
    maske = cv2.erode(maske,None,iterations=3)#asindirma islemi,3 kez tekrarlaniyor
    maske = cv2.dilate(maske,None,iterations=3)#genisletme islemi 3 kez
    tekrarlaniyor
    kopya = maske.copy()#maskemizi kopyalıyoruz

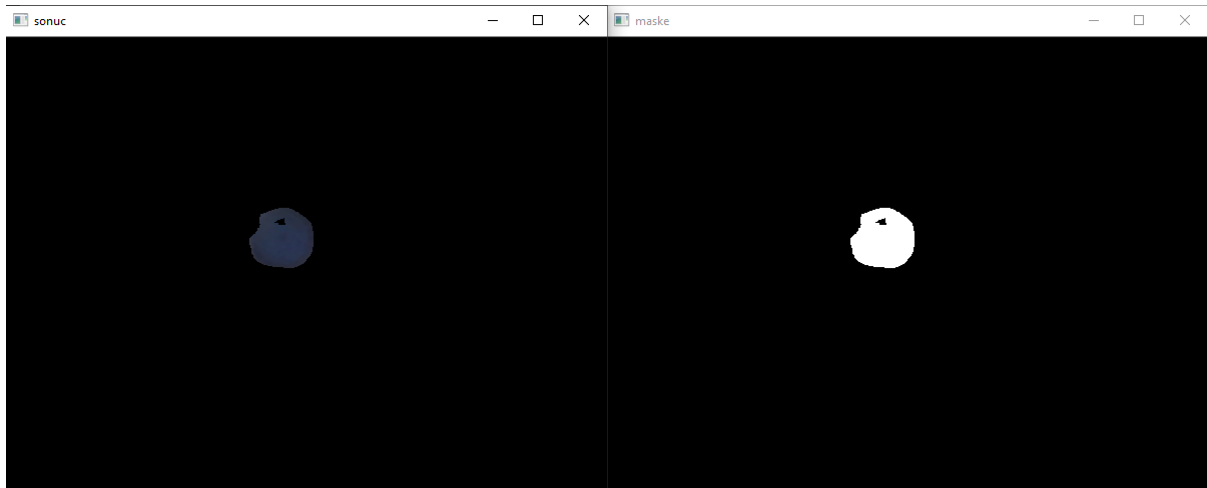
    sonuc = cv2.bitwise_and(kare,kare,mask= maske)#maskenin isleme alinmasi
    cv2.imshow('kare',kare)
    cv2.imshow('maske',maske)
    cv2.imshow('sonuc',sonuc)

    cv2.waitKey(4)

kamera.release()
cv2.destroyAllWindows()

```

## Program Çıktısı



## Nesneyi Çizgi İle Takip Etmek

Burada tespit ettiğimiz nesnenin arkasında kuyruk yapısı oluşturacağız.

ax\_kuyruklu\_top.py

```
import cv2
import imutils
from collections import deque
import numpy as np
#renk araliklarimiz
GENISLIK = 800 # GENISLIK
NOKTA_SAYISI=100 #cizgiyi olusturan nokta sayisinin max 100 olmasini istiyoruz
YESIL = ((29, 86, 6), (64, 255, 255))
KIRMIZI = ((139, 0, 0), (255, 160, 122))
MAVI = ((110, 50, 50), (130, 255, 255))
TURUNCU = ((160, 100, 47), (179, 255, 255))
SARI = ((10, 100, 100), (40, 255, 255))
#rengimizi mavi sectik
altRenk, ustRenk = MAVI

kamera = cv2.VideoCapture(0)
noktalar= deque(maxlen=NOKTA_SAYISI) #boru mekanizmasi
#deque(doubly ended queue) - cift uclu kuyruk hizli ekleme islemi yapar o yuzden
listeye gore tercih edilir
cv2.namedWindow('kare')
cv2.moveWindow('kare', 10, 10)
while True:
    #kamera okuma islemi
    (ok, kare) = kamera.read()
    # en boy oraninin korunmasini saglar genislik degerine gore yuksekligi
    oranliyor
    kare = imutils.resize(kare, GENISLIK)
    #kamerayi aynalama islemi
    kare = cv2.flip(kare,1)
    #detaylari kaybetmek icin blurlama islemi
    hsv = cv2.GaussianBlur(kare, (25,25), 0)
    #bgr renk uzayindan hsv renk uzayina cevirme islemi
    hsv = cv2.cvtColor(hsv, cv2.COLOR_BGR2HSV)
    #deger araliklari
    maske = cv2.inRange(hsv, altRenk, ustRenk)
    maske = cv2.erode(maske, None, iterations=1)#asindirma islemi
    maske = cv2.dilate(maske, None, iterations=1)#genisletme islemi
    kopya = maske.copy()#maskeyi kopyalama
    _,konturlar,_ = cv2.findContours(kopya, cv2.RETR_EXTERNAL,
                                    cv2.CHAIN_APPROX_SIMPLE)
    # cv2.findContours(orjinal imaj = imaj, mod = cv2.RETR_EXTERNAL, metod =
    cv2.CHAIN_APPROX_SIMPLE)
    # mod : kontur bulma yontemi (RETR_LIST,RETR_EXTERNAL,RETR_CCOMP,RETR_TREE)
    # metod : kontur yaklasim yontemi (approximation)
    merkez = None
```



```

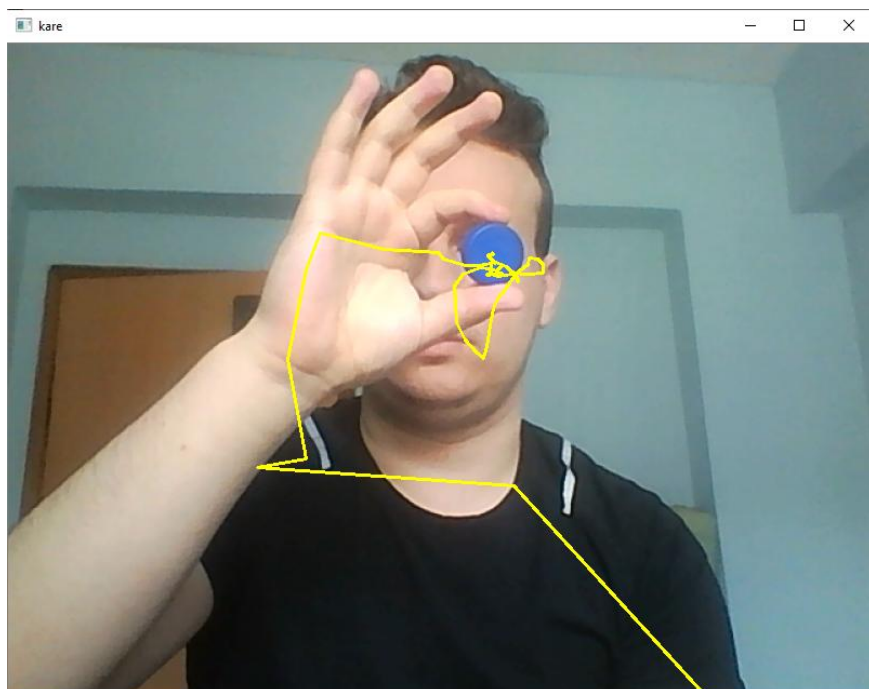
if len(konturlar) > 0:
    cmax = max(konturlar, key=cv2.contourArea)#konturlarin icerisinde en
    buyuk alana sahip olani bulur
    for ctr in konturlar:
        (x, y), yaricap = cv2.minEnclosingCircle(cmax)#cmax'i icerisine
        alabilecek en kucuk cember
        #minEnclosingCircle 2 ciklasi var birisi merkez birisi yaricap
        mts = cv2.moments(cmax) #agirlik hesaplama araci
        merkez = int(mts['m10']/mts['m00']),\
            int(mts['m01']/mts['m00'])
        #agirlik merkezi hesaplama islemi
        if yaricap >= 30: #nesnemizin yaricapi 30dan buyukse etrafina daire
        ciz demis olduk
            cv2.circle(kare, (int(x), int(y)),
                int(yaricap), (255, 255, 0), 4)
            #cv2.circle(imaj,merkez,yaricap,renk,kalinlik)
            noktalar.appendleft(merkez)#appenleft soldan eklemeye basla demek
            for i in range(1,len(noktalar)):
                if noktalar[i] and noktalar[i-1]:

                    cizgi_kal=2 #cizgi kalinligini sabitleme islemi
                    cv2.line(kare,noktalar[i-1],
                        noktalar[i],(0,255,255),cizgi_kal)
                    #kare uzerine cizilecek noktalar[i-1] ile noktalar[i]
                    arasinda sari rengine ve cizgi kalinligi 2

            cv2.imshow("kare", kare)
            key = cv2.waitKey(10) & 0xFF
            if key == ord('q') or key == 27:
                break
            kamera.release()
            cv2.destroyAllWindows()

```

## Program Çıktısı



## Tespit Edilen Renklerin Sayılması

Nesnelerimizi renklerine göre tespit ettiğimize göre artık tespit edilen nesnelerin sayısını bulmak daha kolay. Renk aralığına göre verdiğimiz renk doğrultusunda o renkten kaç tane olduğunu sayabiliriz.

renksay.py

```
import cv2
import numpy as np

# altRenk = np.array([30,60,60])
# ustRenk = np.array([64,255,255])
# RENK='YESIL'
# altRenk=(10, 100, 100)
# ustRenk=(40, 255, 255)
# RENK='SARI'
# altRenk=(170, 100, 100)
# ustRenk=(190, 255, 255)
# RENK='KIRMIZI'
altRenk=(75, 100, 100)
ustRenk=(130, 255, 255)
RENK='MAVi'

kamera = cv2.VideoCapture(0)
#cozunurlugun ayarlanmasi
kamera.set(3,640)
kamera.set(4,480)
cember = True

while True:
    if not kamera.isOpened():break #kamera kontrolu
    _, kare = kamera.read()
    #Bgr renk uzayindan hsv renk uzayina cevirme islemi
    hsv = cv2.cvtColor(kare,cv2.COLOR_BGR2HSV)
    #deger araliklari
    maske = cv2.inRange(hsv,altRenk,ustRenk)
    #cekirdek boyutunun belirlenmesi
    kernel = np.ones((5,5),'int')

    maske = cv2.dilate(maske,kernel)#genisletme islemi
    #konturlarin bulunmasi
    konturlar = cv2.findContours(maske.copy(),cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]
    # cv2.findContours(orjinal imaj = imaj, mod = cv2.RETR_EXTERNAL, metod =
cv2.CHAIN_APPROX_SIMPLE)
    # mod : kontur bulma yontemi (RETR_LIST,RETR_EXTERNAL,RETR_CCOMP,RETR_TREE)
    # metod : kontur yaklasim yontemi (approximation)
    say = 0
```

```

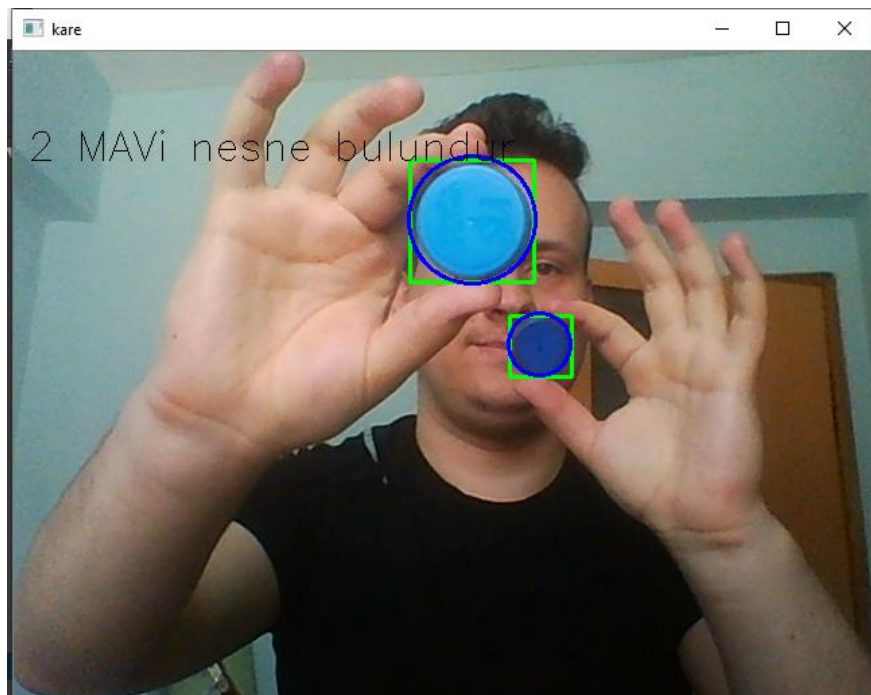
for kontur in konturlar:
    #alan bulma islemi
    alan = cv2.contourArea(kontur)
    #alan 600'den büyükse
    if alan > 600:
        #nesne sayisini 1 arttir
        say += 1
        #boundingRect Bir nokta kümesinin sağ üst sınırlayıcı
        #dikdörtgenini hesaplar.
        (x,y,w,h) = cv2.boundingRect(kontur)
        #dikdörtgen çizme islemi
        cv2.rectangle(kare,(x,y),(x+w,y+h),(0,255,0),2)
        #cember true ise yukarida true yaptik
        if cember:
            #en küçük cemberin merkezini ve capini buluyor
            (x,y), ycap = cv2.minEnclosingCircle(kontur)
            merkez = (int(x),int(y))
            ycap = int(ycap)
            #cember çizme islemi
            img = cv2.circle(kare,merkez,ycap,(255,0,0),2)
        #sayi 0'dan büyükse ekrana ve terminale yazı yazdır
        if say > 0:
            cv2.putText(kare,f"{say} {RENK} nesne bulundu",(10,80),
            cv2.FONT_HERSHEY_SIMPLEX,1,1)

    cv2.imshow('kare',kare)
    k = cv2.waitKey(4) & 0xFF
    if k == 27: break

if kamera.isOpened():
    kamera.release()
cv2.destroyAllWindows()

```

## Program Çıktısı



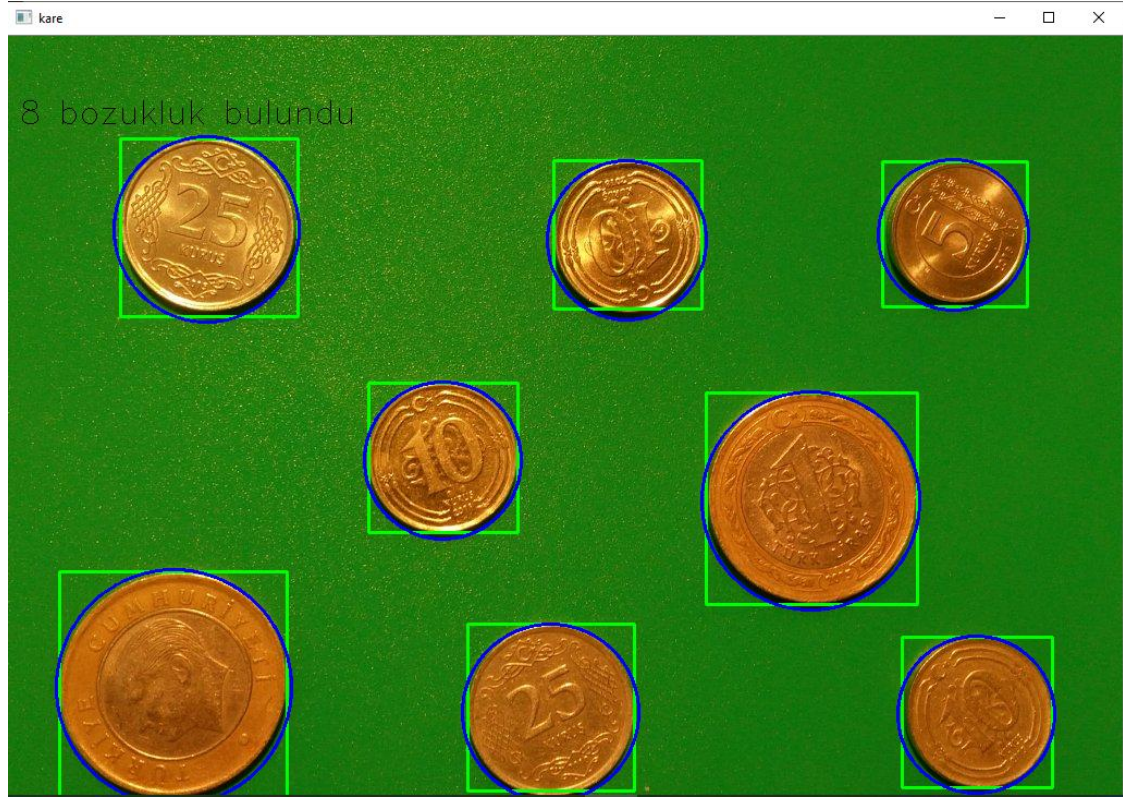
## Bozuk Para Tespiti Ve Sayılması

Burada bozuk paraları bulmak için kenar ve sınırlarından faydalanacağız. Nesneler tespit edildikten sonra boyutları hesaplanacak (bu şekil değeri belirleme işlemi yapılabilir) ve etrafına çember çizilerek kaç tane bozukluk olduğu ekrana yazdırılacak.

parasay.py

```
import cv2
cember = True
#resmi okumak
kare = cv2.imread('../Resimler/paralar04.jpg')
#ayrintilari kaybetmek (resmi yumusatmak)
blur = cv2.GaussianBlur(kare,(3,3),0)
#kenarlari bulmak icin
canny = cv2.Canny(blur,30,250)
#cv2.imshow("Canny",canny)
#cv2.waitKey(0)
#cekirdik boyutu getStructuringElement yapilandirma elemanin olusmasini saglar
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(7,7))
#morphologyEx islemi hem resme hem erosion hem dilation islemi uygular
morf = cv2.morphologyEx(canny,cv2.MORPH_CLOSE,kernel)
#cv2.imshow('morf',morf)
#cv2.waitKey(0)
#sinirlarin bulunmasi
konturlar = cv2.findContours(morf.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
[-2]
#bulunan nesneler baslangicta 0
say = 0
#konturlar bir liste elemani
for kontur in konturlar:
    #alanlari bulunmasi
    alan = cv2.contourArea(kontur)
    #alan degeri belirli bir boyuttan fazlaysa sayma islemini gerceklestir
    if alan > 10000:
        print(alan)
        say+=1
        #seklin sag ust sinirlayici dortgenini tutar
        (x,y,w,h) = cv2.boundingRect(kontur)
        #dikdortgen cizme islemi
        cv2.rectangle(kare,(x,y),(x+w,y+w),(0,255,0),2)
        #cember ifademiz true idi yani dikdortgen varsa o bizim icin paranin oldugu
        alandir
        if cember:
            #cember sinirlarinin bulunmasi islemi
            (x,y), ycap = cv2.minEnclosingCircle(kontur)
            merkez = (int(x),int(y))
            ycap = int(ycap)
            #dairenin cizilmesi islemi
            img = cv2.circle(kare,merkez,ycap,(255,0,0),2)
#bozukluklari sayma islemi
if say > 0:
    cv2.putText(kare,f"{say} bozukluk bulundu", (10,80),cv2.FONT_HERSHEY_SIMPLEX,1,1)
cv2.imshow('kare',kare)
cv2.waitKey(0)
```

Program Çıktısı:



```
15382.0
19613.0
34891.5
30254.5
15593.0
14055.0
15753.0
21162.5
```

(Bozukluların alanları)



## Yüz Tanıma

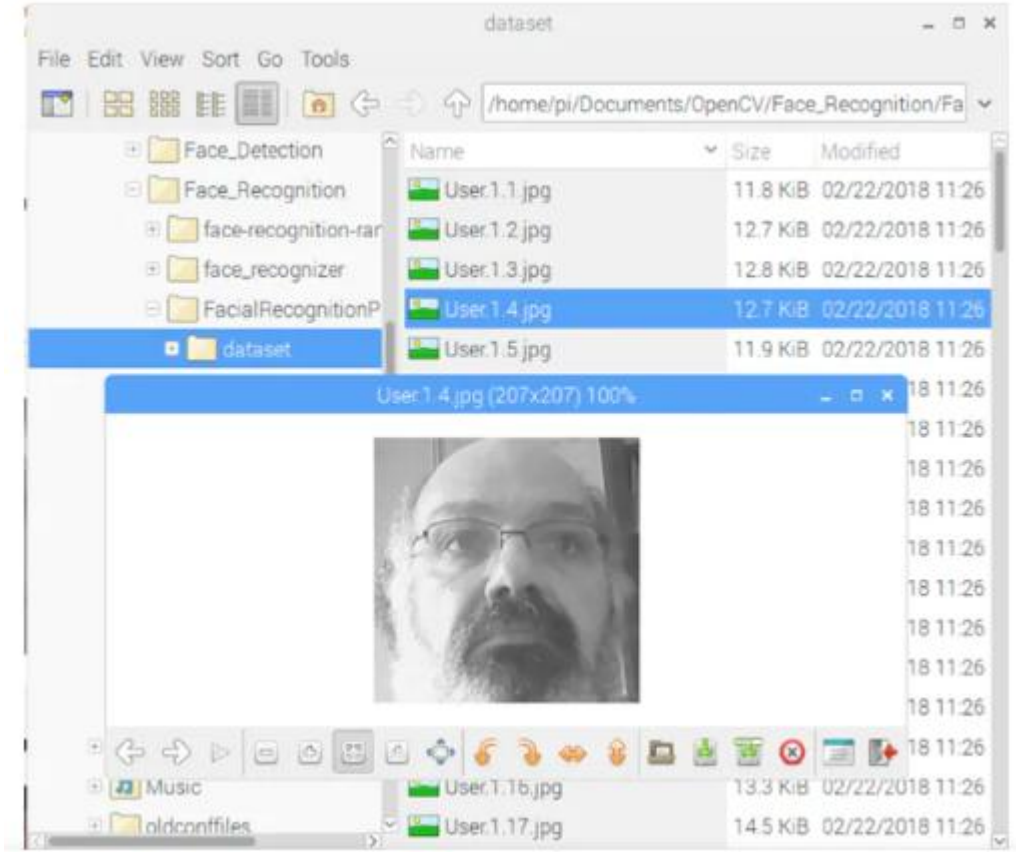
OpenCv içerisinde çeşitli yöntemler ve kütüphaneler kullanılarak yüz tanıma yapılabilir. Biz burada tanıyıcı olarak, OpenCV paketine dahil olan LBPH (Yerel İkili Kalıp Histogramı) Yüz tanıyıcı olarak kullanacağız.

Yüz tanımanın 3 adımı var diyebiliriz. Bunlardan birincisi veri toplama. Yapacağımız işlem videodan canlı olarak yüzümü yakalayıp bunları gri tonajda ve sadece yüzümüz şeklinde kırparak bir klasör içerisinde biriktirecek. Daha detaylı anlatmak gerekirse:

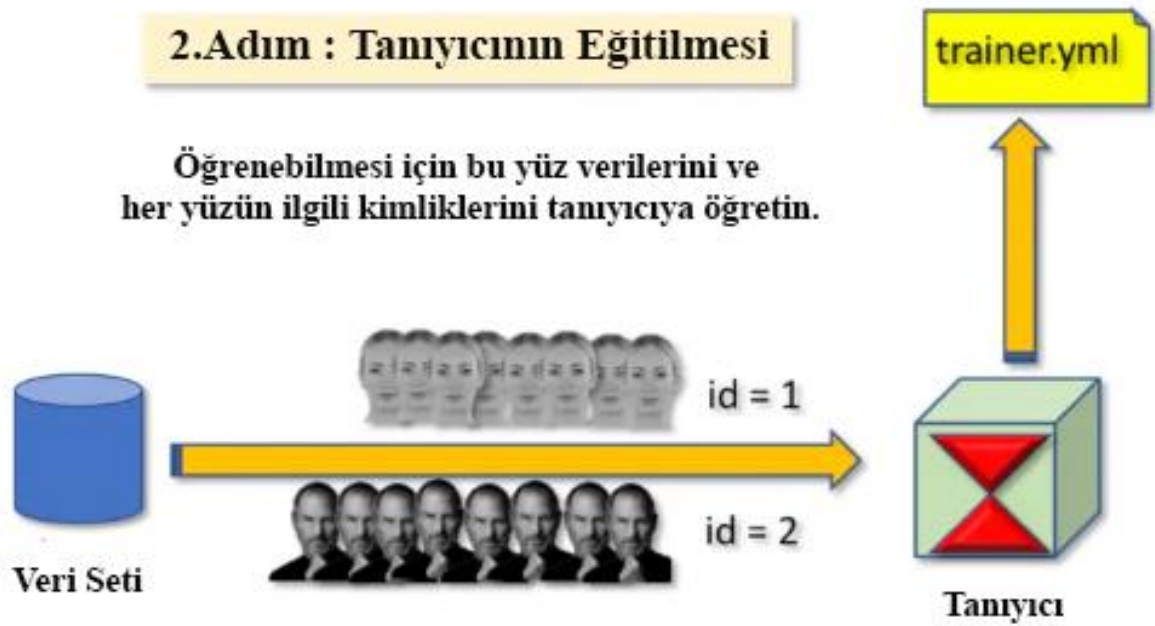
### 1.Adım : Veri Toplama



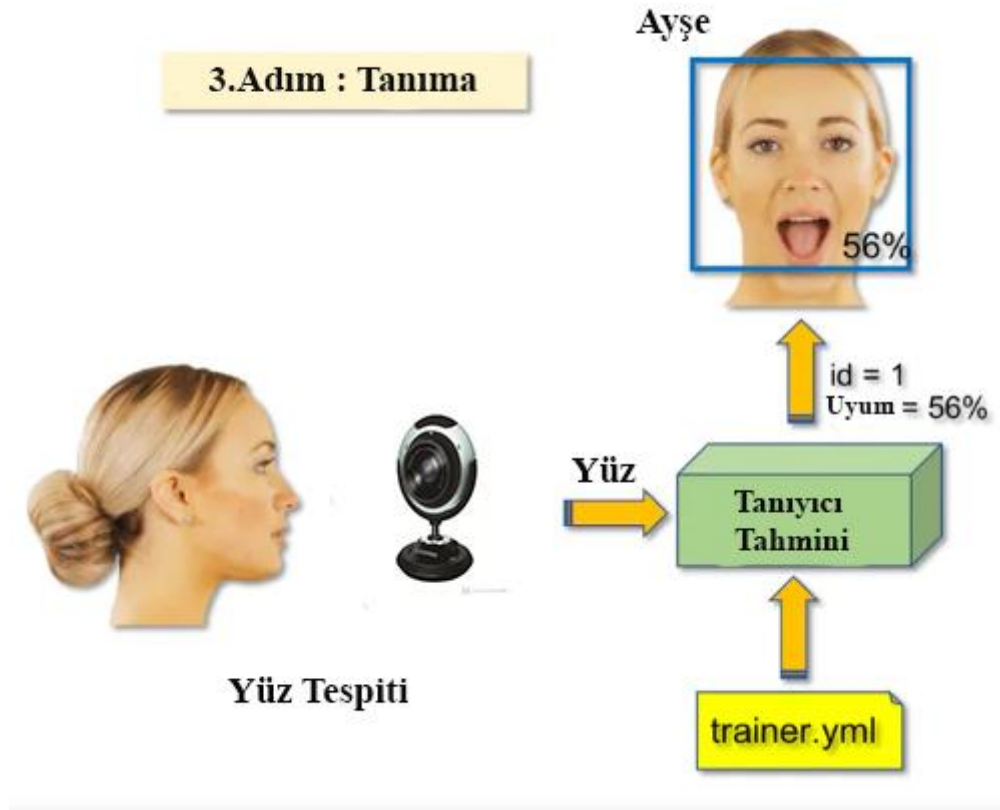
**Alınan fotoğraflar nasıl görünüyor :**



**İkinci adım olarak tanıyıcının veri setimiz ile eğitilmesi var :**



**Artık tanıyıcımızdan tanımladığımız yüzü tanımasını bekliyoruz :**



## Anlatılanları Uygulamak

İlk olarak veri toplama işlemini yapmamız lazım.Kameramızı açarak videodan yüz yakalama işlemini gerçekleştirecez.Proje çalıştırılırken anlatıldığı sırayla çalıştırılması gerekir.

1-yuz\_veriseti.py

```
import cv2

kamera = cv2.VideoCapture(0)
kamera.set(3, 640) # video genişliğini belirle
kamera.set(4, 480) # video yüksekliğini belirle
face_detector =
cv2.CascadeClassifier('Cascades/haarcascade_frontalface_default.xml')
# Her farklı kişi için farklı bir yüz tamsayısı ata
# face_id = input('\n enter user id end press <return> ==> ')
MAXFOTOSAY = 50 # Her bir yüz için kullanılacak imaj sayısı
face_id = 1
print("\n [INFO] Kayıtlar başlıyor. Kameraya bak ve bekle ...")

say = 0
```



```

while(True):
    ret, img = kamera.read()
    # img = cv2.flip(img, -1) # gerekiyorsa kullan
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    yuzler = face_detector.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in yuzler:
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
        say += 1
        # Yakalanan imajı veriseti klasörüne kaydet
        cv2.imwrite("veriseti/" + str(face_id) + '.' + str(say) + ".jpg",
gray[y:y+h,x:x+w])
        cv2.imshow('imaj', img)
        print("Kayıt no: ",say)
        k = cv2.waitKey(100) & 0xff
        if k == 27:
            break
        elif say >= MAXFOTOSAY:
            break
# Belleği temizle
print("\n [INFO] Program sonlanıyor ve bellek temizleniyor.")
kamera.release()
cv2.destroyAllWindows()

```

## 2-yuz\_egitimi.py

```

from cv2 import *

import numpy as np
from PIL import Image
import os

# Verilerin yolu
path = 'veriseti'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("Cascades/haarcascade_frontalface_default.xml")

# imajların alınması ve etiketlenmesi için fonksiyon
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    ornekler=[]
    ids = []
    for imagePath in imagePaths:
        PIL_img = Image.open(imagePath).convert('L') # gri
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.splitext(imagePath)[-1].split(".")[0])
        # print("id= ",id)
        yuzler = detector.detectMultiScale(img_numpy)
        for (x,y,w,h) in yuzler:
            ornekler.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)
    return ornekler,ids
print ("\n [INFO] yuzler eğitiliyor. Birkaç saniye bekleyin ...")

```

```

yuzler,ids = getImagesAndLabels(path)
recognizer.train(yuzler, np.array(ids))
# Modeli egitim/egitim.yml dosyasına kaydet
recognizer.write('egitim/egitim.yml') # Dikkat! recognizer.save() Raspberry Pi
# üzerinde çalışmıyor
# Eğitilen yüz sayısını göster ve kodu sonlandır
print(f"\n [INFO] {len(np.unique(ids))} yüz eğitildi. Betik sonlandırılıyor.")

# print(yuzler)

```

### 3-yuz\_tanima.py

```

import cv2
import numpy as np
from PIL import Image, ImageDraw, ImageFont

def print_utf8_text(image, xy, text, color): # utf-8 karakterleri
    fontName = 'FreeSerif.ttf' # 'FreeSansBold.ttf' # 'FreeMono.ttf'
    'FreeSerifBold.ttf'
    font = ImageFont.truetype(fontName, 24) # font seçimi
    img_pil = Image.fromarray(image) # imajı pillow moduna dönüştür
    draw = ImageDraw.Draw(img_pil) # imajı hazırla
    draw.text((xy[0],xy[1]), text, font=font,
              fill=(color[0], color[1], color[2], 0)) # b,g,r,a
    image = np.array(img_pil) # imajı cv2 moduna çevir (numpy.array())
    return image

recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('egitim/egitim.yml')
cascadePath = "Cascades/haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
font = cv2.FONT_HERSHEY_SIMPLEX
# id sayacını başlat
id = 0
names = ['None', 'Tuğrul']

# Canlı video yakalamayı başlat
kamera = cv2.VideoCapture(0)
kamera.set(3, 1000) # video genişliğini belirle
kamera.set(4, 800) # video yüksekliğini belirle
# minimum pencere boyutunu belirle
minW = 0.1 * kamera.get(3) # genişlik
minH = 0.1 * kamera.get(4) # yükseklik
while True:
    ret, img = kamera.read()
    gri = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    yuzler = faceCascade.detectMultiScale(
        gri,
        scaleFactor=1.2,
        minNeighbors=5,
        minSize=(int(minW), int(minH)),
    )

```

```

for (x, y, w, h) in yuzler:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
    id, uyum = recognizer.predict(gri[y:y + h, x:x + w])

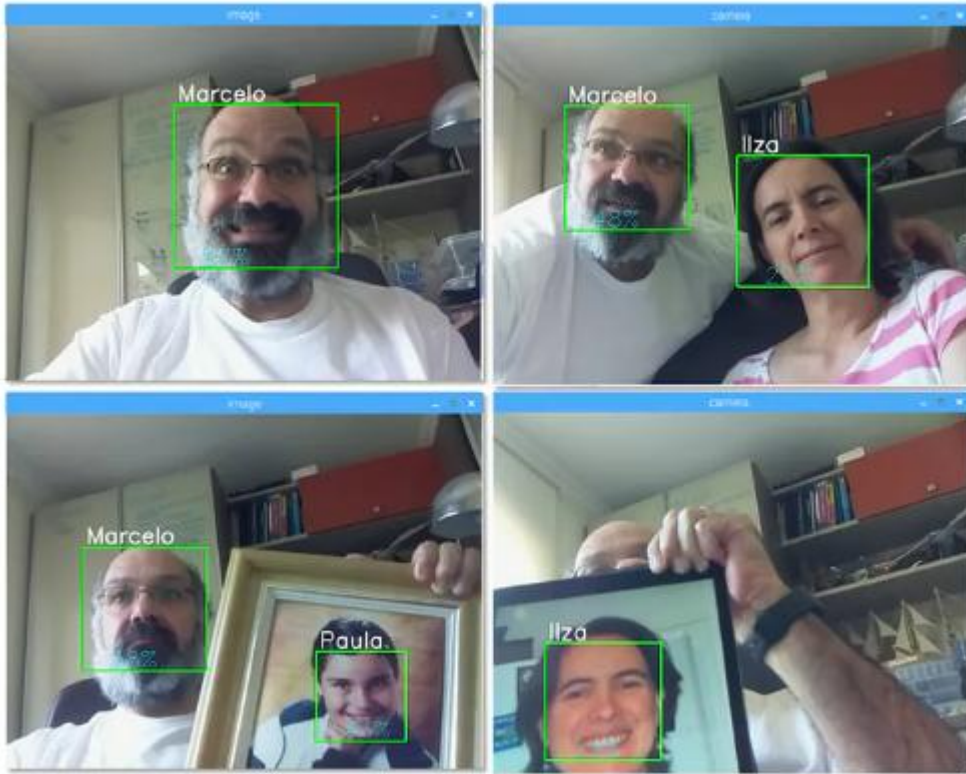
    if (uyum < 100):
        id = names[id]
        uyum = f"Uyum= {round(uyum,0)}%"
    else:
        id = "bilinmiyor"
        uyum = f"Uyum= {round(uyum,0)}%"

    color = (255,255,255)
    img=print_utf8_text(img,(x + 5, y - 25),str(id),color) # Türkçe
    karakterler
    # cv2.putText(img, str(id), (x + 5, y - 5), font, 1, (255, 255, 255), 2)
    cv2.putText(img, str(uyum), (x + 5, y + h + 25), font, 1, (255, 255, 0),
1)

cv2.imshow('kamera', img)
k = cv2.waitKey(10) & 0xff # Çıkış için Esc veya q tuşu
if k == 27 or k==ord('q'):
    break
# Belleği temizle
print("\n [INFO] Programdan çıkıyor ve ortalığı temizliyorum")
kamera.release()
cv2.destroyAllWindows()

```

Program Çıktısı:



## Sonuç :

Proje içerisinde kısa kodlamalar ile büyük ve etkili projeler yapılabileceği gösterilmiştir.OpenCv'nin birçok fonksiyonuna değinilerek ve fonksiyonların çalışma mantıklarını açıklayarak, parametre ve çıktılarını inceleyerek projeler yapılmıştır.Proje dosyalarına <https://github.com/turulkok/OpenCV> adresinden ulaşılabilir.

## Kaynakça

- Aksoy, Ahmet. *OpenCV ve Python ile Eğlenceli Projeler ve Oyunlar*. Abaküs Yayınları, İstanbul: 2019.
- <https://www.youtube.com/user/ahmetax54> Youtube Kanalı
- <https://python.gurmezin.com/> İnternet Sitesi
- <https://www.hackster.io/mjrobot/real-time-face-recognition-an-end-to-end-project-a10826> İnternet Sitesi