OPENCV İLE GÖRÜNTÜ İŞLEME

```
open cv ile görüntü tanımlama

import cv2 as cv



path = r"C:\Users\ridva\Desktop\my-project\python\turcell_opencv\ronaldo.jpg"


# Resmi okuma
img = cv.imread(path +"ronaldo.jpg")


type(img)


#namewindow


cv.namedWindow("opencv", cv.WINDOW_AUTOSIZE)
#imshow --->resimi göter


cv.imshow("opencv",img)
cv.waitKey(1)


cv.destroyAllWindows()
#açıkta pencere var ise hepsini kapatır

import cv2 as cv


path = r"C:\Users\ridva\Desktop\my-project\python\turcell_opencv\ronaldo.jpg"


# Resmin doğru okunması için path kullanılıyor
img = cv.imread(path)
cv.namedWindow("colored", cv.WINDOW_AUTOSIZE)
cv.imshow("colored", img)
cv.waitKey(1)


# cvtColor -------> resmin formatını değiştirmek için kullanılır
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow("gray", gray)
cv.waitKey()


# Gri tonlamalı resmi kaydetme
cv.imwrite(path.replace(".jpg", "_gray_opencv.jpg"), gray)
```

```
# Tüm pencereleri kapat
cv.destroyAllWindows()

# Gri tonlamalı resmi doğrudan okuma
img = cv.imread(path, cv.IMREAD_GRAYSCALE)
cv.imshow("gray", img)
cv.waitKey(1)+
```

kodun detayları:

- Resmi Okuma ve Gösterme

img = cv.imread(path)
cv.namedWindow("colored", cv.WINDOW_AUTOSIZE)
cv.imshow("colored", img)
cv.waitKey(0)

- `cv.imread(path)`: Belirtilen dosya yolundaki resmi okur.
- `cv.imshow()`: Resmi gösterir.
- `cv.waitKey(0)`: Tuşa basılana kadar bekler. Burada `0` parametresi sonsuz beklemeyi ifade eder, yani tuşa basılana kadar pencere açık kalır.

- NumPy ile Kopyalama ve Manipülasyon

m1 = np.copy(img)
m2 = img

- `np.copy(img)`: `img` resminin derin bir kopyasını oluşturur. `m1` değişkeni, `img`'in bağımsız bir kopyasıdır.
- `m2 = img`: `m2`, `img`'in sadece bir referansıdır, yani `img`'deki herhangi bir değişiklik `m2`'yi de etkiler.

- Belirli Bir Bölümü Sıfırlama

img[100:200, 200:300, :] = 0
Bu kod, resmin `(100, 200)` ile `(200, 300)` arasındaki bölgesini siyah yapar (`0` değerini atamak, siyah piksel anlamına gelir). Bu işlem, BGR renk modeline göre, mavi, yeşil ve kırmızı bileşenlerini sıfırlayacaktır.

- Yeni Boş (Siyah) Görüntüler Oluşturma

m3 = np.zeros(img.shape, img.dtype)
cv.imshow("m3", m3)

- `np.zeros(img.shape, img.dtype)`: Resmin aynı boyutlarında ancak tüm pikselleri sıfır (siyah) olan yeni bir resim oluşturur.
- `cv.imshow("m3", m3)`: Bu yeni siyah resmi gösterir.

- 512x512 Boyutunda Siyah ve Gri Resimler

m4 = np.zeros([512, 512], np.uint8)
cv.imshow("m4", m4)

- `np.zeros([512, 512], np.uint8)`: 512x512 boyutunda, tüm pikselleri siyah olan bir resim oluşturur.
- m5 = np.zeros([512, 512], np.uint8)

m5[:, :] = 127

Bu kod, 512x512 boyutunda, her pikselin gri tonlarında (127 değeri) olduğu bir resim oluşturur.

- Beyaz Arka Plan ve Şekiller Çizme

img = np.ones((550, 770, 3), dtype=np.uint8) * 255 # Beyaz arka plan

- u satırda, 550x770 boyutunda, tüm pikselleri beyaz (255 değerinde) olan bir resim oluşturuluyor.
- `np.ones((550, 770, 3), dtype=np.uint8)` ile beyaz bir arka plan için tüm pikselleri 255 (beyaz) olarak ayarladık.

- Dikdörtgenler ve Çizgiler Çizme

cv.rectangle(img, (480, 250), (100, 450), black, 8) # Dikdörtgen

cv.rectangle(img, (580, 150), (200, 350), black, 8) # Dikdörtgen

cv.line(img, (100, 450), (200, 350), black, 8) # Çizgi

- `v.rectangle()`: Resme dikdörtgen çizer. Renk `black` ve kalınlık `8` olarak belirlenmiş.
- `cv.line()`: Resme çizgi çizer, yine siyah renkte ve kalınlığı `8`.

- Metin Ekleme

start_point = (150, 100) # Yazının başlangıç noktası

font_thickness = 2 # Yazının kalınlığı

font_size = 1 # Boyutu

font = cv.FONT_HERSHEY_DUPLEX # Font türü

cv.putText(img, '*www.google.com*', start_point, font, font_size, black, font_thickness)

`cv.putText()`: Resme metin ekler. `start_point` metnin başlangıç koordinatıdır, `font` font türüdür ve `font_size` boyutudur.

Burada `www.google.com` metni resme ekleniyor.

Sonuç

Son olarak, dikdörtgen, çizgi ve metin eklenmiş resmi `cv.imshow()` ile görüntülüyoruz ve `cv.waitKey(0)` ile tuşa basılmasını bekliyoruz.

cv.destroyAllWindows()

`cv.destroyAllWindows()`: Açık olan tüm penceleri kapatır.

kod:

```
import cv2 as cv
```

```
path = "C:\\Users\\ridva\\Desktop\\my-project\\python\\turcell_opencv\\opencv.png"
img = cv.imread(path)
cv.namedWindow("img", cv.WINDOW_AUTOSIZE)
h, w, ch = img.shape
print("Height, Width, Channels:", h, w, ch)
cv.imshow("img", img)


for row in range(h):
    for col in range(w):
        b, g, r = img[row, col]
        b = 255 - b
        g = 255 - g
        r = 255 - r
        img[row, col] = [b, g, r]


cv.imshow("outpout:",img)
cv.waitKey(0)
```

kodun detayları:

`cv.namedWindow("img", cv.WINDOW_AUTOSIZE)` komutu, görüntü için bir pencere oluşturur ve pencere boyutunu görüntünün boyutuna göre otomatik olarak ayarlar.

Görüntü Boyutunun Alınması:

`h, w, ch = img.shape` komutu, resmin yüksekliğini (`h`), genişliğini (`w`) ve renk kanal sayısını (`ch`) alır.

- `h` resmin yüksekliği (satır sayısı),
- `w` resmin genişliği (sütun sayısı),
- `ch` ise renk kanal sayısı (genellikle 3: RGB).

Görüntü Gösterimi:

`cv.imshow("img", img)` komutu, orijinal görüntüyü "img" adıyla bir pencerede gösterir.

Renklerin Tersine Çevrilmesi:

Kodda yer alan `for` döngüleri, her bir pikseli sırayla ziyaret eder. Bu döngülerin içinde:

- `b, g, r = img[row, col]` komutu, ilgili pikselin mavi (`b`), yeşil (`g`) ve kırmızı (`r`) renk değerlerini alır.
- `b = 255 - b`, `g = 255 - g`, `r = 255 - r` komutları, bu renk değerlerinin tersini alır (yani, renklerin "negatifini" oluşturur).
- Son olarak, `img[row, col] = [b, g, r]` komutuyla ters çevrilen renk değerleri ilgili piksele uygulanır.

Sonuç Görüntüsünün Gösterimi:

`cv.imshow("output:", img)` komutu, renkleri tersine çevrilmiş resmi "output:" başlığıyla gösterir.

Tuşa Basmayı Bekleme:

`cv.waitKey(0)` komutu, kullanıcı bir tuşa basana kadar programın durmasını sağlar. Bir tuşa basıldığında pencere kapanır.

verilen iki resimi birleştirme kodu :

```
import cv2 as cv
import numpy as np

# Dosya yolunu belirleyin
path = "C:\\Users\\ridva\\Desktop\\my-project\\python\\turcell_opencv\\"

# Resimleri yükleyin
img1 = cv.imread(path + "sag.png")
img2 = cv.imread(path + "sol.png")

# Resimleri kontrol edin ve aynı yüksekliğe getirin
if img1 is None or img2 is None:
    print("Bir veya daha fazla resim yüklenemedi.")
else:
    if img1.shape[0] != img2.shape[0]:
        img2 = cv.resize(img2, (img1.shape[1], img1.shape[0]))

    # Resimleri gösterin
    cv.imshow("sagel", img1)
    cv.waitKey(1)
    cv.imshow("solel", img2)
    cv.waitKey(1)

    # Resimleri yatay olarak birleştirin ve gösterin
    horizontal = np.hstack((img2, img1))
    cv.imshow("birlesmis_resim", horizontal)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

Resimlerin Yüklenip Yüklenmediğini Kontrol Etme:

```
if img1 is None or img2 is None:
print("Bir veya daha fazla resim yüklenemedi.")
```

Bu blok, resimlerin doğru şekilde yüklenip yüklenmediğini kontrol eder. Eğer herhangi bir resim yüklenememişse, hata mesajı yazdırır.

Resim Yüksekliğinin Eşitlenmesi:

else:

if img1.shape[0] != img2.shape[0]:

img2 = cv.resize(img2, (img1.shape[1], img1.shape[0]))

Eğer resimlerin yükseklikleri aynı değilse, bu satırlar ikinci resmi birincinin yüksekliğine ve genişliğine göre yeniden boyutlandırır.

Resimlerin Gösterilmesi:

cv.imshow("sagel", img1)

cv.waitKey(1)

cv.imshow("solel", img2)

cv.waitKey(1)

Bu satırlar, iki resmi ayrı ayrı pencerelerde gösterir. `cv.waitKey(1)` fonksiyonu, gösterilen resmin bir süre (milisaniye cinsinden) görüntülenmesini sağlar.

Resimlerin Yatay Olarak Birleştirilmesi ve Gösterilmesi: horizontal = np.hstack((img2, img1))

cv.imshow("birlesmis_resim", horizontal)

cv.waitKey(0)

cv.destroyAllWindows()

Bu satırlar, `np.hstack` fonksiyonunu kullanarak iki resmi yatay olarak birleştirir ve birleştirilmiş resmi gösterir. `cv.waitKey(0)` fonksiyonu, kullanıcı bir tuşa basana kadar resmi görüntülemeye devam eder. Son olarak, `cv.destroyAllWindows()` fonksiyonu, açılan tüm pencereleri kapatır.

kod:

```
import cv2 as cv
import numpy as np

path = "C:\\Users\\ridva\\Desktop\\my-project\\python\\turcell_opencv\\"

# Resmi gri tonlamalı olarak yükleyin
src = cv.imread(path + "opencv.png", cv.IMREAD_GRAYSCALE)

# Minimum ve maksimum değerleri bulun (dönüşümdüri dört değeri de alıyoruz)
min_value, max_value, min_loc, max_loc = cv.minMaxLoc(src)

# Minimum ve maksimum değerleri yazdırma
print("min_value: %.2f, max_value: %.2f" % (min_value, max_value))
print("min loc:", min_loc, ",", "max loc:", max_loc)

# Ortalama ve standart sapmayı hesaplayın
means, stddev = cv.meanStdDev(src)
print("mean: %.2f, stddev: %.2f" % (means[0][0], stddev[0][0]))
```

```
# Piksel değerlerini eşikleyin
src[np.where(src < means[0][0])] = 0
src[np.where(src > means[0][0])] = 255

# İkili görüntüyü gösterin
cv.imshow("binary", src)
cv.waitKey(0)
cv.destroyAllWindows()
```

kod :

```
# Piksel Normalizasyonlaştırma
import cv2 as cv
import numpy as np

# Dosya yolunu belirleyin
path = "C:\\Users\\ridva\\Desktop\\my-project\\python\\turcell_opencv\\"

# Resmi yükleyin
src = cv.imread(path + "opencv.png")
print(src.shape)

# Gri tonlamalı resme dönüştürme
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
cv.imshow("gray", gray)
cv.waitKey(0)
print(gray.shape)

# Gri tonlamalı resmin float32 veri tipine dönüştürülmesi
gray = np.float32(gray)
print(gray)

# Min ve max normalizasyon, verilen 2 değer arasında normalizasyon yapar
min_value, max_value, min_loc, max_loc = cv.minMaxLoc(gray)
print("min_value: %.2f, max_value: %.2f" % (min_value, max_value))

means, stddev = cv.meanStdDev(gray)
print("mean: %.2f, stddev: %.2f" % (means[0][0], stddev[0][0]))

# Normalizasyon
dst = np.zeros(gray.shape, dtype=np.float32)
cv.normalize(gray, dst=dst, alpha=0, beta=1.0, norm_type=cv.NORM_MINMAX)
print(dst)
```

```
# Normalizasyonlu resmi gösterin
cv.imshow("NORM_MINMAX", dst)
cv.waitKey(0)
dst=np.zeros(gray.shape,dtype=np.float32)
cv.normalize(gray,dst=dst,alpha=1.0,beta=0,norm_type=cv.NORM_INF)
print(dst)
cv.imshow("norm_inf",np.uint8(dst*255))
cv.waitKey(0)



dst=np.zeros(gray.shape,dtype=np.float32)
cv.normalize(gray,dst=dst,alpha=1.0,beta=0,norm_type=cv.NORM_L1)
print(dst)
cv.imshow("NORM_L1",np.uint8(dst*1000000))
cv.waitKey(0)

dst=np.zeros(gray.shape,dtype=np.float32)
cv.normalize(gray,dst=dst,alpha=1.0,beta=0,norm_type=cv.NORM_L2)
print(dst)
cv.imshow("NORM_L2",np.uint8(dst*1000000))
cv.waitKey(0)



cv.destroyAllWindows()
```

Gri Tonlamaya Dönüştürme:

gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
cv.imshow("gray", gray)
cv.waitKey(0)
print(gray.shape)

- `cv.cvtColor()`: BGR (Blue-Green-Red) renk uzayındaki resmi, gri tonlamalı (grayscale) bir resme dönüştürür.
- `cv.imshow()`: Bu fonksiyonla görseli ekranda gösterir.
- `cv.waitKey(0)`: Ekranda açık kalan görüntüyü görmek için bir tuşa basana kadar bekler.
- `gray.shape`: Gri tonlamalı resmin boyutlarını yazdırır.

Veri Tipini `float32`'ye Dönüştürme:

gray = np.float32(gray)
print(gray)

Minimum ve Maksimum Değerlerin Bulunması:

min_value, max_value, min_loc, max_loc = cv.minMaxLoc(gray)

print("min_value: %.2f, max_value: %.2f" % (min_value, max_value))

- `cv.minMaxLoc()`: Gri tonlamalı resmin minimum ve maksimum piksel değerlerini, bunların konumlarını döndürür.
- `min_value` ve `max_value`: Resimdeki en küçük ve en büyük piksellerin değerlerini gösterir.

Ortalama ve Standart Sapmanın Hesaplanması:

means, stddev = cv.meanStdDev(gray)

print("mean: %.2f, stddev: %.2f" % (means[0][0], stddev[0][0]))

- `cv.meanStdDev()`: Gri tonlamalı resmin ortalama ve standart sapma değerlerini hesaplar.
- `means[0][0]` ve `stddev[0][0]`: Ortalamayı ve standart sapmayı ekrana yazdırır.

Normalizasyon İşlemleri:

Bu adımda resim üzerinde çeşitli normalizasyon türleri uygulanmaktadır.

7.1. Min-Max Normalizasyonu:

dst = np.zeros(gray.shape, dtype=np.float32)

cv.normalize(gray, dst=dst, alpha=0, beta=1.0, norm_type=cv.NORM_MINMAX)

print(dst)

- `cv.normalize()`: Bu fonksiyonla, resmin piksellerini belirli bir aralığa (bu örnekte 0 ile 1 arasına) normalleştirir. `alpha` parametresi alt sınırı, `beta` üst sınırı belirler.
- `cv.NORM_MINMAX`: Bu normalizasyon türü, resmin minimum değerini `alpha`'ya ve maksimum değerini `beta`'ya yerleştirir.

Sonsuz Norm (Infinity Norm):

dst = np.zeros(gray.shape, dtype=np.float32)

cv.normalize(gray, dst=dst, alpha=1.0, beta=0, norm_type=cv.NORM_INF)

print(dst)

cv.imshow("norm_inf", np.uint8(dst*255))

cv.waitKey(0)

- `cv.NORM_INF`: Sonsuz norm, verilerin maksimum mutlak değeriyle ölçeklendirildiği bir normalizasyon türüdür.
- `np.uint8(dst*255)`: Piksel değerlerini 0-255 aralığına dönüştürmek için normalizasyon sonucunu çarpar.

L1 Norm:

dst = np.zeros(gray.shape, dtype=np.float32)

cv.normalize(gray, dst=dst, alpha=1.0, beta=0, norm_type=cv.NORM_L1)

print(dst)

cv.imshow("NORM_L1", np.uint8(dst*1000000))

cv.waitKey(0)

`cv.NORM_L1`: L1 norm, her bir pikselin mutlak değeriyle yapılan normalizasyondur. Sonuçlar genellikle negatif veya küçük olur, bu yüzden çarpma işlemiyle ölçeklendirilir.

L2 Norm:

dst = np.zeros(gray.shape, dtype=np.float32)

cv.normalize(gray, dst=dst, alpha=1.0, beta=0, norm_type=cv.NORM_L2)

print(dst)

cv.imshow("NORM_L2", np.uint8(dst*1000000))

cv.waitKey(0)

`cv.NORM_L2`: L2 norm, piksellerin karelerinin toplamının karekökünü baz alarak yapılan normalizasyondur. L2 normu, verilerdeki büyük farklılıkları dengelemeye çalışır.

Ekranları Kapatma:

```
python
Kodu kopyala
cv.destroyAllWindows()
```

kod: video okuma yazma

```
import cv2 as cv
import numpy as np

capture = cv.VideoCapture("cuklu.mp4")

# Videonun yüksekliği ve genişliği, fps
height = capture.get(cv.CAP_PROP_FRAME_HEIGHT)
width = capture.get(cv.CAP_PROP_FRAME_WIDTH)
count = capture.get(cv.CAP_PROP_FRAME_COUNT)
fps = capture.get(cv.CAP_PROP_FPS)
print(height, width, count, fps)

# VideoWriter nesnesi oluşturma
out = cv.VideoWriter("cukklu.avi", cv.VideoWriter_fourcc('D', 'I', 'V', 'X'), 15, (int(width), int(height)))

while True:
    # Kameradan görüntü al
    ret, frame = capture.read()

    # Görüntü alındı mı kontrol et
    if ret:
        cv.imshow("video-input", frame)
        out.write(frame)

        # 50 milisaniye sonra çıkış yap
        c = cv.waitKey(50)
        if c == 27:  # ESC tuşu
            break
```

```
    else:
        break

capture.release()
out.release()
cv.destroyAllWindows()
#kod detayları
```

import cv2 as cv
import numpy as np

capture = cv.VideoCapture("cuklu.mp4")

- Bu satırlar, gerekli kütüphaneleri içe aktarır (`cv2` ve `numpy`). `cv.VideoCapture("cuklu.mp4")` komutu, belirtilen video dosyasını açar ve video yakalama nesnesini oluşturur.

# Videonun yüksekliği ve genişliği, fps
height = capture.get(cv.CAP_PROP_FRAME_HEIGHT)
width = capture.get(cv.CAP_PROP_FRAME_WIDTH)
count = capture.get(cv.CAP_PROP_FRAME_COUNT)
fps = capture.get(cv.CAP_PROP_FPS)
print(height, width, count, fps)
Bu satırlar, videonun özelliklerini alır: yüksekliği, genişliği, toplam kare sayısı ve saniyedeki kare sayısı (fps). Bu değerler yazdırılır.

# VideoWriter nesnesi oluşturma
out = cv.VideoWriter("cukklu.avi", cv.VideoWriter_fourcc('D', 'I', 'V', 'X'), 15, (int(width), int(height)))
Bu satır, `cv.VideoWriter` nesnesini oluşturur. Bu nesne, işlenmiş videoyu belirtilen dosya adı (`"cukklu.avi"`) ve codec (DIVX) kullanarak yazmak için kullanılır. Video 15 fps hızında ve orijinal videonun genişliği ve yüksekliğinde olacaktır.

while True:
  # Kameradan görüntü al
  ret, frame = capture.read()

  # Görüntü alındı mı kontrol et
  if ret:
    cv.imshow("video-input", frame)
    out.write(frame)

    # 50 milisaniye sonra çıkış yap
    c = cv.waitKey(50)
    if c == 27: # ESC tuşu
      break

else:

break

- `while` döngüsü, videodan kareleri okumaya devam eder.
- `ret, frame = capture.read()` satırı, bir kare okur. `ret` değişkeni, kare başarıyla okunmuşsa `True` olur, aksi takdirde `False`.
- `if ret:` kontrolü, kare başarılı bir şekilde okunmuşsa görüntünün ekranda gösterilmesini (`cv.imshow("video-input", frame)`) ve `VideoWriter` nesnesine yazılmasını (`out.write(frame)`) sağlar.
- `cv.waitKey(50)`, 50 milisaniye boyunca bekler. Eğer bu süre zarfında `ESC` tuşuna basılırsa (`c == 27`), döngü kırılır ve işlemler sonlandırılır.

capture.release()
out.release()
cv.destroyAllWindows()

- `capture.release()`, video yakalama nesnesini serbest bırakır.
- `out.release()`, `VideoWriter` nesnesini serbest bırakır.
- `cv.destroyAllWindows()`, tüm OpenCV pencerelerini kapatır

kod:

```
import cv2 as cv
import numpy as np

path=""
src=cv.imread(path+"opencv.png")

#x flip

dst1=cv.flip(src,0)
cv.imshow("x-flip",dst1)
cv.waitKey(0)

#y flip
dst2=cv.flip(src,1)
cv.imshow("y-flip",dst2)
cv.waitKey(0)
```

```
#XY
dst3=cv.flip(src,-1)
cv.imshow("xY-flip",dst3)
cv.waitKey(0)
```

GÖRÜNTÜ ÜZERİNDE GEOMETRİK ŞEKİLLER ÇİZME

kod:

```
import cv2 as cv
import numpy as np

image = np.zeros((512, 512, 3), dtype=np.uint8)

# Kare oluşturma
cv.rectangle(image, (100, 100), (300, 300), (255, 0, 0), 2, cv.LINE_8, 0)

# Daire oluşturma
cv.circle(image, (256, 256), 50, (0, 0, 255), 2, cv.LINE_8, 0)

# Elips oluşturma
cv.ellipse(image, (256, 256), (150, 50), 360, 0, 360, (0, 255, 0), 2, cv.LINE_8, 0)

for i in range(100000):
    image[:, :, :] = 0
    x1 = np.random.rand() * 512
    y1 = np.random.rand() * 512
    x2 = np.random.rand() * 512
    y2 = np.random.rand() * 512

    b = np.random.randint(0, 256)
    g = np.random.randint(0, 256)
    r = np.random.randint(0, 256)

    cv.line(image, (int(x1), int(y1)), (int(x2), int(y2)), (b, g, r), 4, cv.LINE_8, 0)
    cv.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)), (b, g, r), 4, cv.LINE_8, 0)

    cv.imshow("image", image)
    c = cv.waitKey(20)
```

```
    if c == 27:  # ESC
        break

cv.destroyAllWindows()
```

kod açıklama:

image = np.zeros((512, 512, 3), dtype=np.uint8)
Bu satır, 512x512 boyutlarında siyah bir görüntü oluşturur. Her piksel üç kanal (RGB) içerecek şekilde oluşturulur ve veri tipi 8-bit unsigned integer (`uint8`) olarak ayarlanır.

# Kare oluşturma
cv.rectangle(image, (100, 100), (300, 300), (255, 0, 0), 2, cv.LINE_8, 0)
Bu satır, belirtilen koordinatlar arasında mavi renkli (`(255, 0, 0)`) ve 2 piksel kalınlığında bir kare çizer

# Daire oluşturma
cv.circle(image, (256, 256), 50, (0, 0, 255), 2, cv.LINE_8, 0)
Bu satır, merkez koordinatı (256,256) olan, kırmızı renkli (`(0, 0, 255)`) ve 2 piksel kalınlığında bir daire çizer.

# Elips oluşturma
cv.ellipse(image, (256, 256), (150, 50), 360, 0, 360, (0, 255, 0), 2, cv.LINE_8, 0)
Bu satır, merkez koordinatı (256,256) olan, yeşil renkli (`(0, 255, 0)`) ve 2 piksel kalınlığında bir elips çizer. Elipsin büyük ekseni 150, küçük ekseni 50 piksel uzunluğundadır ve 360 derece döndürülmüştür.

```
for i in range(100000):
    image[:, :, :] = 0
    x1 = np.random.rand() * 512
    y1 = np.random.rand() * 512
    x2 = np.random.rand() * 512
    y2 = np.random.rand() * 512

    b = np.random.randint(0, 256)
    g = np.random.randint(0, 256)
    r = np.random.randint(0, 256)
```

- Bu döngü, 100.000 kez çalışacak şekilde ayarlanmıştır.
- İlk olarak, `image[:, :, :] = 0` ifadesi görüntüyü sıfırlar (tamamen siyah yapar).
- `np.random.rand()` fonksiyonu, 0 ile 1 arasında rastgele değerler üretir ve bunlar 512 ile çarpılarak 0 ile 512 arasında rastgele koordinatlar elde edilir.
-

`np.random.randint(0, 256)` fonksiyonu, 0 ile 255 arasında rastgele bir renk değeri üretir.

cv.line(image, (int(x1), int(y1)), (int(x2), int(y2)), (b, g, r), 4, cv.LINE_8, 0)

cv.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)), (b, g, r), 4, cv.LINE_8, 0)

Bu satırlar, rastgele üretilen koordinatlar ve renklerle bir çizgi ve bir kare çizer.

cv.imshow("image", image)

c = cv.waitKey(20)

if c == 27: # ESC

break

- `cv.imshow("image", image)` fonksiyonu, güncellenen görüntüyü ekranda gösterir.
- `cv.waitKey(20)` fonksiyonu, 20 milisaniye bekler. Eğer bu süre içinde ESC tuşuna basılırsa (`c == 27`), döngü kırılır ve işlem sonlanır.
- cv.destroyAllWindows()

kod:

```
import cv2 as cv

# Video okuma yazma ve işleme: kameradan görüntü
capture = cv.VideoCapture(0)  # Dahili kamera kullanılacak ise 0, harici kamera kullanılacak ise 1 yazılır

height = capture.get(cv.CAP_PROP_FRAME_HEIGHT)
width = capture.get(cv.CAP_PROP_FRAME_WIDTH)
count = capture.get(cv.CAP_PROP_FRAME_COUNT)
fps = capture.get(cv.CAP_PROP_FPS)
print(height, width, count, fps)

def process(image, opt=1):
    dst = None
    if opt == 0:
        dst = cv.bitwise_not(image)
    elif opt == 1:
        dst = cv.GaussianBlur(image, (15, 15), 0)
    elif opt == 2:
        dst = cv.Canny(image, 100, 200)
    return dst

index = 1  # İşlem seçimi için başlangıç değeri
while True:
```

```
    ret, frame = capture.read()
    if ret:
        cv.imshow("video-input", frame)
        c = cv.waitKey(50)
        if c >= 49:
            index = c - 49
        result = process(frame, index)
        cv.imshow("result", result)
        if c == 27:  # ESC tuşu
            break
    else:
        break

capture.release()
cv.destroyAllWindows()
```

açıklamaları :

Video Yakalama Nesnesi Oluşturma:

capture = cv.VideoCapture(0)
Bu satır, dahili kameradan görüntü yakalamak için `cv.VideoCapture` nesnesi oluşturur. Harici kamera kullanılacaksa 1 yazılır.

Video Özelliklerini Alma:

```
height = capture.get(cv.CAP_PROP_FRAME_HEIGHT)
width = capture.get(cv.CAP_PROP_FRAME_WIDTH)
count = capture.get(cv.CAP_PROP_FRAME_COUNT)
fps = capture.get(cv.CAP_PROP_FPS)
print(height, width, count, fps)
```

Bu satırlar, videonun yüksekliği, genişliği, toplam kare sayısı ve saniyedeki kare sayısı (fps) gibi özelliklerini alır ve yazdırır.

Görüntü İşleme Fonksiyonu:

```
def process(image, opt=1):
    dst = None
    if opt == 0:
        dst = cv.bitwise_not(image)
    elif opt == 1:
        dst = cv.GaussianBlur(image, (15, 15), 0)
    elif opt == 2:
        dst = cv.Canny(image, 100, 200)
```

```
    return dst
```

Bu fonksiyon, verilen görüntü üzerinde çeşitli işlemler gerçekleştirir:

- `opt == 0` olduğunda, bitwise NOT (ters çevirme) işlemi yapar.
- `opt == 1` olduğunda, Gaussian bulanıklaştırma uygular.
- `opt == 2` olduğunda, Canny kenar tespiti uygular.

Ana Döngü:

```
index = 1  # İşlem seçimi için başlangıç değeri
while True:
    ret, frame = capture.read()
    if ret:
        cv.imshow("video-input", frame)
        c = cv.waitKey(50)
        if c >= 49:
            index = c - 49
        result = process(frame, index)
        cv.imshow("result", result)
        if c == 27:  # ESC tuşu
            break
    else:
        break
```

Bu döngü, sürekli olarak kameradan görüntü alır ve işlenmiş görüntüyü gösterir:

- `ret` değeri `True` olduğunda, görüntü başarıyla alınmış demektir.
- `cv.imshow("video-input", frame)` ile ham görüntüyü gösterir.
- `cv.waitKey(50)` ile 50 milisaniye bekler ve tuş girişini okur.
- `if c >= 49:` ile tuş girişine göre işlem seçimini günceller (`1` tuşu Gaussian bulanıklaştırma, `2` tuşu Canny kenar tespiti yapar).
- İşlenmiş görüntüyü `cv.imshow("result", result)` ile gösterir.
- `ESC` tuşuna basıldığında döngüden çıkar.

Kapatma İşlemleri:

```
capture.release()
cv.destroyAllWindows()
```

kod:

```
import cv2 as cv

# Resmi yükleyin
img = cv.imread("opencv.png")
cv.namedWindow("rgb", cv.WINDOW_AUTOSIZE)
cv.imshow("rgb", img)
cv.waitKey(1)

# RGB'den gri tonlamalıya dönüştürme
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imshow("gray", gray)
cv.waitKey(1)



HSV=cv.cvtColor(img,cv.COLOR_BGR2HSV)
cv.imshow("HSV", HSV)
cv.waitKey(0)
cv.destroyAllWindows()
```

GEOMETRİK DÖNÜŞÜMLER

```
import cv2 as cv
import numpy as np



#kaydırma yöntemi #shifting

img=cv.imread("opencv.png")
rows=img.shape[0]#satır
cols=img.shape[1]#sutun

m = np.float32([[1, 0, 300], [0, 1, 90]])

shifted=cv.warpAffine(img,m,(cols,rows))

cv.imshow('orginal',img)
cv.waitKey(0)

cv.imshow('shifed',shifted)
cv.waitKey(1)
```

```
#rotation döndürme
m=cv.getRotationMatrix2D((cols/2,rows/2),90,1)
dst=cv.warpAffine(img,m,(cols,rows))
cv.imshow('img',dst)
cv.waitKey(1)

#scaling
res=cv.resize(img,None,fx=0.4,fy=0.2,interpolation=cv.INTER_CUBIC)
cv.imshow('img',res)
cv.waitKey(0)
```

Kaydırma (Shifting)

#kaydırma yöntemi #shifting

```
img=cv.imread("opencv.png")
rows=img.shape[0] #satır
cols=img.shape[1] #sutun

m = np.float32([[1, 0, 300], [0, 1, 90]])
shifted=cv.warpAffine(img,m,(cols,rows))

cv.imshow('orginal',img)
cv.waitKey(0)

cv.imshow('shifed',shifted)
cv.waitKey(1)
```

- **Resmi Yükleme:** `img=cv.imread("opencv.png")`
  - Bu satır, belirtilen dosya yolundaki resmi yükler.
- **Resmin Boyutlarını Alma:** `rows=img.shape[0]` ve `cols=img.shape[1]`
  - `rows`, resmin satır (yükseklik) sayısını, `cols` ise sütun (genişlik) sayısını tutar.
- **Kaydırma Matrisi:** `m = np.float32([[1, 0, 300], [0, 1, 90]])`
  - Bu satır, resmin 300 piksel sağa ve 90 piksel aşağı kaydırılmasını belirten kaydırma matrisini tanımlar.
- **Kaydırılmış Resim:** `shifted=cv.warpAffine(img,m,(cols,rows))`

- Bu satır, orijinal resmi `m` matrisi ile kaydırır ve yeni resmi `shifted` değişkenine atar.

- **Resimleri Gösterme:** `cv.imshow('orginal',img)` ve `cv.imshow('shifed',shifted)`

  - Bu satırlar, orijinal ve kaydırılmış resmi gösterir. `cv.waitKey(0)` ve `cv.waitKey(1)` satırları, tuş girişi beklemek için kullanılır.

Döndürme (Rotation)

#rotation döndürme
m=cv.getRotationMatrix2D((cols/2,rows/2),90,1)
dst=cv.warpAffine(img,m,(cols,rows))
cv.imshow('img',dst)
cv.waitKey(1)

- **Döndürme Matrisi:** `m=cv.getRotationMatrix2D((cols/2,rows/2),90,1)`

  - Bu satır, resmin merkez noktası etrafında (cols/2, rows/2) 90 derece döndürülmesini belirten dönüş matrisini oluşturur. `1` değeri, dönüş sırasında ölçeklendirmeyi belirtir (burada 1 ile ölçeklendirme yapılmaz).

- **Döndürülmüş Resim:** `dst=cv.warpAffine(img,m,(cols,rows))`

  - Bu satır, orijinal resmi `m` matrisi ile döndürür ve yeni resmi `dst` değişkenine atar.

- **Döndürülmüş Resmi Gösterme:** `cv.imshow('img',dst)`

  - Bu satır, döndürülmüş resmi gösterir. `cv.waitKey(1)` tuş girişi beklemek için kullanılır.

Ölçeklendirme (Scaling)
#scaling
res=cv.resize(img,None,fx=0.4,fy=0.2,interpolation=cv.INTER_CUBIC)
cv.imshow('img',res)
cv.waitKey(0)

- **Ölçeklendirilmiş Resim:** `res=cv.resize(img,None,fx=0.4,fy=0.2,interpolation=cv.INTER_CUBIC)`

  - Bu satır, orijinal resmi x ekseninde `0.4` ve y ekseninde `0.2` faktörleriyle ölçeklendirir. `cv.INTER_CUBIC` değeri, kübik interpolasyon kullanarak yeniden boyutlandırma işlemini gerçekleştirir.

- **Ölçeklendirilmiş Resmi Gösterme:** `cv.imshow('img',res)`

- Bu satır, ölçeklendirilmiş resmi gösterir. `cv.waitKey(0)` tuş girişi beklemek için kullanılır.

kod :

```
import cv2
import cv2 as cv

# Görüntüyü oku
src = cv.imread("me.jpg")

# Görüntünün boyutlarını al
h, w = src.shape[:2]

# Görüntü kopyasını al
img = src.copy()

# Geçerli bir dilimleme kullanarak roi (region of interest) seç
roi = img[300:750, 950:1300, :]  # float yerine tam sayı kullanıyoruz

# ROI boyutlarını yazdır
print("ROI Shape:", roi.shape[:2])

# ROI'yi hedef boyutlara (450, 350) uyacak şekilde yeniden boyutlandır
roi_resized = cv.resize(roi, (350, 450))  # Hedef boyut (350, 450)

# ROI'yi img içine yerleştir
img[0:450, 0:350, :] = roi_resized
cv.imshow("Image with ROI", img)
cv.waitKey(0)

# ROI'yi yeniden boyutlandır (zoom out)
res = cv.resize(roi_resized, None, fx=0.3, fy=0.3, interpolation=cv.INTER_CUBIC)

# Yeniden boyutlandırılmış görüntüyü göster
cv.imshow("Resized", res)
cv.waitKey(0)

# Yeniden boyutlandırılmış resmi img üzerine yerleştir
resized_h, resized_w = res.shape[:2]
img[0:resized_h, 0:resized_w, :] = res
cv.imshow("Image with Resized ROI", img)
cv.waitKey(0)
```

```
# Kaynak görüntüye yeniden boyutlandırılmış resmi yerleştir
src[0:resized_h, 0:resized_w, :] = res
cv.imshow("Source Image with Resized ROI", src)
cv.waitKey(0)

cv.destroyAllWindows()
```

histrogram oluşturmada kaldım

kod:

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

def costum_hist(gray):
    h, w = gray.shape
    hist = np.zeros([256], dtype=np.int32)
    for row in range(h):
        for col in range(w):
            pv = gray[row, col]
            hist[pv] += 1

    y_pos = np.arange(0, 256, 1, dtype=np.int32)
    plt.bar(y_pos, hist, align='center', color='r', alpha=0.5)
    plt.xticks(y_pos, y_pos)
    plt.ylabel('Frequency')
    plt.title('Histogram')
    plt.show()

def image_hist(image):
    cv.imshow("input", image)
    colors = ('blue', 'green', 'red')
    for i, color in enumerate(colors):
        hist = cv.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(hist, color=color)
        plt.xlim([0, 256])
    plt.show()

src = cv.imread("me.jpg")
if src is None:
```

```
    print("Resim yüklenemedi. Dosya yolunu ve dosya adını kontrol edin.")
else:
    cv.namedWindow("input", cv.WINDOW_AUTOSIZE)
    cv.imshow("INPUT", src)
    cv.waitKey(0)


gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)


costum_hist(gray)
image_hist(src)
```

açıklaması:

`costum_hist` Fonksiyonu (Özel Histogram Fonksiyonu)

```
def costum_hist(gray):
    h, w = gray.shape
    hist = np.zeros([256], dtype=np.int32)
    for row in range(h):
        for col in range(w):
            pv = gray[row, col]
            hist[pv] += 1


    y_pos = np.arange(0, 256, 1, dtype=np.int32)
    plt.bar(y_pos, hist, align='center', color='r', alpha=0.5)
    plt.xticks(y_pos, y_pos)
    plt.ylabel('Frequency')
    plt.title('Histogram')
    plt.show()
```

- **`gray.shape`**: Gri tonlamalı görüntünün (gray) boyutlarını alır. `h` (yükseklik) ve `w` (genişlik) değerlerini verir.
- **`hist = np.zeros([256], dtype=np.int32)`**: 0'dan 255'e kadar olan gri tonlarındaki piksellerin frekansını sayacak bir histogram dizisi oluşturur. Histogram, her piksel değeri için (0–255 arasındaki her bir gri ton) bir sayacın tutulduğu bir dizidir.
- **`for row in range(h)` ve `for col in range(w)`**: Görüntüyü her bir pikseliyle gezip her pikselin değerini (`pv`) alır.
- **`hist[pv] += 1`**: Pikselin gri tonunu kullanarak, histogramdaki karşılık gelen değeri artırır.
- **`y_pos = np.arange(0, 256, 1, dtype=np.int32)`**: X eksenindeki değerleri (0–255) tanımlar.
- **`plt.bar(y_pos, hist, align='center', color='r', alpha=0.5)`**: `y_pos` ile histogram verilerini çizmek için bir çubuk grafik (bar chart) oluşturur. Çubukların rengi kırmızı (`'r'`) ve saydamlık değeri %50 (`alpha=0.5`) olarak ayarlanmıştır.
- **`plt.xticks(y_pos, y_pos)`**: X eksenindeki etiketleri (0–255 arası) doğru şekilde yerleştirir.
- **`plt.ylabel('Frequency')` ve `plt.title('Histogram')`**: Y eksenine "Frekans" etiketi ekler ve grafiğe başlık ekler.
- **`plt.show()`**: Grafiği ekranda görüntüler.

Bu fonksiyon, bir görüntüdeki gri tonlarının dağılımını (frekansını) görselleştirir.

`image_hist` Fonksiyonu (Renkli Histogram Fonksiyonu)

```
def image_hist(image):
    cv.imshow("input", image)
    colors = ('blue', 'green', 'red')
    for i, color in enumerate(colors):
        hist = cv.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(hist, color=color)
        plt.xlim([0, 256])
    plt.show()
```

- **`cv.imshow("input", image)`**: Görüntüyü ekranda görüntüler.
- **`colors = ('blue', 'green', 'red')`**: Renk kanallarının sırasıyla mavi, yeşil ve kırmızı renklerini belirtir.
- **`for i, color in enumerate(colors)`**: Renk kanallarını sırayla gezmek için bir döngü başlatır. `i` kanalın indeksini, `color` ise kanalın rengini temsil eder.
- **`hist = cv.calcHist([image], [i], None, [256], [0, 256])`**: Her bir renk kanalı için histogram hesaplar. Bu, `image` adlı renkli görüntüde, `i`-inci (0 için mavi, 1 için yeşil, 2 için kırmızı) kanalın değerlerinin histogramını hesaplar.
- **`plt.plot(hist, color=color)`**: Her kanal için histogramı çizer. Renk, kanalın rengini belirtir.
- **`plt.xlim([0, 256])`**: X eksenini 0 ile 256 arasında sınırlar (gri tonları veya renk kanalı değerleri 0–255 arasında olduğu için).
- **`plt.show()`**: Grafiklerin görüntülenmesini sağlar.

Bu fonksiyon, renkli bir görüntüde her bir renk kanalının (mavi, yeşil, kırmızı) histogramlarını çizer.

```
src = cv.imread("me.jpg")
if src is None:
    print("Resim yüklenemedi. Dosya yolunu ve dosya adını kontrol edin.")
else:
    cv.namedWindow("input", cv.WINDOW_AUTOSIZE)
    cv.imshow("INPUT", src)
    cv.waitKey(0)
```

- **`src = cv.imread("me.jpg")`**: `me.jpg` adlı resmi yükler. Eğer resim bulunamazsa `None` döner.
- **`if src is None:`**: Eğer resim yüklenememişse, hata mesajı verir.
- **`cv.imshow("INPUT", src)`**: Görüntüyü ekranda görüntüler.
- **`cv.waitKey(0)`**: Görüntü ekranında herhangi bir tuşa basılana kadar bekler.

Gri Tonlamalı Görüntü ve Histogram Çizme

```
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)


costum_hist(gray)
image_hist(src)
```

- **`gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)`**: Renkli görüntüyü gri tonlamalıya dönüştürür.
- **`costum_hist(gray)`**: Gri tonlamalı görüntü için özel histogram fonksiyonunu çağırır.
- **`image_hist(src)`**: Renkli görüntü için renkli histogram fonksiyonunu çağırır.

kod:

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


def costum_hist(gray):
    h, w = gray.shape
    hist = np.zeros([256], dtype=np.int32)
    for row in range(h):
        for col in range(w):
            pv = gray[row, col]
            hist[pv] += 1


    y_pos = np.arange(0, 256, 1, dtype=np.int32)
    plt.bar(y_pos, hist, align='center', color='r', alpha=0.5)
    plt.xticks(y_pos, y_pos)
    plt.ylabel('Frequency')
    plt.title('Histogram')
    plt.show()

src = cv.imread("lab.png")
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
cv.imshow("input", gray)
cv.waitKey(0)
costum_hist(gray)

dst = cv.equalizeHist(gray)
cv.imshow("eh", dst)
cv.waitKey(0)

costum_hist(dst)
cv.destroyAllWindows()
```

açıklaması:

Histogram Gösterme:

cv.imshow("input", gray)

cv.waitKey(0)

costum_hist(gray)

Resmi gösterir ve histogramını hesaplayarak gösterir.

Histogram Eşitleme:

dst = cv.equalizeHist(gray)

cv.imshow("eh", dst)

cv.waitKey(0)

costum_hist(dst)

Histogram eşitlemesi uygular, eşitlenmiş resmi gösterir ve histogramını hesaplayarak gösterir.

```
Histogram Comparison (src1 vs src2): 0.10479444206963749
Histogram Comparison (src1 vs src3): 0.9201827552343452
Histogram Comparison (src2 vs src3): 0.08643255885001302
```

kodu:

```
import cv2 as cv

# Resimleri yükleme
src1 = cv.imread("opencv.png")
src2 = cv.imread("lab.png")
src3 = cv.imread("sag.png")

# BGR'den HSV'ye dönüştürme
hsv1 = cv.cvtColor(src1, cv.COLOR_BGR2HSV)
hsv2 = cv.cvtColor(src2, cv.COLOR_BGR2HSV)
hsv3 = cv.cvtColor(src3, cv.COLOR_BGR2HSV)

# Histogram hesaplama
hist1 = cv.calcHist([hsv1], [0, 1], None, [60, 64], [0, 180, 0, 256])
hist2 = cv.calcHist([hsv2], [0, 1], None, [60, 64], [0, 180, 0, 256])
hist3 = cv.calcHist([hsv3], [0, 1], None, [60, 64], [0, 180, 0, 256])

# Histogram normalize etme
cv.normalize(hist1, hist1, 0, 1.0, cv.NORM_MINMAX)
cv.normalize(hist2, hist2, 0, 1.0, cv.NORM_MINMAX)
cv.normalize(hist3, hist3, 0, 1.0, cv.NORM_MINMAX)

# Histogram karşılaştırma
result1_2 = cv.compareHist(hist1, hist2, cv.HISTCMP_CORREL)
result1_3 = cv.compareHist(hist1, hist3, cv.HISTCMP_CORREL)
result2_3 = cv.compareHist(hist2, hist3, cv.HISTCMP_CORREL)

# Sonuçları yazdırma
print("Histogram Comparison (src1 vs src2):", result1_2)
print("Histogram Comparison (src1 vs src3):", result1_3)
print("Histogram Comparison (src2 vs src3):", result2_3)
```

```
import cv2 as cv

# Resimleri yükleme
src1 = cv.imread("opencv.png")
src2 = cv.imread("lab.png")
src3 = cv.imread("sag.png")

# BGR'den HSV'ye dönüştürme
hsv1 = cv.cvtColor(src1, cv.COLOR_BGR2HSV)
hsv2 = cv.cvtColor(src2, cv.COLOR_BGR2HSV)
hsv3 = cv.cvtColor(src3, cv.COLOR_BGR2HSV)

# Histogram hesaplama
hist1 = cv.calcHist([hsv1], [0, 1], None, [60, 64], [0, 180, 0, 256])
hist2 = cv.calcHist([hsv2], [0, 1], None, [60, 64], [0, 180, 0, 256])
hist3 = cv.calcHist([hsv3], [0, 1], None, [60, 64], [0, 180, 0, 256])

# Histogram normalize etme
cv.normalize(hist1, hist1, 0, 1.0, cv.NORM_MINMAX)
cv.normalize(hist2, hist2, 0, 1.0, cv.NORM_MINMAX)
cv.normalize(hist3, hist3, 0, 1.0, cv.NORM_MINMAX)

# Histogram karşılaştırma
result1_2 = cv.compareHist(hist1, hist2, cv.HISTCMP_CORREL)
result1_3 = cv.compareHist(hist1, hist3, cv.HISTCMP_CORREL)
result2_3 = cv.compareHist(hist2, hist3, cv.HISTCMP_CORREL)

# Sonuçları yazdırma
print("Histogram Comparison (src1 vs src2):", result1_2)
print("Histogram Comparison (src1 vs src3):", result1_3)
print("Histogram Comparison (src2 vs src3):", result2_3)
```

KOD:

```
import cv2 as cv
import numpy as np

# Resmi okuma
src = cv.imread("agla.jpg")

# Pencere oluşturma ve resmi gösterme
cv.namedWindow("input", cv.WINDOW_AUTOSIZE)
```

```
cv.imshow("input", src)
cv.waitKey(1)

# Görüntü boyutları
h, w = src.shape[:2]

# Bilateral filtre uygulama
dst = cv.bilateralFilter(src, 0, 5, 10)

# Sonuç için yeni boş bir görüntü oluşturma
result = np.zeros([h, w*2, 3], dtype=src.dtype)

# Orijinal resmi ve filtrelenmiş resmi yerleştirme
result[0:h, 0:w, :] = src
result[0:h, w:2*w, :] = dst

# Sonuç görüntüsünü gösterme
cv.imshow("result", result)
cv.waitKey(0)
cv.destroyAllWindows()
```

açıklaması:

- **Resmin Okunması**:

  ```
  python
  Kodu kopyala
  src = cv.imread("agla.jpg")
  ```

  Bu satır, `agla.jpg` adlı resmi okur ve `src` değişkenine atar.

- **Pencere Oluşturma ve Resmi Gösterme**:

  ```
  python
  Kodu kopyala
  cv.namedWindow("input", cv.WINDOW_AUTOSIZE)cv.imshow("input", src)cv.waitKey(1)
  ```

  Bu satırlarla, bir pencere oluşturulur (pencere boyutunun otomatik ayarlandığı) ve orijinal resim gösterilir. `cv.waitKey(1)` resmin görünmesini sağlar.

- **Görüntü Boyutlarının Alınması**:

  ```
  python
  Kodu kopyala
  h, w = src.shape[:2]
  ```

Burada, resmin yüksekliği (`h`) ve genişliği (`w`) alınır. Bu boyutlar, daha sonra sonucu görüntülemek için kullanılacaktır.

- **Bilateral Filtre Uygulaması**:

```
python
Kodu kopyala
dst = cv.bilateralFilter(src, 0, 5, 10)
```

Bu satır, `src` resmine bilateral filtre uygular. Bilateral filtre, görüntüdeki kenarları koruyarak gürültüyü azaltan bir tekniktir. Filtre parametreleri:

- `0`: Filtre çekirdek boyutu otomatik belirlenir.
- `5`: Renk uzayındaki yakınlık (daha düşük değer, daha fazla yakınlık).
- `10`: Uzaysal yakınlık (piksel arasındaki mesafe).

- **Yeni Bir Boş Görüntü Oluşturma**:

```
python
Kodu kopyala
result = np.zeros([h, w*2, 3], dtype=src.dtype)
```

Burada, orijinal ve filtrelenmiş resimleri yan yana gösterecek bir boş `result` görüntüsü oluşturuluyor. Bu görüntünün boyutları, orijinal resmin yüksekliği kadar (h) ve genişliğinin iki katı kadar (w*2) olacak şekilde belirlenmiştir.

- **Orijinal ve Filtrelenmiş Resimleri Yerleştirme**:

```
python
Kodu kopyala
result[0:h, 0:w, :] = srcresult[0:h, w:2*w, :] = dst
```

Bu adımda, orijinal görüntü (`src`) sol tarafa, filtrelenmiş görüntü (`dst`) ise sağ tarafa yerleştirilir. Sonuçta `result` görüntüsünde orijinal ve filtrelenmiş resim yan yana gösterilmiş olur.

- **Sonuç Görüntüsünü Gösterme**:

```
python
Kodu kopyala
cv.imshow("result", result)cv.waitKey(0)cv.destroyAllWindows()
```

Son olarak, `result` görüntüsü yeni bir pencere içerisinde gösterilir. `cv.waitKey(0)` ile kullanıcı bir tuşa basana kadar pencere açık kalır, ardından `cv.destroyAllWindows()` ile tüm pencereler kapatılır.

kod:

```
#kenarları koruyarak filtreleme işlemi

import cv2 as cv
import numpy as np

src=cv.imread("agla.jpg")
cv.imshow("input",src)
cv.waitKey(0)
h, w = src.shape[:2]

dst=cv.edgePreservingFilter(src,sigma_s=90,sigma_r=0.4,flags=cv.RECURS_FILTER)
#sigma s 0-200 arası sigma r =0 ile 1 arası değer alır
result = np.zeros([h, w*2, 3], dtype=src.dtype)



result[0:h, 0:w, :] = src
result[0:h, w:2*w, :] = dst

result=cv.resize(result,(w,h//2))
cv.imshow("result",result)
cv.waitKey(0)
```

kod:

```
import cv2 as cv
import numpy as np

# Resmi yükle
src = cv.imread("opencv.png")

# Görüntü boyutlarını al
h, w = src.shape[:2]

# X ve Y yönlerinde Sobel filtreleri uygula
x_grad = cv.Sobel(src, cv.CV_32F, 1, 0)  # X yönündeki gradyan
y_grad = cv.Sobel(src, cv.CV_32F, 0, 1)  # Y yönündeki gradyan

# Gradyanları mutlak değere dönüştür
x_grad = cv.convertScaleAbs(x_grad)
y_grad = cv.convertScaleAbs(y_grad)
```

```
# X ve Y gradyanlarının toplamını al (birleştir)
dst = cv.add(x_grad, y_grad)

# Üç resmin her birini aynı anda göster
cv.imshow("Original Image", src)
cv.imshow("X Gradient", x_grad)
cv.imshow("Y Gradient", y_grad)
cv.imshow("Gradient Magnitude", dst)

# Görüntüler açıkken bekle
cv.waitKey(0)

# Ekranı kapat
cv.destroyAllWindows()
```

kod:

```
import cv2 as cv
import numpy as np

src=cv.imread("kenau.jpeg")
cv.imshow("kenau",src)
cv.waitKey(300)

edge=cv.Canny(src,100,300)
cv.imshow("mask_image",edge)
cv.waitKey(0)
```

kod:

```
import cv2 as cv
import numpy as np

# Görüntüyü yükle
src = cv.imread("hadise.png")
cv.namedWindow("input", cv.WINDOW_AUTOSIZE)
cv.imshow("input", src)
cv.waitKey(0)

# Görüntüyü gri tonlamaya dönüştür
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
```

```
# Otsu yöntemiyle ikili (binary) dönüşüm
ret, binary = cv.threshold(gray, 0, 255, cv.THRESH_BINARY | cv.THRESH_OTSU)

# Sonuçları göster
cv.imshow("binary", binary)
cv.waitKey(0)

cv.destroyAllWindows()
```

kod:

```
# IMAGE CONTOURS (GÖRÜNTÜ KONTURLARI)
#####################################

import cv2 as cv

def threshold_demo(image):
    dst = cv.GaussianBlur(image, (3, 3), 0)
    gray = cv.cvtColor(dst, cv.COLOR_BGR2GRAY)
    ret, binary = cv.threshold(gray, 0, 255, cv.THRESH_OTSU | cv.THRESH_BINARY)
    cv.imshow("binary", binary)
    return binary

def canny_demo(image):
    t=100
    canny_output=cv.Canny(image,t,t*2)
    cv.imshow("canny_output",canny_output)
    return canny_output



src=cv.imread("hadise.png")
cv.namedWindow("input",cv.WINDOW_AUTOSIZE)
cv.imshow("input",src)
cv.waitKey(0)



binary= threshold_demo(src)

canny=canny_demo(src)

contours,hierarchy=cv.findContours(canny,cv.RETR_TREE,cv.CHAIN_APPROX_SIMPLE)

for c in range(len(contours)):
```

```
    cv.drawContours(src,contours,c,(0,0,255),2,8)


cv.imshow("contours-demo",src)
cv.waitKey(0)
```

açıklaması:

Kodun Genel Yapısı

1.
    - Giriş görüntüsünü bulanıklaştırır.
    - Gri tonlamaya dönüştürür.
    - İkili görüntü (binary) oluşturmak için Otsu eşikleme algoritmasını uygular.
    - Çıktı: Kenar algılama için işlenmiş ikili görüntü.
2. **`canny_demo(image)`**:
    - Canny kenar algılama algoritmasını uygular.
    - Algılanan kenarları içeren bir görüntü döndürür.
3. **Kontur Bulma**:
    - `cv.findContours()`:
        - Canny çıktısından konturları bulur.
    - `cv.drawContours()`:
        - Tespit edilen konturları orijinal görüntü üzerine çizer.

Giriş Görüntüsünü Yükleme:

```
src = cv.imread("hadise.png")cv.imshow("input", src)

hadise.png dosyasını okur ve ekranda "input" başlıklı bir pencerede gösterir.


Eşikleme İşlemi (threshold_demo):
binary = threshold_demo(src)
Giriş görüntüsü üzerinde bir Gaussian bulanıklaştırma uygulanır.
Gri tonlamaya dönüştürülür.
Otsu eşikleme yöntemi ile ikili (siyah-beyaz) bir görüntü elde edilir.


Canny Kenar Algılama (canny_demo):

canny = canny_demo(src)
Canny kenar algılama uygulanır. Kenarları algılanmış bir çıktı döner.


Kontur Tespiti:
contours, hierarchy = cv.findContours(canny, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
cv.findContours ile görüntüdeki kenarlardan konturlar çıkarılır.
```

- **`cv.RETR_TREE`**: Konturları ve kontur hiyerarşisini (alt ve üst ilişkiler) algılar.
- **`cv.CHAIN_APPROX_SIMPLE`**: Kontur noktalarını basitleştirir.

Kontur Çizimi:

```
for c in range(len(contours)):
```

cv.drawContours(src, contours, c, (0, 0, 255), 2, 8)

Her bir tespit edilen kontur, orijinal görüntü üzerine çizilir.

- `(0, 0, 255)`: Konturların kırmızı renkte çizilmesini sağlar.
- `2`: Çizgi kalınlığı.

Çıkış Görüntüsü:

cv.imshow("contours-demo", src)

cv.waitKey(0)

kod:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

def canny_demo(image):
    t = 100
    canny_output = cv.Canny(image, t, t * 2)
    cv.imshow("canny_output", canny_output)
    return canny_output

src = cv.imread("sudoku.jpg")
if src is None:
    print("Görüntü yüklenemedi!")
    exit()

cv.namedWindow("input", cv.WINDOW_AUTOSIZE)
cv.imshow("input", src)
cv.waitKey(500)

binary = canny_demo(src)
cv.imshow("binary", binary)
cv.waitKey(5000)

lines = cv.HoughLines(binary, 1, np.pi / 180, 150, None, 0, 0)
if lines is not None:
    for i in range(0, len(lines)):
```

```
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        pt1 = (int(x0 + 1000 * (-b)), int(y0 + 1000 * (a)))
        pt2 = (int(x0 - 1000 * (-b)), int(y0 - 1000 * (a)))
        cv.line(src, pt1, pt2, (0, 0, 255), 3, cv.LINE_8)

cv.imshow("contours-demo", src)
cv.waitKey(0)
cv.destroyAllWindows()
```

açıklama:

Görüntü Okuma ve Kontrol

```
src = cv.imread("sudoku.jpg")
if src is None:
    print("Görüntü yüklenemedi!")
    exit()
```

- **Amaç:** `sudoku.jpg` adlı bir görüntüyü yükler. Eğer görüntü bulunamazsa programdan çıkar.

Görüntü Gösterimi

```
cv.namedWindow("input", cv.WINDOW_AUTOSIZE)
cv.imshow("input", src)
cv.waitKey(500)
```

Amaç: Yüklenen görüntüyü bir pencere içinde gösterir. `cv.waitKey(500)` ile 500 ms bekler.

Canny Kenar Algılama

```
def canny_demo(image):
    t = 100
    canny_output = cv.Canny(image, t, t * 2)
    cv.imshow("canny_output", canny_output)
    return canny_output

binary = canny_demo(src)
cv.imshow("binary", binary)
cv.waitKey(5000)
```

-

Amaç: Görüntüdeki kenarları belirlemek için Canny kenar algılama uygulanır:

  - `t = 100`: Alt eşik değeri.
  - `t * 2 = 200`: Üst eşik değeri.

- Kenar algılama sonucunu (`canny_output`) döndürür ve görüntüler.

Hough Doğru Tespiti

lines = cv.HoughLines(binary, 1, np.pi / 180, 150, None, 0, 0)

- **Amaç:** `cv.HoughLines` fonksiyonuyla tespit edilen kenarlar üzerinden doğrular bulunur.
  - `binary`: Canny kenar algılama sonucu (ikili görüntü).
  - `1`: `rho` çözünürlüğü (1 piksel).
  - `np.pi / 180`: `theta` çözünürlüğü (1 derece).
  - `150`: Hough dönüşümü için oy eşik değeri (150 oy gereklidir).

Eğer doğrular tespit edilirse, dönen sonuç bir `lines` dizisi olur.

Doğru Çizimi

```
if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        pt1 = (int(x0 + 1000 * (-b)), int(y0 + 1000 * (a)))
        pt2 = (int(x0 - 1000 * (-b)), int(y0 - 1000 * (a)))
        cv.line(src, pt1, pt2, (0, 0, 255), 3, cv.LINE_8)
```

  - Eğer doğrular tespit edildiyse:
    - Her bir doğruyu tanımlayan `(rho, theta)` değerleri alınır.
    - `(rho, theta)` koordinatlarını kullanarak doğruların uç noktaları `(pt1, pt2)` hesaplanır.
    - `cv.line` ile bu doğrular çizilir (kırmızı renkte `(0, 0, 255)` ve 3 piksel kalınlıkta).

Sonuç Görüntüleme

- Çizilen doğruların yer aldığı görüntü yeni bir pencere içinde gösterilir.
- Kullanıcı bir tuşa basana kadar beklenir, ardından tüm pencereler kapatılır.

kod:

```
import cv2 as cv
```

```
import numpy as np

def canny_demo(image):
    t = 100
    canny_output = cv.Canny(image, t, t * 2)
    cv.imshow("Canny Output", canny_output)
    return canny_output

# Görüntüyü yükle
src = cv.imread("suboku.jpg")
if src is None:
    print("Görüntü yüklenemedi. Lütfen dosya yolunu kontrol edin.")
    exit()

# Orijinal görüntüyü göster
cv.imshow("Input", src)

# Canny kenar algılama
binary = canny_demo(src)

# Probabilistik Hough Dönüşümü ile çizgileri tespit et
linesP = cv.HoughLinesP(binary, 1, np.pi / 180, 55, None, 50, 10)

# Çizgileri çiz
if linesP is not None:
    for i in range(len(linesP)):
        l = linesP[i][0]
        cv.line(src, (l[0], l[1]), (l[2], l[3]), (255, 0, 0), 1, cv.LINE_AA)

# Sonuç görüntüsünü göster
cv.imshow("Hough Line Demo", src)

# Bekleme ve pencereleri kapatma
cv.waitKey(0)
cv.destroyAllWindows()
```

kod:

```
import cv2 as cv
import numpy as np

src = cv.imread("demirpara.jpg")
gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
```

```
gray = cv.GaussianBlur(gray, (9, 9), 2, 2)
circles = cv.HoughCircles(gray, cv.HOUGH_GRADIENT, dp=1, minDist=10, param1=100, param2=50, minRadius=20, maxRadius=100)

if circles is not None:
    circles = np.round(circles[0, :]).astype("int")
    for c in circles:
        cx, cy, r = c
        cv.circle(src, (cx, cy), 2, (0, 255, 0), 3)
        cv.circle(src, (cx, cy), r, (0, 255, 0), 3)
    cv.imshow("Hough Circle Detection", src)
    cv.waitKey(0)
else:
    print("No circles were detected")

cv.destroyAllWindows()
```

açıklması:

Görüntüyü yükleme:

src = cv.imread("demirpara.jpg")

Bu satırda `demirpara.jpg` dosyasındaki görüntü `src` adlı bir değişkene yüklenir. `cv.imread()` fonksiyonu ile görüntü dosyası okunur.

Görüntüyü gri tonlara dönüştürme:

gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)

Renkli görüntü `src`, gri tonlamalı bir görüntüye dönüştürülür. Bu işlem, daire tespiti için daha basit bir veri kümesi oluşturur ve genellikle işlemeyi hızlandırır.

Gaussian Blur (Gauss bulanıklaştırması):

gray = cv.GaussianBlur(gray, (9, 9), 2, 2)

Bu satırda görüntüye Gaussian bulanıklaştırma uygulanır. Bu işlem, görüntüdeki gürültüyü azaltır ve daire tespiti için daha düzgün bir yüzey oluşturur. `(9, 9)` bulanıklık penceresinin boyutudur, diğer iki değer ise bulanıklaştırma için sigma değerleridir.

Daireleri tespit etme (Hough Daire Dönüşümü):

circles = cv.HoughCircles(gray, cv.HOUGH_GRADIENT, dp=1, minDist=10, param1=100, param2=50, minRadius=20, maxRadius=100)

Bu satırda, Hough Daire Dönüşümü kullanılarak daireler tespit edilir. `cv.HoughCircles()` fonksiyonu şu parametreleri kullanır:

- `gray`: Gri tonlamalı görüntü.
- `cv.HOUGH_GRADIENT`: Daire tespiti için kullanılan yöntem (Hough Dairesi Yöntemi).
- `dp=1`: Dairesel parametrelerin çözünürlüğünü belirler. Bu, görüntünün boyutuyla orantılıdır.

- `minDist=10`: Dairelerin merkezleri arasındaki minimum mesafe. Bu, birbirine yakın dairelerin ayrılmasını sağlar.
- `param1=100` ve `param2=50`: Canny kenar tespiti için sınır değerleridir. Bu, daireleri doğru şekilde tespit etmek için kullanılır.
- `minRadius=20` ve `maxRadius=100`: Dairelerin minimum ve maksimum yarıçaplarıdır.

Dairelerin çizilmesi:

if circles is not None:

circles = np.round(circles[0, :]).astype("int")

for c in circles:

cx, cy, r = c

cv.circle(src, (cx, cy), 2, (0, 255, 0), 3)

cv.circle(src, (cx, cy), r, (0, 255, 0), 3)

- `np.round(circles[0, :]).astype("int")`: Dairelerin koordinatlarını yuvarlar ve tam sayılara dönüştürür.
- `cv.circle(src, (cx, cy), 2, (0, 255, 0), 3)`: Dairelerin merkezini yeşil renkte bir nokta ile işaretler.
- `cv.circle(src, (cx, cy), r, (0, 255, 0), 3)`: Dairenin çevresini yeşil renkte bir çizgiyle çizer.
-
- sonucu gösterme
- cv.imshow("Hough Circle Detection", src)

cv.waitKey(0)

video arkaplanda kaldırma:

```
import cv2 as cv
cap=cv.VideoCapture('cuklu.mp4')
fgbg=cv.createBackgroundSubtractorMOG2(history=250,varThreshold=250)


while True:
    ret,frame=cap.read()
    fgmask=fgbg.apply(frame)
    background=fgbg.getBackgroundImage()
    cv.imshow('input',frame)
    cv.imshow('input', fgmask)
    cv.imshow('background',background)
    k=cv.waitKey(10)&0xff
    if k ==27:
        break


cap.release()


cv.destroyAllWindows()
```

kodun açıklaması:

Video Kaynağını Açma

```
cap = cv.VideoCapture('cuklu.mp4')
```

Arka Plan Çıkarıcıyı Başlatma

```
fgbg = cv.createBackgroundSubtractorMOG2(history=250, varThreshold=250)
```

`history=250`: Arka plan modelinin "geçmiş" boyutunu belirler. Bu, arka planın ne kadar eski bilgilere dayanarak güncelleneceğini kontrol eder.

`varThreshold=250`: Hareketli nesnelerin tanımlanabilmesi için kullanılacak olan varyans eşik değeridir. Yüksek bir değer, daha az hassasiyetle daha büyük değişikliklerin algılanmasını sağlar.

Video Karesini Okuma ve Arka Plan Çıkarma

```
ret, frame = cap.read()
```

`cap.read()` fonksiyonu, videodan bir kare okur. `ret`, videodan başarılı bir şekilde bir kare okunup okunmadığını gösterir (True ya da False), `frame` ise okunan görüntü karesini tutar.

Eğer kare başarıyla okunmuşsa (`ret` True ise), işlem devam eder.

Arka Plan Maskesi ve Arka Plan Görüntüsünü Uygulama

```
fgmask = fgbg.apply(frame)
background = fgbg.getBackgroundImage()
```

`fgmask = fgbg.apply(frame)`: Bu fonksiyon, mevcut kareyi alır ve hareketli nesneleri arka plandan ayırarak bir **mask** oluşturur. Mask, nesnelerin bulunduğu bölgeleri beyaz yaparken, arka planı siyah tutar.

`background = fgbg.getBackgroundImage()`: Bu fonksiyon, arka planın şu anki halini döndürür. Arka plan, hareketli nesnelerden arındırılmış bir görüntü olur.

Görüntüleri Gösterme

```
cv.imshow('input', frame)
cv.imshow('fgmask', fgmask)
if background is not None:
    cv.imshow('background', background)
```

- `cv.imshow('input', frame)`: Orijinal video karesini ekranda gösterir.
- `cv.imshow('fgmask', fgmask)`: Maskelenmiş görüntüyü (arka plandan ayrılmış nesneleri) ekranda gösterir.
- `cv.imshow('background', background)`: Arka plan görüntüsünü (hareketsiz kısmı) gösterir, ancak arka plan yoksa (`background is None`), bu görüntü gösterilmez.

Kullanıcıdan 'Esc' Tuşu ile Çıkma

```
k = cv.waitKey(10) & 0xFF
if k == 27:
    break
```

`cv.waitKey(10)`: Bu fonksiyon, 10 milisaniye kadar bir tuşa basılmasını bekler. Eğer bir tuşa basılmadıysa, kod devam eder. Eğer tuşa basılırsa, basılan tuşun ASCII kodunu döndürür.

`& 0xFF`: Tuşun kodunu 8 bitlik bir değere indirger.

Eğer kullanıcı **Esc** tuşuna (`27` ASCII kodu) basarsa, döngüden çıkılır ve video işlemi sonlanır.

Kaynağı Serbest Bırakma ve Pencereleri Kapatma

cap.release()

cv.destroyAllWindows()

haraketli nesneleri video dan çıkarma:

```
import cv2 as cv
import numpy as np

cap = cv.VideoCapture("yürü.mp4")
fgbg = cv.createBackgroundSubtractorMOG2(history=500, varThreshold=100)



def process(image):
    mask = fgbg.apply(image)
    line = cv.getStructuringElement(cv.MORPH_RECT, (1, 5), (-1, -1))
    mask = cv.morphologyEx(mask, cv.MORPH_OPEN, line)
    cv.imshow("mask", mask)
    contours, hierarchy = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    for c in range(len(contours)):
        area = cv.contourArea(contours[c])
        if area > 150:
            rect = cv.minAreaRect(contours[c])
            cv.ellipse(image, rect, (0, 255, 0), 2, 8)
            center = np.int32(rect[0])
            cv.circle(image, tuple(center), 2, (255, 0, 0), 2, 8)

    return image



while True:
    ret, frame = cap.read()
    if not ret:
        break

    fgmask = fgbg.apply(frame)
    background = fgbg.getBackgroundImage()

    cv.imshow('input', frame)
    cv.imshow('fgmask', fgmask)
    cv.imshow('background', background)
```

```
    processed_frame = process(frame)
    cv.imshow('processed_frame', processed_frame)

    k = cv.waitKey(10) & 0xFF
    if k == 27:  # Escape key to exit
        break

cap.release()
cv.destroyAllWindows()
```

açıklama:

Video Yükleme

cap = cv.VideoCapture("yürü.mp4")

Bu satır, `"yürü.mp4"` adlı video dosyasını açmak için `cv.VideoCapture` fonksiyonunu kullanır. `cap` nesnesi video akışını temsil eder.

Arka Plan Çıkarma (Background Subtraction)

fgbg = cv.createBackgroundSubtractorMOG2(history=500, varThreshold=100)

Bu satır, `cv.createBackgroundSubtractorMOG2` fonksiyonuyla bir arka plan çıkarıcı oluşturur. `MOG2` (Mixture of Gaussians) algoritması kullanılarak hareketli nesneler tespit edilir.

- `history=500`: Arka plan modelinin geçmişindeki kare sayısı.
- `varThreshold=100`: Arka plan modelinin güncellenmesinde kullanılan değişim eşiği.

`process` Fonksiyonu

```
def process(image):
    mask = fgbg.apply(image)
    line = cv.getStructuringElement(cv.MORPH_RECT, (1, 5), (-1, -1))
    mask = cv.morphologyEx(mask, cv.MORPH_OPEN, line)
    cv.imshow("mask", mask)
    contours, hierarchy = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```

- `fgbg.apply(image)`: Bu fonksiyon, her kareyi alır ve `fgbg` arka plan çıkarıcısı ile o karedeki hareketli nesneleri tespit eder. Elde edilen maske, arka plandan farklı olan (hareketli) bölümleri gösterir.
- `cv.getStructuringElement`: Bu fonksiyon, maske üzerinde morfolojik işlemler yapabilmek için bir yapısal element oluşturur. Burada dikdörtgen bir yapı (`(1, 5)` boyutlarında) seçilmiş.
- `cv.morphologyEx`: Bu fonksiyon, maskeyi iyileştirmek için "açma" (opening) işlemi uygular. Bu işlem, maskede gürültüleri ortadan kaldırır.
- `cv.findContours`: Maskedeki dış hatları bulur ve `contours` listesine kaydeder. Bu, hareketli nesnelerin şekillerini çıkarabilmemize olanak tanır.

Konturları İşleme ve Çizim

```
for c in range(len(contours)):
    area = cv.contourArea(contours[c])
    if area > 150:
        rect = cv.minAreaRect(contours[c])
        cv.ellipse(image, rect, (0, 255, 0), 2, 8)
        center = np.int32(rect[0])
        cv.circle(image, tuple(center), 2, (255, 0, 0), 2, 8)
```

- `cv.contourArea`: Her konturun alanını hesaplar.
- `cv.minAreaRect`: Bir konturun etrafında en küçük alanı kapsayan dikdörtgeni bulur.
- `cv.ellipse`: Elde edilen dikdörtgeni elips şeklinde çizer.
- `cv.circle`: Elde edilen dikdörtgenin merkezine bir nokta çizer.

Buradaki amaç, maskede hareketli nesneleri tespit ettikten sonra bu nesnelerin etrafını çizmek ve merkezi noktalarını belirlemektir.

Ana Döngü

```
while True:
    ret, frame = cap.read()
    if not ret:
        break
```

Bu döngü, videonun her bir karesini okur ve işlemi her bir kare için tekrarlar. Eğer video bitmişse (`ret` False dönerse), döngü sonlanır.

Arka Plan Maskesi ve Arka Plan Görüntüsü

```
fgmask = fgbg.apply(frame)
background = fgbg.getBackgroundImage()
```

- `fgmask`: Bu, her kareye uygulanan arka plan çıkarıcıdan elde edilen maskedir (hareketli nesnelerin bulunduğu bölgeleri gösterir).
- `background`: Bu, arka plan çıkarıcının sürekli olarak güncellediği ve oluşturduğu sabit arka plan görüntüsüdür.

Ekranda Görüntüleme

```
cv.imshow('input', frame)
cv.imshow('fgmask', fgmask)
cv.imshow('background', background)
processed_frame = process(frame)
cv.imshow('processed_frame', processed_frame)
```

- `cv.imshow('input', frame)`: Orijinal video karesi.
- `cv.imshow('fgmask', fgmask)`: Arka plandan çıkarılmış maskenin görüntüsü.
- `cv.imshow('background', background)`: Arka plan çıkarıcısının oluşturduğu arka plan görüntüsü.
- `cv.imshow('processed_frame', processed_frame)`: İşlenmiş kare, hareketli nesnelerle birlikte.

Çıkış Kontrolü

```
k = cv.waitKey(10) & 0xFF
```

if k == 27: # Escape key to exit

  break

Burada `cv.waitKey(10)` fonksiyonu, 10 milisaniye boyunca bir tuşa basılıp basılmadığını kontrol eder. Eğer "Esc" tuşuna basılırsa (`k == 27`), program durur.

Kaynakları Serbest Bırakma

cap.release()

cv.destroyAllWindows()

video kaynağını serbest bırakır ve tüm OpenCV pencerelerini kapatır.

yogunluk temelli optik akış:

```
# OpenCV ve NumPy kütüphanelerini içeri aktarıyoruz.
import cv2 as cv
import numpy as np

# "yürü.mp4" adlı video dosyasını açıyoruz.
cap = cv.VideoCapture("yürü.mp4")

# İlk kareyi (frame1) okuyoruz.
ret, frame1 = cap.read()

# Eğer video açılamazsa veya ilk kare okunamazsa, hata mesajı gösterip çıkıyoruz.
if not ret:
    print("Video dosyası açılamadı.")
    cap.release()  # Video kaynağını serbest bırakıyoruz.
    cv.destroyAllWindows()  # OpenCV pencerelerini kapatıyoruz.
    exit()  # Programı sonlandırıyoruz.

# İlk kareyi gri tonlamalıya çeviriyoruz (optik akış hesaplaması için gri tonlama gereklidir).
prvs = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)

# HSV renk uzayında bir görüntü oluşturuyoruz, burada sadece S (doygunluk) kanalını 125 olarak ayarlıyoruz.
hsv = np.zeros_like(frame1)  # 'frame1' boyutunda sıfırlarla bir dizi oluşturuyoruz.
hsv[..., 1] = 125  # HSV'de doygunluk kanalını 125 yapıyoruz.

# Dense optik akış fonksiyonu, her bir karedeki hareketi izlemek için hesaplamalar yapacak.
def dense_opt_flow(hsv, prvs):
    while True:
        # Bir sonraki kareyi okuyoruz.
        ret, frame2 = cap.read()

        # Eğer daha fazla kare yoksa (video sonlanmışsa), döngüden çıkıyoruz.
```

```
        if not ret:
            break

        # İkinci kareyi de gri tonlamalıya çeviriyoruz.
        nextt = cv.cvtColor(frame2, cv.COLOR_BGR2GRAY)

        # Farneback yöntemini kullanarak optik akışı hesaplıyoruz.
        # 'prvs' ve 'nextt' kareleri arasındaki hareketi (akışı) hesaplıyoruz.
        flow = cv.calcOpticalFlowFarneback(prvs, nextt, None, 0.5, 3, 15, 3, 5, 1.2, 0)

        # Akışın büyüklüğünü (mag) ve yönünü (ang) hesaplıyoruz.
        mag, ang = cv.cartToPolar(flow[..., 0], flow[..., 1])

        # HSV görüntüsündeki açı (yön) kanalını güncelliyoruz.
        hsv[..., 0] = ang * 180 / np.pi / 2  # Açıyı derece cinsinden, 0-180 arası değerler olacak şekilde hesaplıyoruz.

        # HSV görüntüsündeki parlaklık (value) kanalını, büyüklük (magnitude) ile normalleştiriyoruz.
        hsv[..., 2] = cv.normalize(mag, None, 0, 255, cv.NORM_MINMAX)

        # HSV görüntüsünü BGR renk uzayına dönüştürüyoruz ve bunu görüntülüyoruz.
        bgr = cv.cvtColor(hsv, cv.COLOR_HSV2BGR)
        cv.imshow('Optical Flow', bgr)  # Optik akışın renkli görsellemesini gösteriyoruz.

        # Gerçek video çerçevesini de gösteriyoruz.
        cv.imshow("Frame", frame2)

        # Kullanıcı 'ESC' tuşuna basarsa döngüden çıkıyoruz.
        k = cv.waitKey(30) & 0xFF
        if k == 27:
            break

        # Önceki kareyi, şu anki kare olarak güncelliyoruz.
        prvs = nextt

# Dense optik akış fonksiyonunu çağırıyoruz.
dense_opt_flow(hsv, prvs)

# Video kaynağını serbest bırakıyoruz.
cap.release()

# OpenCV pencerelerini kapatıyoruz.
cv.destroyAllWindows()
```

```
import cv2 as cv
import numpy as np

src = cv.imread('agla.jpg')

if src is None:
    print("Image not found!")
    exit()

src = cv.resize(src, (0, 0), fx=0.5, fy=0.5)

r = cv.selectROI('input', src, False)

roi = src[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2])]
img = src.copy()

cv.rectangle(img, (int(r[0]), int(r[1])), (int(r[0] + r[2]), int(r[1] + r[3])), (255, 0, 0), 2)

mask = np.zeros(src.shape[:2], dtype=np.uint8)

rect = (int(r[0]), int(r[1]), int(r[2]), int(r[3]))

bgdmodel = np.zeros((1, 65), np.float64)
fgdmodel = np.zeros((1, 65), np.float64)

cv.grabCut(src, mask, rect, bgdmodel, fgdmodel, 5, mode=cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask == 1) + (mask == 3), 255, 0).astype('uint8')

background = cv.imread('back.png')

h, w, ch = src.shape
background = cv.resize(background, (w, h))
mask = np.zeros(src.shape[:2], dtype=np.uint8)
bgdmodel = np.zeros((1, 65), np.float64)
fgdmodel = np.zeros((1, 65), np.float64)

cv.grabCut(src, mask, rect, bgdmodel, fgdmodel, 5, mode=cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 1) + (mask == 3), 255, 0).astype('uint8')

se = cv.getStructuringElement(cv.MORPH_RECT, (3, 3))
cv.dilate(mask2, se, mask2)
```

```
mask2 = cv.GaussianBlur(mask2, (5, 5), 0)


background = cv.GaussianBlur(background, (0, 0), 15)
mask2 = mask2 / 255.0
a = mask[..., None]
result = a * (src.astype(np.float32) + (1 - a) * (background.astype(np.float32)))
cv.imshow("result", result.astype(np.uint8))
cv.waitKey(0)
cv.destroyAllWindows()
```

kodun açıklaması:

Görüntünün Okunması ve Kontrol Edilmesi

```
src = cv.imread('agla.jpg')

if src is None:
    print("Image not found!")
    exit()
```

- `cv.imread('agla.jpg')`: `'agla.jpg'` dosyasını okuyup `src` değişkenine atar. Görüntü başarıyla okunmazsa, `None` dönebilir.
- Eğer görüntü bulunamazsa, "Image not found!" yazdırılır ve program sonlanır.

Görüntünün Yeniden Boyutlandırılması

```
src = cv.resize(src, (0, 0), fx=0.5, fy=0.5)
```

`cv.resize()` fonksiyonu, görüntünün boyutlarını değiştirir. `fx` ve `fy` parametreleri görüntüyü yatay ve dikey olarak %50 küçültür.

ROI (Region of Interest) Seçimi

```
r = cv.selectROI('input', src, False)
```

`cv.selectROI()` fonksiyonu, kullanıcıya fare ile bir dikdörtgen seçme imkanı sunar. Bu dikdörtgen, işlenecek bölgeyi belirler. Sonuç olarak `r`, ROI'nin koordinatlarını (x, y, genişlik, yükseklik) döndüren bir tuple olur.

ROI'nin Kesilmesi ve Dikdörtgenin Çizilmesi

```
roi = src[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2])]
img = src.copy()

cv.rectangle(img, (int(r[0]), int(r[1])), (int(r[0] + r[2]), int(r[1] + r[3])), (255, 0, 0), 2)
```

- ROI bölgesini `src` görüntüsünden kesip `roi` olarak saklar.
- `cv.rectangle()` fonksiyonu, seçilen ROI'nin etrafına bir dikdörtgen çizer. Bu dikdörtgen, (255, 0, 0) renkli (mavi) ve 2 px kalınlığında olur.

GrabCut Algoritmasıyla Arka Planın Ayrılması

```
mask = np.zeros(src.shape[:2], dtype=np.uint8)
rect = (int(r[0]), int(r[1]), int(r[2]), int(r[3]))
bgdmodel = np.zeros((1, 65), np.float64)
fgdmodel = np.zeros((1, 65), np.float64)

cv.grabCut(src, mask, rect, bgdmodel, fgdmodel, 5, mode=cv.GC_INIT_WITH_RECT)
mask2 = np.where((mask == 1) + (mask == 3), 255, 0).astype('uint8')
```

GrabCut Algoritması: Bu algoritma, görüntüdeki ön plan ve arka planı ayırmaya yarar. `cv.grabCut()` fonksiyonu ile belirtilen ROI içindeki arka plan ve ön plan ayrılır.

- `mask`: Görüntüdeki her pikselin hangi sınıfa ait olduğunu gösteren bir maske oluşturulur.
- `mask2`: Maskede belirli sınıflara (ön plan ve arka plan) karşılık gelen piksellerin değeri 255, diğerlerinin 0 olarak değiştirilir.

Yeni Arka Planın Yüklenmesi ve Yeniden Boyutlandırılması

```
background = cv.imread('back.png')
h, w, ch = src.shape
background = cv.resize(background, (w, h))
```

Yeni bir arka plan (`back.png`) dosyası okunur ve bu arka plan, seçilen görüntünün boyutlarına göre yeniden boyutlandırılır.

Maskenin İşlenmesi (Dilatasyon ve Bulanıklaştırma)

```
se = cv.getStructuringElement(cv.MORPH_RECT, (3, 3))
cv.dilate(mask2, se, mask2)

mask2 = cv.GaussianBlur(mask2, (5, 5), 0)
```

- **Dilatasyon**: `cv.dilate()` fonksiyonu, maskeyi genişletir. Bu işlem, maskenin sınırlarını belirginleştirir.
- **Gaussian Bulanıklaştırma**: `cv.GaussianBlur()` fonksiyonu, maskeyi ve arka planı bulanıklaştırarak daha yumuşak bir geçiş sağlar.

Arka Plan ve Ön Planın Birleştirilmesi

```
background = cv.GaussianBlur(background, (0, 0), 15)
mask2 = mask2 / 255.0
a = mask[..., None]
result = a * (src.astype(np.float32) + (1 - a) * (background.astype(np.float32)))
```

- **Gaussian Bulanıklaştırma**: Arka plan üzerine ek bir bulanıklaştırma uygulanır.
- **Birleştirme**: `result`, orijinal görüntü ile yeni arka planın birleşimidir. Maskenin (ön plan) ve arka planın karışımıyla son görüntü elde edilir.

Sonuç Görüntüsünün Gösterilmesi

cv.imshow("result", result.astype(np.uint8))

cv.waitKey(0)

cv.destroyAllWindows()

----------OPEN CV 401----------

kodları:

```
import cv2 as cv
import numpy as np

# Görüntüyü yükle
src = cv.imread("opencv.png")

# Tuz ve karabiber gürültüsü ekleyen fonksiyon
def add_salt_pepper_noise(image):
    h, w = image.shape[:2]
    nums = 1000  # Gürültü piksel sayısı
    rows = np.random.randint(0, h, nums, dtype=int)  # Satır indeksleri
    cols = np.random.randint(0, w, nums, dtype=int)  # Sütun indeksleri

    for i in range(nums):
        if i % 2 == 1:
            image[rows[i], cols[i]] = (255, 255, 255)  # Beyaz noktalar
        else:
            image[rows[i], cols[i]] = (0, 0, 0)  # Siyah noktalar
    return image

# Görüntü boyutlarını al
h, w = src.shape[:2]

# Gürültülü bir kopya oluştur
copy = np.copy(src)
copy = add_salt_pepper_noise(copy)

# Sonuç görüntüsünü birleştirmek için yer ayır
result = np.zeros([h, w * 2, 3], dtype=src.dtype)
result[0:h, 0:w, :] = src
```

```
result[0:h, w:2 * w, :] = copy

# Sonuçları göster ve kaydet
cv.imshow("Original and Noisy", result)
cv.imwrite("result.png", result)
cv.waitKey(0)
cv.destroyAllWindows()
```

kod:

```
import cv2 as cv
import numpy as np

# Görseli oku
src = cv.imread("opencv.png")

# Sharpen filtre matrisi
sharpen_op = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, 1, 0]], dtype=np.float32)

# Filtreyi uygula
sharpen_image = cv.filter2D(src, cv.CV_32F, sharpen_op)

# Değerleri 8 bit formatına dönüştür
sharpen_image = cv.convertScaleAbs(sharpen_image)

# Sonucu göster
cv.imshow("Sharpened Image", sharpen_image)
cv.waitKey(0)
cv.destroyAllWindows()
```

kod:

```
import cv2 as cv
import numpy as np

src = cv.imread("satro.png")



def harris(image):
    blocksize = 2
    aperturesize = 3
    k = 0.04
```

```
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    dst = cv.cornerHarris(gray, blocksize, aperturesize, k)
    dst_norm = np.empty(dst.shape, dtype=np.float32)
    cv.normalize(dst, dst_norm, alpha=0, beta=255, norm_type=cv.NORM_MINMAX)

    for i in range(dst_norm.shape[0]):
        for j in range(dst_norm.shape[1]):
            if int(dst_norm[i, j]) > 120:
                cv.circle(image, (j, i), 2, (0, 255, 0), 2)

    return image


result = harris(src)

cv.imshow('result', result)
cv.waitKey(0)
cv.destroyAllWindows()
```

açıklaması:

```
Harris Köşe Tespiti Fonksiyonu:

def harris(image):bu fonksiyon, Harris köşe tespiti algoritmasını içerir.
Fonksiyon, image parametresi olarak alınan görüntü üzerinde köşe tespiti yapacaktır.
```

Harris Algoritmasının Parametreleri:

blocksize = 2
aperturesize = 3
k = 0.04

- `blocksize`: Her bir pikselin komşuluk alanı boyutudur. Bu, köşe belirlemek için kullanılan lokal bölgelerin boyutunu belirler. Burada, 2x2 bir komşuluk alanı kullanılıyor.
- `aperturesize`: Sobel operatörünün kullanılacağı pencere boyutudur. Burada 3x3 bir pencere kullanılmaktadır.
- `k`: Harris algoritmasının köşe tepkisini hesaplamak için kullanılan sabit bir çarpandır. Bu parametre genellikle 0.04 ile 0.06 arasında seçilir.

Köşe Tespiti Hesaplaması:

dst = cv.cornerHarris(gray, blocksize, aperturesize, k)
`cv.cornerHarris()`, verilen gri tonlamalı görüntüde Harris köşe tespiti algoritmasını uygular. Sonuç, her pikselin köşe olma olasılığını içeren bir 2D dizisi (`dst`) olarak döner.

onucun Normalizasyonu:

dst_norm = np.empty(dst.shape, dtype=np.float32)

cv.normalize(dst, dst_norm, alpha=0, beta=255, norm_type=cv.NORM_MINMAX)

`cv.normalize()`, `dst` matrisini [0, 255] aralığına normalize eder. Bu işlem, tespit edilen köşelerin belirginliğini artırarak görsel olarak daha kolay algılanmalarını sağlar.

Köşe Noktalarını İşaretleme:

for i in range(dst_norm.shape[0]):

  for j in range(dst_norm.shape[1]):

    if int(dst_norm[i, j]) > 120:

      cv.circle(image, (j, i), 2, (0, 255, 0), 2)

- Bu döngü, `dst_norm` matrisindeki her pikseli kontrol eder. Eğer bir pikselin değeri 120'den büyükse, bu piksel bir köşe olarak kabul edilir.
- c`v.circle()`, bu köşe noktalarına yeşil renkli (0, 255, 0) küçük daireler çizer.

Sonuçları Döndürme:

return image

Bu satırda, üzerinde köşe tespiti yapılmış olan görüntü (`image`) geri döndürülür.

Köşe Tespiti Fonksiyonunun Çalıştırılması ve Görüntünün Gösterilmesi:

result = harris(src)

cv.imshow('result', result)

cv.waitKey(0)

cv.destroyAllWindows()

- `harris(src)` fonksiyonu çağrılarak köşe tespiti yapılır ve `result` değişkenine atanır.
- `cv.imshow()` fonksiyonu ile sonuç görüntüsü ekranda gösterilir.
- `cv.waitKey(0)` ile görüntü, bir tuşa basılana kadar ekranda kalır.
- `cv.destroyAllWindows()` ile tüm OpenCV pencereleri kapatılır.

kod:

```
import cv2 as cv
import numpy as np

# "b.jpg" adlı resmi oku
src = cv.imread("b.jpg")

def process(image):
    # Resmi gri tonlamaya çevir (köşe tespiti için gereklidir)
```

```
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Gri tonlamalı resimde köşeleri tespit et
    # maxCorners: Tespit edilecek maksimum köşe sayısı (35)
    # qualityLevel: Köşelerin kalitesi (0.05, %5 kalite eşiği)
    # minDistance: Tespit edilen köşeler arasındaki minimum mesafe (10 piksel)
    corners = cv.goodFeaturesToTrack(gray, maxCorners=35,
                                     qualityLevel=0.05, minDistance=10)

    # Eğer köşeler tespit edildiyse (corners None ise, köşe yok demektir)
    if corners is not None:
        # Her bir köşe noktasında dön
        for pt in corners:
            # Köşe için rastgele renkler oluştur
            b = np.random.randint(0, 256)  # Rastgele mavi renk değeri
            g = np.random.randint(0, 256)  # Rastgele yeşil renk değeri
            r = np.random.randint(0, 256)  # Rastgele kırmızı renk değeri

            # Köşenin x ve y koordinatlarını al (pt[0] köşenin koordinatlarıdır)
            x = np.int32(pt[0][0])
            y = np.int32(pt[0][1])

            # Köşenin etrafına rastgele renk ile bir daire çiz
            cv.circle(image, (x, y), 5, (int(b), int(g), int(r)), 2)  # 5 piksellik bir yarıçap, 2 kalınlık
    return image

# Girdi resmini işleyerek tespit edilen köşeleri işaretle
result = process(src)

# Sonuç resmini %50 oranında küçült
result_resized = cv.resize(result, (0, 0), fx=0.5, fy=0.5)

# Sonuç resmini 'result' adlı bir pencere içinde göster
cv.imshow('result', result_resized)

# Bir tuşa basana kadar bekle (resmi göstermek için gerekli)
cv.waitKey(0)

# Tüm OpenCV pencerelerini kapat
cv.destroyAllWindows()
```

kod:

```
import cv2 as cv  # OpenCV kütüphanesini cv olarak içeri aktarır.
import numpy as np  # NumPy kütüphanesini np olarak içeri aktarır.

# "ablok.jpg" adlı resmi okur ve 'src' değişkenine atar.
src = cv.imread("ablok.jpg")

# Resim üzerinde köşe tespiti yapan bir işlem fonksiyonu tanımlar.
def process(image):
    # Resmi gri tonlamaya dönüştürür (renkli resimden gri tonlamalıya).
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Gri tonlamalı resim üzerinde köşeleri tespit eder.
    # `maxCorners=35` en fazla 35 köşe arar, `qualityLevel=0.01` köşelerin kalitesini belirler,
    # `minDistance=10` köşeler arasındaki minimum mesafeyi belirtir.
    corners = cv.goodFeaturesToTrack(gray, maxCorners=35, qualityLevel=0.01, minDistance=10)

    # Eğer köşeler tespit edilmişse, aşağıdaki işlemleri yapar:
    if corners is not None:
        print("Köşe sayısı:", len(corners))  # Tespit edilen köşe sayısını ekrana yazdırır.

        # Tespit edilen her köşe için bir daire çizer.
        for pt in corners:
            x, y = np.int32(pt[0])  # Köşe noktasının (x, y) koordinatlarını alır.
            # Rastgele bir renk oluşturur.
            b = np.random.randint(0, 256)
            g = np.random.randint(0, 256)
            r = np.random.randint(0, 256)
            # Rastgele renk ile köşe noktasını işaretler.
            cv.circle(image, (x, y), 5, (int(b), int(g), int(r)), 2)

        # Köşe noktalarını daha hassas şekilde düzeltebilmek için bazı parametreler ayarlanır.
        winSize = (3, 3)  # Pencerelerin boyutları (3x3).
        zeroZone = (-1, -1)  # 'zeroZone' boş bir bölgeyi belirtir (bu, pencere hareketini engeller).
        criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_COUNT, 40, 0.001)  # Düzeltme için kriterler belirler.

        # `cornerSubPix`, köşe noktalarını daha hassas bir şekilde düzeltir.
        refined_corners = cv.cornerSubPix(gray, corners, winSize, zeroZone, criteria)

        # Düzeltildiği köşe noktalarını yazdırır.
        for i in range(refined_corners.shape[0]):
```

```
            print(" -- Refined Corner [", i, "] (", refined_corners[i, 0, 0], ",", refined_corners[i, 0, 1], ")")

    return image  # İşlem tamamlanmış resmi döndürür.

# İşlem fonksiyonunu çağırarak resmi işler.
processed_image = process(src)

# Resmi küçültme işlemi yapar. Boyutlarını yarıya indirir.
height, width = processed_image.shape[:2]  # Resmin boyutlarını alır (yükseklik, genişlik).
new_width = int(width * 0.5)  # Yeni genişlik: Eski genişliğin yarısı.
new_height = int(height * 0.5)  # Yeni yükseklik: Eski yüksekliğin yarısı.
resized_image = cv.resize(processed_image, (new_width, new_height))  # Resmi yeni boyutlara göre yeniden boyutlandırır.

# İşlenmiş resmi gösterir.
cv.imshow('Processed Image', resized_image)

# Kullanıcı bir tuşa basana kadar programın devam etmesini sağlar.
cv.waitKey(0)

# Tüm pencereleri kapatır.
cv.destroyAllWindows()
```

kod

```
import cv2
import numpy as np

path = "./"

def template_demo():
    src = cv2.imread(path + "opem.jpg")
    tpl = cv2.imread(path + "openc.png")

    if src is None or tpl is None:
        print("Gorseller yuklenemedi!")
        return

    cv2.imshow("Ana Gorsel", src)
    cv2.imshow("Sablon Gorsel", tpl)

    th, tw = tpl.shape[:2]
```

```
    result = cv2.matchTemplate(src, tpl, cv2.TM_CCOEFF_NORMED)

    t = 0.8
    loc = np.where(result >= t)

    for pt in zip(*loc[::-1]):
        cv2.rectangle(src, pt, (pt[0] + tw, pt[1] + th), (0, 255, 0), 2)

    cv2.imshow("Eslesen Sablonlar", src)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

template_demo()
```

----------OpenCV 501 Ders Program----------

QR Kodu Algılama

```
################################
# UYGULAMA: QR CODE DETECT AND DECODE
################################

import cv2 as cv
import numpy as np

path = ""
src = cv.imread(path + "qrcode.png")

gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)

qrcoder = cv.QRCodeDetector()

codeinfo, points, straight_qrcode = qrcoder.detectAndDecode(gray)

qrcoder = cv.QRCodeDetector()

codeinfo, points, straight_qrcode = qrcoder.detectAndDecode(gray)

print(points)

result = np.copy(src)
```

```
cv.drawContours(result, [np.int32(points)], 0, (0, 0, 255), 2)


print("qrcode information is :\n%s" % codeinfo)
cv.imshow("result",result)
cv.waitKey(0)
//çıktısı:qrcode information is :
https://www.qrcode-monkey.com
```

kodun açıklaması:

Bu kod, bir QR kodunu algılamak ve çözmek için OpenCV'nin sağladığı araçları kullanıyor. İşleyiş adım adım şu şekildedir:

---

1. Görüntü Yükleme

- `cv.imread(path + "qrcode.png")`: Dosya yolunda bulunan "qrcode.png" adlı resmi okur ve bir matris (numpy dizisi) olarak `src` değişkenine atar.

---

2. Gri Tonlamalı Dönüşüm

- `cv.cvtColor(src, cv.COLOR_BGR2GRAY)`: RGB (renkli) görüntüyü gri tonlamalı görüntüye çevirir. QR kod tespit algoritmaları genellikle gri tonlamalı görüntülerde çalıştığı için bu adım yapılır.

---

3. QR Kod Algılama ve Çözme

- `QRCodeDetector` sınıfı, QR kodu algılamak ve içeriğini (kod bilgisini) çözmek için kullanılır.
- `detectAndDecode` yöntemi:
  - QR kodu algılar.
  - Eğer bir QR kod tespit edilirse, **`codeinfo`** değişkenine kod içeriğini (örneğin bir metin veya URL) döndürür.
  - QR kodun dört köşe noktasını **`points`** değişkenine döndürür.
  - Normalleştirilmiş (düzeltilmiş) QR kod görüntüsünü **`straight_qrcode`** değişkenine döndürür.

---

4. Sonuç Görselleştirme

- Algılanan QR kodun etrafına bir kırmızı çerçeve çizilir:
  - `cv.drawContours`: Algılanan QR kodun dört köşe noktasını birleştirerek bir çokgen (kontur) oluşturur.
  - `(0, 0, 255)`: Kırmızı renk kodu (BGR formatında).

    - `2`: Çizgi kalınlığı.
- `cv.imshow`: Görüntüyü bir pencerede görüntüler.
- `cv.waitKey(0)`: Kullanıcı bir tuşa basana kadar pencerenin açık kalmasını sağlar.

---

5. Sonuçların Yazdırılması

- `print(points)`: QR kodun algılanan köşe noktalarını ekrana yazdırır.
- `print("qrcode information is :\n%s" % codeinfo)`: QR kodun içerdiği bilgiyi ekrana yazdırır.

---

6. Tekrar Eden Kod Bloku

Kodda QR kodun tespiti ve çözülmesi işlemi iki kez yapılmış:

```
qrcoder = cv.QRCodeDetector()codeinfo, points, straight_qrcode = qrcoder.detectAndDecode(gray)
```

Ancak bu işlem yalnızca bir kez yeterlidir. İkinci tekrar bir işlevsellik sağlamaz ve kodun gereksiz yere uzamasına neden olur.

Gerçek zamanlı nesne tanıma, bir görüntü ya da video akışındaki nesneleri hızlı ve doğru bir şekilde tespit etmek için kullanılan bir bilgisayarlı görü teknolojisidir. Bu alanda yaygın olarak kullanılan iki temel yöntem **Derin Sinir Ağları (DNN)** ve **Tek Şut Dedektörü (SSD)** yöntemleridir.

---

1. DNN (Deep Neural Networks)

Derin Sinir Ağları, yapay sinir ağlarının çok katmanlı bir yapısıdır ve genellikle derin öğrenme teknikleriyle eğitilir. DNN'ler görüntü işleme ve nesne tanıma için çeşitli modellerin temelini oluşturur.

Avantajları:

- **Esneklik**: DNN, çeşitli nesne tanıma görevleri için özelleştirilebilir.
- **Doğruluk**: Yüksek doğruluk oranına sahiptir çünkü veriyi işlemek için çok katmanlı, karmaşık bir öğrenme sürecine sahiptir.
- **Özellik Öğrenimi**: DNN'ler, görüntüdeki yüksek ve düşük seviyeli özellikleri otomatik olarak öğrenir.

Dezavantajları:

- **Zaman ve Kaynak Kullanımı**: Eğitim ve çıkarım sırasında çok fazla hesaplama kaynağı gerektirir.
- **Gerçek Zamanlı İşleme**: DNN tek başına gerçek zamanlı işleme için genellikle yavaştır; bu nedenle optimizasyon gereklidir.

Kullanım Alanı:

DNN genellikle **Faster R-CNN** gibi modellerde kullanılır. Bu model, nesne bölgelerini belirlemek için RPN (Region Proposal Network) kullanır ve ardından bu bölgelerdeki nesneleri sınıflandırır.

---

2. SSD (Single Shot MultiBox Detector)

SSD, tek bir sinir ağı üzerinden görüntüdeki nesneleri tespit eden ve sınıflandıran bir modeldir. Adından da anlaşılacağı gibi, **tek bir adımda (single shot)** hem sınıflandırma hem de konumlandırma işlemlerini yapar.

Özellikleri:

- **Hızlı Çalışma**: SSD, hızlı bir şekilde çalışır ve gerçek zamanlı nesne tanıma için uygundur.
- **Çok Ölçekli Algılama**: Farklı boyutlardaki nesneleri tespit etmek için çok ölçekli öznitelik haritalarını kullanır.
- **End-to-End Eğitim**: SSD, doğrudan giriş görüntüsünden sonuç üretecek şekilde eğitilir.

Avantajları:

- **Hafiflik ve Hız**: SSD, daha az karmaşıklık ve hesaplama gereksinimi ile yüksek hız sağlar.
- **Gerçek Zamanlı Performans**: Özellikle düşük güçlü cihazlarda ve gömülü sistemlerde bile iyi sonuç verir.

Dezavantajları:

- **Küçük Nesneler**: Küçük nesneleri algılamada performansı düşebilir.
- **Doğruluk**: DNN tabanlı yaklaşımlara göre biraz daha düşük doğruluk oranı olabilir.

Kullanım Alanı:

Mobil cihazlarda, gömülü sistemlerde ve otonom araçlarda nesne algılama gibi gerçek zamanlı gereksinimlerde sıkça kullanılır.

DNN ve SSD'nin Karşılaştırması

Özellik	DNN	SSD
Doğruluk	Daha yüksek	Orta-yüksek
Hız	Daha yavaş	Daha hızlı
Karmaşıklık	Daha karmaşık	Daha hafif ve basit
Kullanım Alanı	Yüksek doğruluk gereken durumlar	Gerçek zamanlı uygulamalar

---

Gerçek Zamanlı Kullanımda Ortak Yaklaşımlar

- **Optimizasyon Teknikleri**: Gerçek zamanlı nesne tanımada, modeller genellikle **TensorRT**, **ONNX** ya da **Pruning ve Quantization** gibi tekniklerle hızlandırılır.
- **Donanım Desteği**: GPU'lar ve TPU'lar gibi donanımlar, gerçek zamanlı işlemler için performansı artırır.
- **Hibrit Yaklaşımlar**: Yüksek doğruluk için DNN ve hızlı çıkarım için SSD bir arada kullanılabilir.

Bu yöntemler, kamera tabanlı güvenlik sistemleri, artırılmış gerçeklik, otonom araçlar ve robotik uygulamalar gibi birçok alanda yaygın şekilde kullanılmaktadır.