OPENCV Video İşlemleri

OpenCV'de video okuma, bilgisayarla görme ve görüntü işlemenin çok önemli bir yönüdür. OpenCV, bilgisayarla görme görevleri için çok çeşitli işlevler ve algoritmalar sağlayan açık kaynaklı bir kütüphanedir.

OpenCV'de video okuma, bir kameradan veya bir dosyadan video yakalama ve analiz ve işleme için videonun karelerini yorumlama işlemidir. Bu, gözetim, nesne izleme ve video düzenleme gibi bilgisayarla görme uygulamalarında yaygın bir görevdir.

OpenCV'de bir videoyu okumak için ilk adım cv2.VideoCapture() fonksiyonunu kullanarak bir video yakalama nesnesi oluşturmaktır. Bu fonksiyon, video dosyasının veya video akışının yolunu bir argüman olarak alır.

Video yakalama nesnesi oluşturulduktan sonra, videodan tek tek kareleri okumak için nesnenin read() yöntemini kullanabiliriz. Bu yöntem, okuma işleminin durumunu ve karenin kendisini içeren bir tuple döndürür.

Burada “ret” mantıksal değişkenini (True or False) kullanarak video okumanın başarılı olup olmadığını test edebiliriz.

Kareleri görüntülemek için, pencere adını ve kareyi argüman olarak alan cv2.imshow() fonksiyonunu kullanabiliriz. Çerçevelerin düzgün görüntülenmesi için imshow() fonksiyonundan sonra cv2.waitKey() fonksiyonunun kullanılması önemlidir.

OpenCV, kareleri okuma ve görüntülemeye ek olarak, kareleri yeniden boyutlandırma, döndürme ve çevirme gibi temel video manipülasyonu için işlevler de sağlar. Bu işlevler, daha fazla işlem veya analiz gerçekleştirmeden önce kareleri önceden işlemek için kullanılabilir.

cv2.VideoWriter() fonksiyonunu kullanarak kareleri yeni bir videoya yazmak da mümkündür. Bu fonksiyon çıkış video dosyası adını, fourcc kodunu, kare hızını ve kare boyutunu argüman olarak alır. Fourcc kodu, video codec bileşenini belirtmek için kullanılan 4 baytlık bir koddur.

Genel olarak, OpenCV'de video okuma, bilgisayarla görme uygulamaları için güçlü ve çok yönlü bir araçtır. Video verilerinin analizine ve manipülasyonuna izin vererek gözetim, video düzenleme ve nesne izleme gibi alanlarda çok çeşitli uygulamalara olanak tanır.

İlk Örnek:

Video Okuma

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

# Dosyadan video oku
x = cv2.VideoCapture('ormanYanginiVideo.mp4')

# Dosyadan Görüntü okuma başarılı olduğu sürece while döngüsü çalışsın
while(x.isOpened()):
    # Videodan bir çerçeve oku.
    ret, frame = x.read()
    frame=cv2.resize(frame,(768,432))

    # Gri seviyeye dönüştür
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Ekrandan görüntüleme
    cv2.imshow('frame',frame)
    #istenirse video görüntüleme yavaşlatılabilir.
    time.sleep(0.1)

    # q tuşuna basıldığında çık.
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# İşin bittikten sonra her şeyi serbest bırak.
x.release()
cv2.destroyAllWindows()
```

Video nesnesi yaratma işlemi

-Video okuma işlemi, -
-Kare kare okuyoruz. -
-"ret" dikkat ediniz.

Ekranda görüntüleme

Görüntülemeyi kapatma

Çerçeveler Üzerinde İşlemler:

BLOB Analizi:

BLOB (Binary Large Object) ikili büyük nesne anlamına gelir ve bir ikili görüntüde bağlı piksellerin bir grubunu ifade etmektedir. Büyük terimi belirli boyuttaki nesne olarak adlandırılır. Dolayısıyla büyük boyutun dışında kalan küçük nesneler gürültü olarak değerlendirilir. Görüntü işlemede nesne tespit etme konusunda özellikle yaygın kullanılır. Bir görüntüde odaklandığımız nesneleri genellikle önce ikilik görüntüye çevirip BLOB haline getiririz.

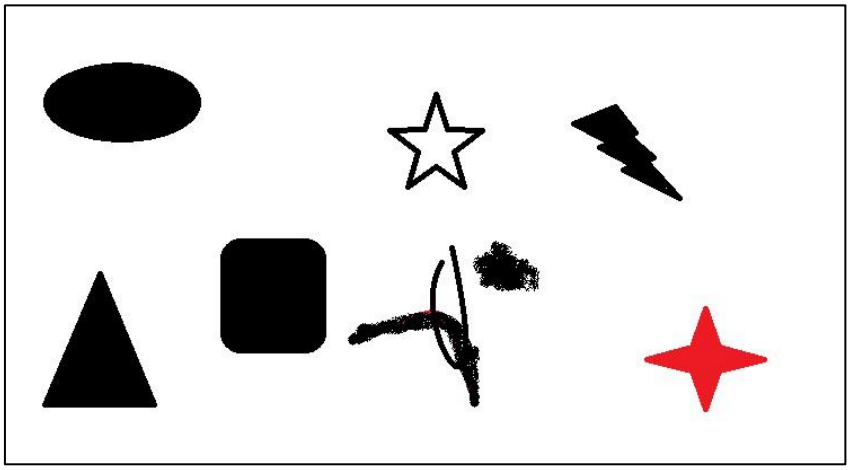
Şekil 2. BLOB analizi için örnek imge.

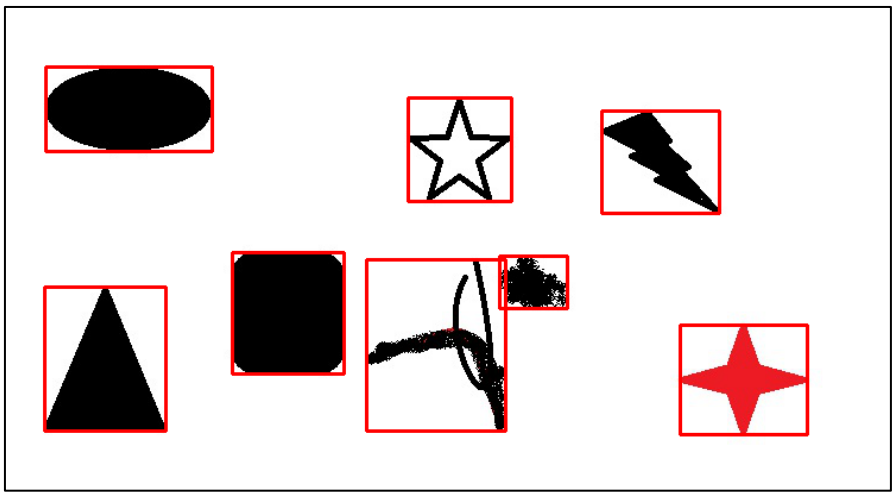
BLOB analizinin amacı: Bilgisayar görmesi, insan bilgisayar etkileşimi veya örüntü tanıma için nesnelerin etiketlenmesini ya da başka bir ifade ile ikili görüntüdeki büyük nesneleri diğerlerinden (gürültüden) ayırıp etiketleyip öznitelik verileri üretmektir.

Aşağıda bir görüntüdeki BLOB'ları tespit eden ve kapsayıcı kutu içine alan program verilmiştir.

```
import cv2
img = cv2.imread('blobs1.jpg')
# Gri seviye dönüşümü
gray_img = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)

# 7x7 gaussian blur filtresi uygulama
blurred = cv2.GaussianBlur(gray_img, (7, 7), 0)

# Eşikleme işlemi ile siyah/beyaz görüntü elde etme
abc,threshold = cv2.threshold(blurred,150,255,cv2.THRESH_BINARY_INV)

#  Bileşen (blob) analiz işlevini uygulayın
(toplamBlob, etiket_id, degerler, centroid)  = cv2.connectedComponentsWithStats(threshold, 4, cv2.CV_32S)

# Her bir blob için işlem yapılıyor.
for i in range(1, toplamBlob):

    # Blobların alanları
    area = degerler[i, cv2.CC_STAT_AREA]
    print(area)
    # Her bir blobun sol üst köşe, en ve boy koordinatları alınıyor.
    x1 = degerler[i, cv2.CC_STAT_LEFT]
    y1 = degerler[i, cv2.CC_STAT_TOP]
    w = degerler[i, cv2.CC_STAT_WIDTH]
    h = degerler[i, cv2.CC_STAT_HEIGHT]

    # Kapsayıcı kutunun (Bounding box) koordinatları hesaplanıyor.
    pt1 = (x1, y1)
    pt2 = (x1+ w, y1+ h)
    (X, Y) = centroid[i]
    # Herbir blob'a kutu çiziliyor.
    cv2.rectangle(img,pt1,pt2,(0, 0, 255), 2)

cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Orijinal Görüntü:





Renk tespiti ve kutu içine alınması:

Bir görüntüdeki renkleri tespit etmek için öncelikle görüntüyü HSV (Hue-Saturation-Value) renk modeline dönüştürmemiz gerekmektedir. HSV renk modeli, renkleri "renk tonu" (hue), "doygunluk" (saturation) ve "parlaklık" (value) gibi üç temel özelliğe ayırarak temsil eder. Bu model, renkleri ayırt etmek ve renkler arasındaki benzerlikleri kolayca analiz etmek için idealdir. Ancak BGR görüntülerde de renk tespit etmek mümkündür. Ana renkler ve ara renkler farklı yöntemlerle tespit edilebilir.

Video Görüntülerinde Nesne Tespiti

Görüntü işlemede nesne tespiti, bir görüntünün içindeki nesneleri tanımak ve konumlarını belirlemek için kullanılan bir tekniktir. Bu işlem genellikle bir görüntünün içindeki nesneleri sınıflandırmak için kullanılır ve bu sınıflandırma birçok farklı amaç için kullanılabilir. Örneğin, bir güvenlik kamerasında nesne tespiti kullanılarak insanların ve araçların hareketleri takip edilebilir veya bir alışveriş merkezinde insanların nerelerde daha fazla zaman geçirdiği belirlenebilir.

Örnek çalışmamızda video görüntüsündeki belirli bir renge odaklanacağız ve o renkteki nesneleri kapsayıcı kutu içine almaya çalışacağız.

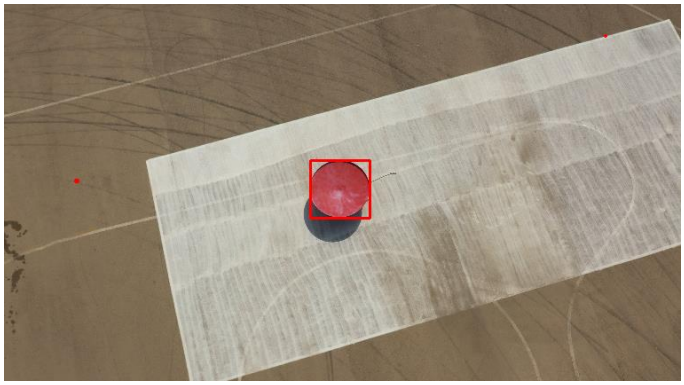
```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
import time

# Dosyadan video oku
#x = cv2.VideoCapture('ormanYanginiVideo.mp4')
x = cv2.VideoCapture('IHA_Yarismasi.MOV')
# Dosyadan Görüntü okuma başarılı olduğu sürece while döngüsü çalışsın.
while(x.isOpened()):
    # Videodan bir çerçeve oku.
    ret, frame = x.read()
    frame=cv2.resize(frame,(768,432))
    xGri=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    xred=frame[:,:,2]
    kirmiziRenk=cv2.subtract(xred,xGri)
    z,xIkili=cv2.threshold(kirmiziRenk,30,255,cv2.THRESH_BINARY)
    #  Bileşen (blob) analiz işlevini uygulayın
    (toplamBlob, etiket_id, degerler, centroid)  = cv2.connectedComponentsWithStats(xIkili, 4, cv2.CV_32S)
```

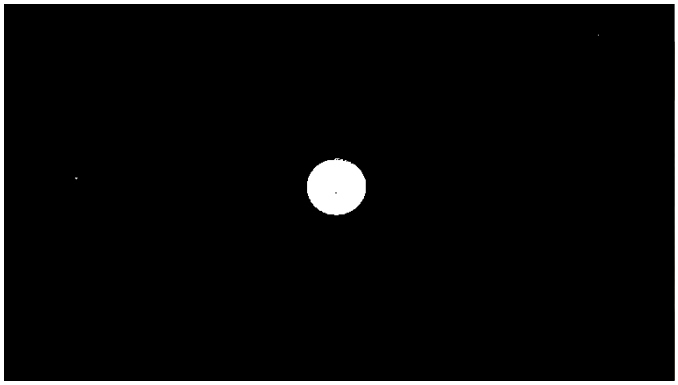
Kırmızı renk tespiti

Resimdeki kırmızı renkli nesnelerin blob'ları seçiliyor.

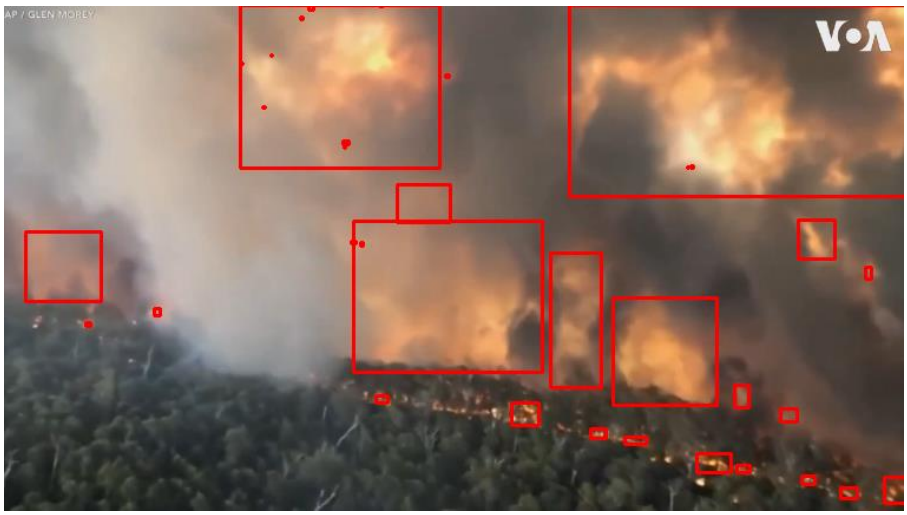
```
    # Her bir blob için işlem yapılıyor.
    for i in range(1, toplamBlob):
        # Blobların alanları
        area = degerler[i, cv2.CC_STAT_AREA]
        print(area)
        # Her bir blobun sol üst köşe, en ve boy koordinatları alınıyor.
        x1 = degerler[i, cv2.CC_STAT_LEFT]
        y1 = degerler[i, cv2.CC_STAT_TOP]
        w = degerler[i, cv2.CC_STAT_WIDTH]
        h = degerler[i, cv2.CC_STAT_HEIGHT]

        # Kapsayıcı kutunun (Bounding box) koordinatları hesaplanıyor.
        pt1 = (x1, y1)
        pt2 = (x1+ w, y1+ h)
        (X, Y) = centroid[i]
        # Herbir blob'a kutu çiziliyor.
        cv2.rectangle(frame,pt1,pt2,(0, 0, 255), 2)

    # Ekrandan görüntüleme
    cv2.imshow('frame',frame)
    cv2.imshow('Ikili',xIkili)
    #istenirse video görüntüleme yavaşlatılabilir.
    time.sleep(0.1)

    # q tuşuna basıldığında çık.
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# İşin bittikten sonra her şeyi serbest bırak.
x.release()
cv2.destroyAllWindows()
```

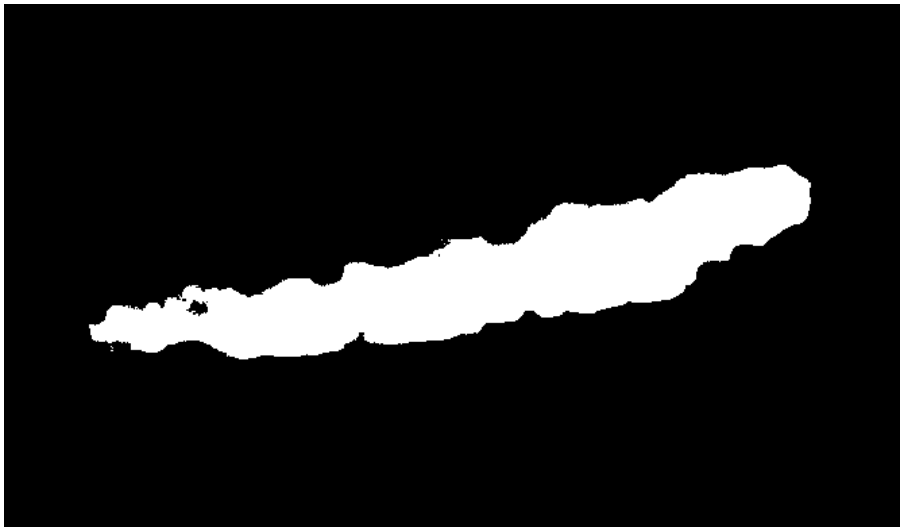
Sonuç1: (Yarışma Hedef Görüntüsü)





Sonuç2: (Orman yangını videosu)





Video Yazma

OpenCV'deki cv2.VideoWriter() işlevi, video karelerini bir dosyaya veya kameraya yazmak için kullanılabilen bir VideoWriter nesnesi oluşturmak için kullanılır. Bu, video düzenleme, nesne izleme ve gözetim gibi bilgisayarla görme uygulamalarında yaygın bir görevdir.

Bir VideoWriter nesnesi oluşturmak için aşağıdaki parametreleri belirtmeniz gerekir:

- Çıktı dosyası yolu ve adı (bir dosyaya yazılıyorsa)
- Video codec bileşenini belirtmek için kullanılan 4 baytlık bir kod olan FourCC kodu
- Videonun kare hızı (saniye başına kare olarak)
- Video karelerinin boyutu (piksel cinsinden)

Örneğin, saniyede 80 kare hızı ve 640x480 piksel kare boyutuyla "output.mp4" adlı bir dosyaya yazmak üzere bir VideoWriter nesnesi oluşturmak için aşağıdaki kodu kullanırsınız:

```
fourCC=cv2.VideoWriter_fourcc(*'XVID')
yaz = cv2.VideoWriter("output.mp4", fourCC, 80, (640, 480))
```

Video karelerini yazmayı bitirdiğinizde VideoWriter nesnesinin serbest bırakılması gerektiğine dikkat etmek önemlidir. Bu, nesne üzerinde release() yöntemi çağrılarak yapılır.

Genel olarak, OpenCV'deki cv2.VideoWriter() işlevi, video karelerini bir dosyaya veya kameraya yazmak için güçlü bir araçtır. Çok çeşitli bilgisayarla görme uygulamalarına kolay entegrasyon sağlar.

Örnek:

```
import cv2
# Aşağıdaki parametrelerle bir VideoWriter nesnesi oluşturun:
# - Çıktı dosyası yolu ve adı
# - FourCC kodu (video kodekini belirtmek için kullanılan 4 baytlık bir kod)
# - Videonun kare hızı
# - Video karelerinin boyutu

#VideoCapture nesnesi kullanarak video çerçevelerini okuyun
x = cv2.VideoCapture('daria_walk.avi')
ret, frame = x.read()
h, w, _= frame.shape # Genişlik ve yükseklik elde etmek için çerçeveyi kullanın

frameTime = 100
fourcc = cv2.VideoWriter_fourcc(*"XVID") # XVID herhangi bir şekilde değiştirilebilir
fps = 1000 / frameTime # fps oranını hesapla
# Video yazma nesnesi
yaz = cv2.VideoWriter("Pothole testing 2.mp4", fourcc, fps, (w, h))

while ret:
    frame=frame+40
    yaz.write(frame)
    cv2.imshow("Frame", frame)
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
    ret, frame=x.read()
# VideoCapture ve VideoWriter nesnelerini serbest bırakın
x.release()
yaz.release()
# Herhangi bir açık pencereyi yok edin (isteğe bağlı)
cv2.destroyAllWindows()
```

Kameradan Canlı Görüntü Kaydetme

Videodan yazma işleminden tek farkı görüntü kaynağının PC'nin entegre kamerası ya da USB'den bağlı kamera olmasıdır.

```
import cv2

cap = cv2.VideoCapture(0)
ret, frame = cap.read() #Bir çerçeve oku

h, w, _ = frame.shape # Çerçevenin en ve boyunu çıkart

frameTime = 100

fourcc = cv2.VideoWriter_fourcc(*"XVID") # Video kodeklerini ayarla
fps = 1000 / frameTime # Calculate fps
writer = cv2.VideoWriter("KameraVideosu.mp4", fourcc, fps, (w, h)) # Video yazma nesnesini oluştur

while ret: # Videonun sonunu belirlemek için ret değişkenini kullan
    frame[:,:,2]=0
    writer.write(frame) # Çerçeveyi yaz
    cv2.imshow("frame", frame)
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
    ret, frame = cap.read()

writer.release()
cap.release()
cv2.destroyAllWindows()
```

Kameradan Alınan Görüntüleri İşleme:

Görüntü işleme işlemi, görüntüler üzerinde değişiklikler yaparak veya görüntülerden bilgi toplamak için yapılabilir. Örneğin, görüntülerdeki nesneleri tanımlama, görüntülerdeki renkleri değiştirme veya görüntülerdeki nesnelerin konumunu belirleme gibi işlemler yapılabilir. Görüntü işleme için birçok farklı yöntem ve algoritma mevcuttur. Bu yöntemler ve algoritmalar, görüntüler üzerinde değişiklikler yapmak veya görüntülerden bilgi toplamak için kullanılabilir. Örneğin, filtreler, görüntüler üzerinde renk değişiklikleri yapmak için kullanılabilir. Benzer şekilde, öğrenme algoritmaları, görüntülerdeki nesneleri tanımlama gibi işlemler için kullanılabilir.

Örnek:

```python
#Kamera Kırmızı Nesne Bulma
import cv2

# Kamera örüntülerini okumaya başkla
x = cv2.VideoCapture(0)

while(x.isOpened()):
    # Videodan bir çerçeve oku.
    ret, frame = x.read()
    frame=cv2.resize(frame,(768,432))
    xGri=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    xred=frame[:,:,0]
    kirmiziRenk=cv2.subtract(xred,xGri)
    z,xIkili=cv2.threshold(kirmiziRenk,70,255,cv2.THRESH_BINARY)
    #  Bileşen (blob) analiz işlevini uygulayın
    (toplamBlob, etiket_id, degerler, centroid)  = cv2.connectedComponentsWithStats(xIkili, 4, cv2.CV_32S)
    # Her bir blob için işlem yapılıyor.
    for i in range(1, toplamBlob):
        # Blobların alanları
        area = degerler[i, cv2.CC_STAT_AREA]
        print(area)
        # Her bir blobun sol üst köşe, en ve boy koordinatları alınıyor.
        x1 = degerler[i, cv2.CC_STAT_LEFT]
        y1 = degerler[i, cv2.CC_STAT_TOP]
        w = degerler[i, cv2.CC_STAT_WIDTH]
        h = degerler[i, cv2.CC_STAT_HEIGHT]
        # Kapsayıcı kutunun (Bounding box) koordinatları hesaplanıyor.
        pt1 = (x1, y1)
        pt2 = (x1+ w, y1+ h)
        (X, Y) = centroid[i]
        # Herbir blob'a kutu çiziliyor.
        cv2.rectangle(frame,pt1,pt2,(0, 0, 255), 2)
```

```python
    # Ekrandan görüntüleme
    cv2.imshow('frame',frame)
    cv2.imshow('Ikili',xIkili)

    # q tuşuna basıldığında çık.
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# İşin bittikten sonra her şeyi serbest bırak.
x.release()
cv2.destroyAllWindows()
```

KONTUR BULMA:

Konturlar basitçe, aynı renk veya yoğunluğa sahip tüm sürekli noktaları (sınır boyunca) birleştiren bir eğri olarak açıklanabilir. Konturlar, şekil analizi ve nesne algılama ve tanıma için kullanışlı bir araçtır.

Daha iyi doğruluk için ikili görüntüler kullanın.

Bu yüzden konturları bulmadan önce, eşik veya canny kenar algılama uygulayın.

OpenCV'de konturları bulmak, siyah arka plandan beyaz nesneyi bulmak gibidir. Bu nedenle, bulunacak nesnenin beyaz ve arka planın siyah olması gerektiğini unutmayın.
Örnek:

```
import numpy as np
import cv2 as cv
im = cv.imread('test.jpg')
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
```

cv.findContours() fonksiyonunda üç argüman vardır,

- birincisi kaynak görüntü,
- ikincisi kontur alma modu,
- üçüncüsü kontur yaklaştırma yöntemidir.

Ayrıca, **konturları** ve **hiyerarşiyi** çıktı olarak verir. Konturlar, görüntüdeki tüm konturların bir Python listesidir. Her bir kontur, nesnenin sınır noktalarının (x,y) koordinatlarının bir Numpy dizisidir.

print(contours[0]) ile içeriğini görebiliriz. CHAIN_APPROX_NONE kesinlikle tüm kontur noktalarını saklar. CHAIN_APPROX_SIMPLE olarak ayarlanırsa yatay, dikey ve diyagonal segmentleri sıkıştırır ve yalnızca uç noktalarını bırakır.

```
import cv2
img = cv2.imread('blobs1.jpg')
cv2.imshow("orjinal", img)
xred=img[:,:,2]
# Gri seviye dönüşümü
gray_img = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)

kirmiziRenk=cv2.subtract(xred,gray_img)
cv2.imshow("gri", kirmiziRenk)
z,xIkili=cv2.threshold(kirmiziRenk,70,255,cv2.THRESH_BINARY)
cv2.imshow("Binary", xIkili)

contours, hierarchy = cv2.findContours(xIkili,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
for i in range(0, len(contours)):
    img = cv2.drawContours(img, contours[i], -1, (0,255,0), 3)
cv2.imshow("Frame", img)
# cv2.waitKey()
# cv2.destroyAllWindows()
```

Yukarıdaki örnekte, imgede bulunan kırmızı renkli tüm nesnelerin konturunu döngü içinde çizen bir program bulunmaktadır.

Tüm konturlar döngü içinde çiziliyor. Her konturun bir indisi var. “i” indis numarasını temsil ediyor.

```
for i in range(0, len(contours)):
    img = cv2.drawContours(img, contours[i], -1, (0,255,0), 3)
```

Kontur kalınlığı “3” olacak

Konturun tamamını çizeceğimizi gösterir.

Kontur yeşil renkte olacak.