SAC RACE

SAC AUTONOMOUS CAR RACE

Görüntü İşleme
Eğitim Kitapçığı

(17 Ağustos 2019 tarihinde Güncellenmiştir)

Görüntü İşleme Nedir?

Görüntü işleme kısaca dijital formdaki bir görüntünün analiz, dönüşüm ve oluşturulmasını içeren bilgisayar bilimi alanıdır. Görüntü işleme sayesinde kameralardan alınan veriler analiz edilebilir. Görüntü işleme birçok işlem için kullanılabilir. Bunlardan bazıları güvenlik kameraları, plaka tanıma ve tabiki otonom araçlardır.

Kamera silahı tespit ediyor.

İstanbul Serbest Geçiş Sistemi

Otonom Araç Şerit Takibi

Görüntü İşleme Kütüphaneleri

Görüntü işleme için neredeyse tüm yazılım dilleri kullanılabilir ve bu yazılım dilleri için yazılmış kütüphaneler vardır. Bu kütüphaneler görüntüdeki her piksel ile uğraşmak yerine daha genel komutlar vermemizi sağlar. Bu kütüphanelerden bazıları PIL/Pillow, SimpleCV ve OpenCV'dir. Açık kaynaklı olan bu kütüphanelerden en gelişmişi, en çok farklı işlem için kullanılabilir olanı ve en canlı komüniteye sahip olan OpenCV'dir. Bu nedenle OpenCV diğer kütüphanelere karşı öndedir. Biz de bu nedenle OpenCV'yi kullanacağız.

OpenCV

OpenCV kütüphanesi en geniş çaplı açık kaynaklı görüntü işleme kütüphanesidir. 2500'ün üzerinde optimize algoritma kullanılabilir durumdadır. 47 bin kişinin üzerinde bir komüniteye sahip kütüphane yaklaşık 18 milyon kez indirilmiştir. OpenCV Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda ve Toyota başta olmak üzere birçok şirket tarafından desteklenmekte ve kullanılmaktadır. C++, Python, Java ve MATLAB'i resmî olarak destekleyen OpenCV Windows, Linux, MacOS ve Android işletim sistemleriyle uyumludur.Bu diller haricindeki dillerde kullanmak için de birçok aracı sistem mevcuttur. Biz OpenCV'yi Python ile kullanacağız.

OpenCV Kurulumu

OpenCV'yi farklı işletim sistemlerinde ve yazılım dillerinde kurmak için farklı yöntemler bulunmaktadır. OpenCV'yi Python'a kurmak için en kullanışlı yöntem pip package installer'ı kullanmaktır.

Terminalde aşağıdaki komut ile openCV'yi kurabilirsiniz.

```
pip install opencv-python
```

SAC Race için büyük ihtimal ihtiyacınız olmayacak olsa da aşağıdaki komut ile OpenCV'yi ekstra modülleri ile birlikte kurabilirsiniz.

```
pip install opencv-contrib-python
```

Not: Kurulumu SAC Racer aracına yapmanıza gerek olmayacaktır. Araca yükleyeceğiniz SAC Racer OS image'ında bulunmaktadır. Kendi bilgisayarlarınıza yükleyiniz.

OpenCV'yi Python Script'ine İlave Etmek

OpenCV diğer tüm kütüphaneler gibi `import` komutuyla python'da kullanılır.

```
import cv2
import numpy
```

openCV'nin yanında numpy'ı da import etmeniz birçok işlem için gerekebilir.

Bilgisayardaki Bir Resmi Kullanmak

Bir resmi OpenCV ile yüklemek için `imread()` fonksiyonu kullanılır.

```
img = cv2.imread("PATH/TO/IMAGE")
```

Bu satırla resimi `img` değişkenine bir piksel dizisi (array) olarak atadık. Burada `"PATH/TO/IMAGE"` kullanacağınız resmin bilgisayardaki yerini gösteren yoldur. Bu yolu yazarken olabildiğince relative* değil absolute** path kullanmanızı tavsiye ederiz. Relative path kullandığınızda terminali açtığınız lokasyon farklı olduğunda hata alırsınız.
Relative path*: `./images/image1.png`
Absolute path**: `/Users/cihan/Desktop/images/image1.png`

Resim Göstermek

Görüntü işleme gibi birçok durumda direkt görülebilir sonuçlar oluşturan bir alan ile uğraştığımız için resmi bir işlem yaptıktan sonra ekranda görmek debugging için iyi bir yöntemdir. Resmi göstermek için imshow komutu kullanılır:

```
cv2.imshow("name", img)
cv2.waitKey(0)
```

Burada `"name"` resmi göstermek için açılacak pencerenin ismidir. Aynı ismi kullanarak başka bir resim ile imshow yaparsanız resim aynı penerede açılır. Yani önceki resim ekrandan kaybolur. İsmi değiştirerek imshow yaptığınızda ikinci bir pencere açılır ve iki resmi de ekranda bulundurabilirsiniz.

`waitKey` fonksiyonu resmin belirli bir süre ekranda kalması içindir. Kullanılmazsa kod devam eder ve resim gösterilmeden kapanır. Parametre olarak 0 kullandığınızda boşluk tuşuna basana kadar resim ekranda kalır. Diğer sayılar kaç milisaniye bekleyeceğini belirtir. (0 ve pozitif sayılar kullanılabilir.)

Kameradan Canlı Görüntü Almak

Kameradan canlı video almak için VideoCapture metodu kullanılır.

```
vcap = cv2.VideoCapture(0)

while True:
      img, ret = vcap.read()

      # Videoyu canlı olarak göstermek için
      cv2.imshow("name", img)
      cv2.waitKey(0)
```

VideoCapture'daki 0 parametresi 0 nolu kameradan (laptop için bilgisayarın içindeki kamera) görüntü almak istediğimi belirtiyor. Birden çok kamera bağlıysa sırayla 1, 2 gibi atanıyorlar.

Resim Kaydetmek

Resimleri imwrite ile kaydedebilirsiniz.

```
cv2.imwrite("PATH/TO/WRITE", img)
```

Basit Resim İşlemleri

Resmi kırpmak için aslında numpy dizisi olması özelliğini kullanıyoruz.

```
img = cv2.resize(img, (250, 250)) # resim boyutunu değiştiriyoruz.
img = img[0:200, 0:100] #Bu satır ile resmin y ekseninde ilk 200, x
ekseninde ilk 100 pikseli kalacak şekilde kırpıyoruz.
```

```
yhalf = img.shape[1]/2
Xhalf = img.shape[0]/2
img = img[0: yhalf, 0:xhalf]
```

Yukarıdaki kod parçası ile resim sol üst çeyreği kalacak şekilde kırpılmaktadır. Burada img.shape resmin boyutlarını veren bir tuple'dır. Img.shape[0] genişlik, 1 yükseklik, 2 de renktir.

Bulanıklaştırma

Aşağıda bulanıklaştırma (blur) için kullanabileceğiniz birkaç yöntem verilmiştir.

```
blur = cv2.blur(img,(5,5))
```

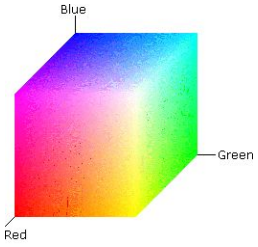
```
blur = cv2.GaussianBlur(img,(5,5),0)
```

```
blur = cv2.medianBlur(img,5)
```

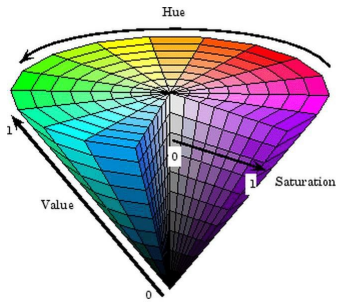
Bu örneklerde (5,5) ve 5 resmin hem x, hem y ekseninde 5 piksellik kerneller ile bulanıklaşacağını belirtir. Bu sayıyı büyüterek daha fazla bulanıklaşmasını sağlarsınız. Çalışma prensipleri ve detayları için şu dökümana bakabilirsiniz: https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html

Renk Uzayları

Renk uzayı dijital olarak her rengin tek bir nokta olarak temsil edildiği bir koordinat sistemidir. Örneğin her rengin kırmızı, yeşil ve mavi ile ifade edilebileceğini bilirsiniz. Bu mantığa dayalı olarak oluşturulan RGB (Red, Green, Blue) renk uzayında x,y,z eksenleri yerine bu üç rengin geçtiği ve bu renkleri arttırıp azaltarak renkler elde edilen bir koordinat sistemidir.



Çok kullandığımız başka bir renk uzaayı ise HSV'dir. Hue, Saturation, Value ismindeki bu uzayı görüntü işlemede kullanmak çok daha kolaydır. Hue renk değerini ifade eder. Saturation, beyaz ile renkli arasında, Value, siyah ile renkli arasındadır. Bu sayede belli bir rengi veya tonu bulmak çok daha kolay olur. Bu renk uzayı ise küp yerine bir kon şeklinde ifade edilir. Konide Hue açı değeri, Saturation içerden dışarı doğru bir değer ve value derinliktir.



OpenCV'yi kullanarak resmin hangi renk uzayı kullanılarak ifade edildiğini değiştirebilirsiniz. Bu işlem için `cvtColor()` fonksiyonu kullanılır.

```
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

Burada `cv2.COLOR_BGR2HSV` hangi renk uzayları arasında dönüşüm yapıldığını belirler. Bu örnekte en çok kullanacağınızı düşündüğüm BGR'dan (RGB'nin birçok bilgisayarda kullanılan şekli) HSV'ye dönüşüm gösterilmektedir.

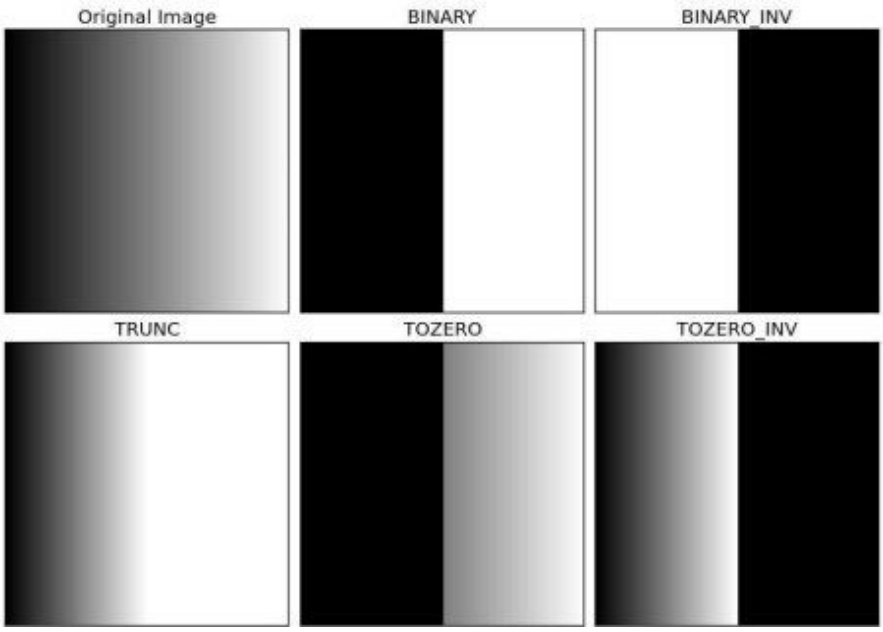
Diğer dönüşümler için şu dökümana bakabilirsiniz:
https://docs.opencv.org/trunk/d8/d01/group__imgproc__color__conversions.html

Thresholding

Belli bir değerin üstünde veya altındaki piksellere belli bir işlem yapan fonksiyonlar thresholding olarak adlandırılır. Threshold işlemi OpenCV ile çok kolay bir şekilde yapılabilir.

```
ret, thresh = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
```

BINARY yerine kullanabileceğiniz diğer yöntemler ve sonuçları için aşağıdaki tabloyu inceleyin.



Renk Tespiti

SAC Race yazılımlarında en çok yapacağınız OpenCV işlerinden biri renk tespitidir. Renk tespiti için aslında ana yöntem doğru renk uzayını seçmek ve renk verilerine threshold uygulamaktır. Bu işlem için inRange fonksiyonunu kullanmaktayız.

Renk tespiti yapmak için iki eşik değeri (alt ve üst) belirlemek gerekiyor. Örneğin HSV'de çalışıyorsak ve yeşil rengi tespit etmek istiyorsak bu eşikler şu şekilde olabilir:

```
lower = np.array([50,40,35])
upper = np.array([85,255,255])
```

HSV için ilk değer 0 - 180, diğer ikisi 0 - 255 arasında olacak şekilde kullanılabilecek tüm renkleri kapsar. Dikkat edilmelidir ki internetteki bir RGB to HSV dönüştürücü kullanırsanız değerler 0-360, 0-100, 0-100 arasında olacaktır. Daha sonra bu değerleri kullanarak inRange() fonksiyonu çalıştırılır.

```
mask = cv2.inRange(hsv, lower, upper)
```

Mask, sadece iki eşik arasında kalan renkteki alanların tamamen beyaz, kalan yerlerin tamamen siyah olduğu bir resimdir.

Renk ile Obje Tespiti (Contours)

Renk tespiti ile istediğimiz alanlardaki objeyi bulduk ancak bu objenin lokasyonunu ve bazı özelliklerini otonom olarak belirleyebilmek için birkaç işlem daha yapmamız gerekiyor.

Bu işlemlerden önemli bir tanesi kontür işlemleri. Kontür işlemlerini kullanarak rengini bulduğunuz objenin etrafını tam olarak kaplayan dikdörtgen veya daire şeklini bulabilirsiniz.

```
_, contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
```

Yukarıdaki satırı kullanarak renk tespiti ile oluşturduğunuz beyaz şekillerin etrafını kaplayan noktalar oluşturuyoruz.

```
areas = [cv2.contourArea(c) for c in contours]
max_index = np.argmax(areas)
cnt=contours[max_index]
```

Yukarıdaki kod parçası ile contours'dan en büyük alan kaplayanını buluyoruz. Bazen renk tespiti ile oluşturduğumuz maskede hatalar olabiliyor ve istemediğimiz alanlarda küçük beyaz alanlar oluşuyor. En büyüğünü bularak bu sorunu çözüyoruz.

Yukarıda kullandığımız işlemlerden biri dikkatinizi çekecektir. Contours bir dizi ve contours[0] ile ilk elemanına ulaşalım. Yukarıda da kullandığımız contour'ün alanını bulan fonksiyon şu şekilde:

```
c = contours[0]
area = cv2.contourArea(c)
```

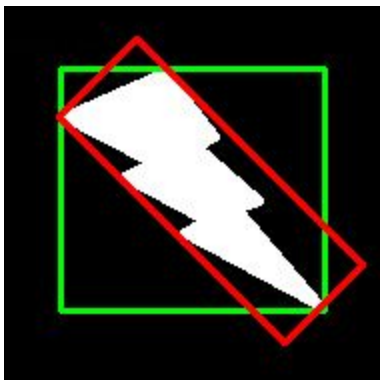
Bulduğumuz "en büyük" kontür ile bir çok işlem yapabiliriz. Bunlardan en çok kullanılanlara göz atalım.

```
x,y,w,h = cv2.boundingRect(cnt)
img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

Yukarıdaki kod parçasıyla bulduğumuz objeyi kaplayan düz bir dikdörtgen bulup onu yeşil renkle çizdik.

```
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
im = cv2.drawContours(im,[box],0,(0,0,255),2)
```

Yukarıdaki kod parçasıyla objeyi kapsayan en küçük dikdörtgeni bulup çizebiliriz. Bu örneklerin çıktılarına bakalım:



Bu kod parçasıyla objeyi kaplayan en küçük daireyi çizebiliriz:

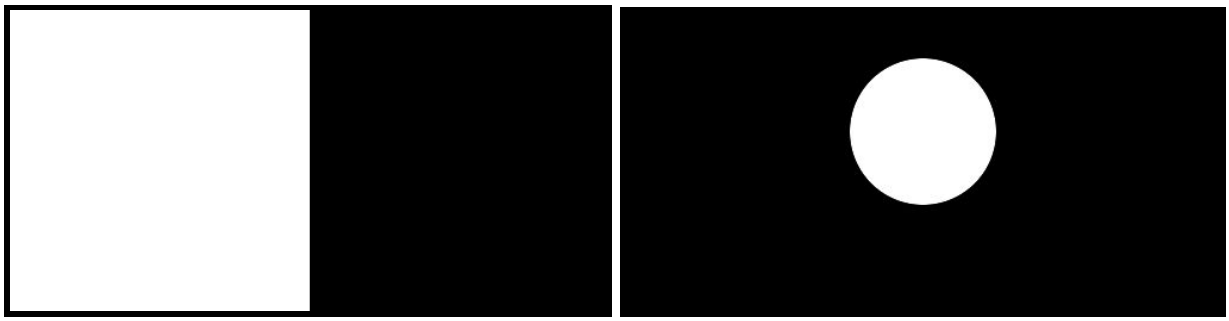
```
(x,y),radius = cv2.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
img = cv2.circle(img,center,radius,(0,255,0),2)
```

Görelim:

Diğer benzer işlemleri bu internet sitesinden görebilirsiniz:
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html

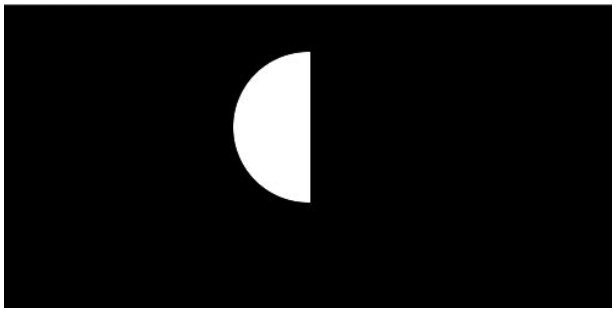
Bitwise İşlemler

Bu işlemler aslında mantık operatörlerinin siyah beyaz resimler üzerine uygulanmasıdır. Bu bölümde şu iki resmi kullanacağız:



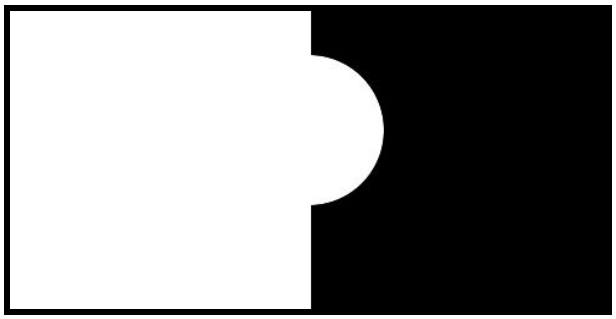
```
bit_and = cv2.bitwise_and(img2, img1, mask=None)
```

Bitwise and işlemi iki resimde de beyaz olan kısımları beyaz bırakarak kalan kısımları siyah yapar. Mantıktaki AND operatörü gibi, 1 ve 1 = 1, 1 ve 0 = 0, 0 ve 0 = 0. Sonuç şu şekilde olacaktır:



```
bit_or = cv2.bitwise_or(img2, img1, mask = None)
```

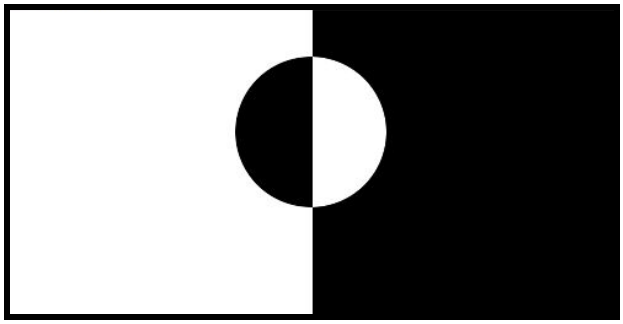
Bitwise or işlemi iki resimden birinde beyaz ise o pikseli beyaz bırakır. Mantıktaki OR gibi, 1 veya 1 = 1, 1 veya 0 = 1, 0 veya 1 = 1, 0 veya 0 = 0. Sonuç şu şekilde olacaktır:



* Etrafına resmin görülebilmesi için sonradan siyah çizgi koyulmuştur.

```
bit_xor = cv2.bitwise_xor(img1, img2, mask = None)
```

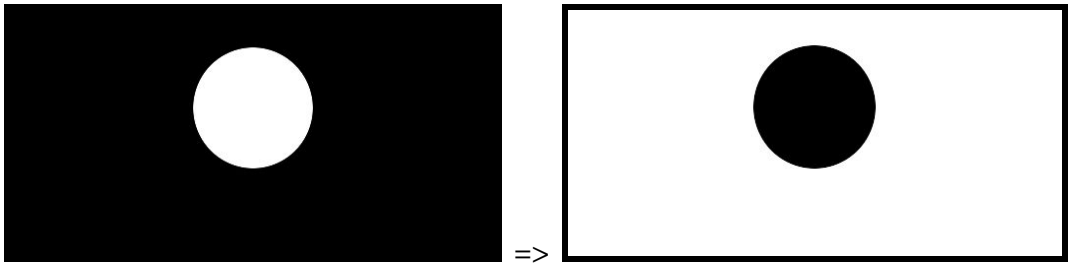
Bitwise xor işleminde 1 xor 1 = 0, 1 xor 0 = 1, 0 xor 1 = 1, 0 xor 0 = 0.



* Etrafına resmin görülebilmesi için sonradan siyah çizgi koyulmuştur.

```
bit_not = cv2.bitwise_not(img2, mask = None)
```

Bitwise not işlemi 0 ve 1'leri, siyah ve beyazı yer değiştirir. Tek resim ile uygulanır.



* Etrafına resmin görülebilmesi için sonradan siyah çizgi koyulmuştur.

Daha fazla bitwise bilgisi için:
https://www.geeksforgeeks.org/arithmetic-operations-on-images-using-opencv-set-2-bitwise-operations-on-binary-images/

Perspektif Dönüşümü

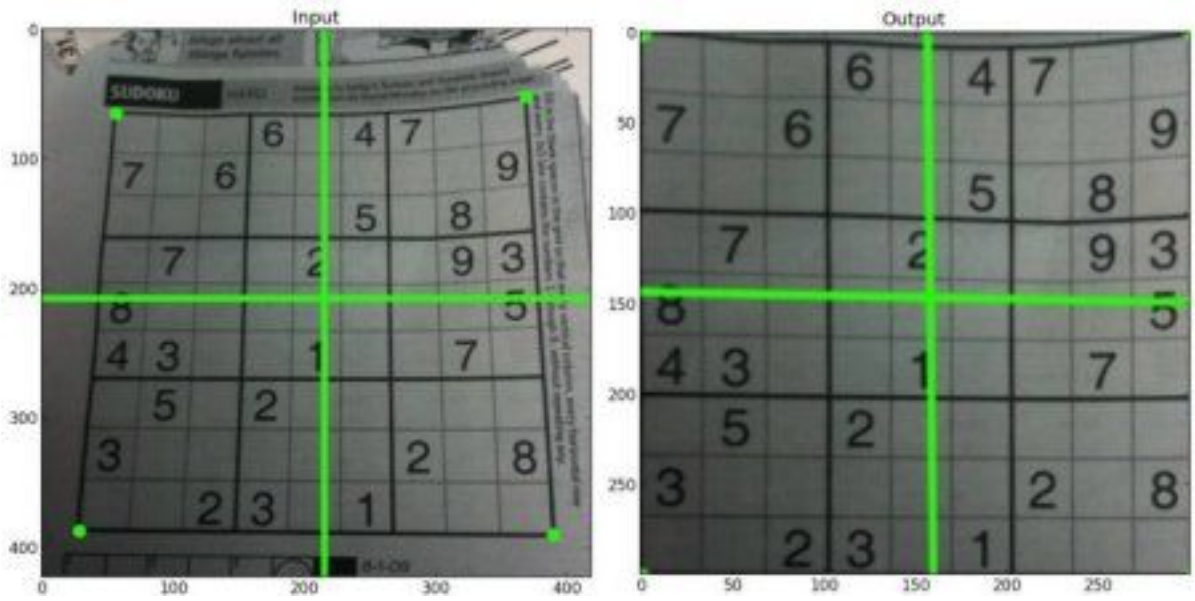
Perspektif dönüşümü resmi sanki farklı bir açıdan bakıyormuş gibi dönüştürmenizi sağlar. Bu işlem için resimden 4 nokta ve yeni resimde hangi noktalara geleceği seçilir.

```
pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])
```

Daha sonra aşağıdaki işlemler ile dönüştürülmüş resim elde edilir. Burada (300,300) yeni resmin boyutudur.

```
M = cv.getPerspectiveTransform(pts1,pts2)
img = cv.warpPerspective(img,M,(300,300))
```

Sonuç aşağıdaki gibidir:



İlk resim (Yeşil dört nokta seçilmiştir) Değiştirilmiş resim

Daha Fazla OpenCV için Bu Sayfaları Kontrol Edin

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html

Bu Dökümanı Yazan: Cihan Alperen Bosnalı (cihanbosnali@gmail.com)

Okuduğunuz İçin Teşekkürler!

SAC RACE
SAC AUTONOMOUS CAR RACE