Kapsamlı OpenCV-Python Sunumu

Bilgisayarlı Görü Dünyasına Giriş

Bilgisayarlı Görü İş Akışı

01

Veri Girişi & GUI

Görüntü ve videoların sisteme alınması, kullanıcıya görsel geri bildirim sağlanması.

`imread, imshow, putText`

02

Ön İşleme

Gürültü giderme, renk uzayı dönüşümü ve eşikleme ile verinin analize hazırlanması.

`cvtColor, blur, threshold`

03

Özellik Çıkarımı

Görüntüdeki anlamlı yapıların (kenar, köşe, kontur) matematiksel olarak tespiti.

`Canny, findContours, ORB`

04

Analiz & Karar

Tespit edilen özelliklerin sınıflandırılması, takibi ve anlamlandırılması.

`Cascade, Video Analysis`

Kritik Fonksiyonlar: Kullanım Senaryoları

Fonksiyon	Aşama	Neden Kullanılır?	Örnek Senaryo
`cv.cvtColor()`	ÖN İŞLEME	Veri boyutunu azaltmak ve analize hazırlamak.	Renkli resmi griye çevirip kenar algılamaya hazırlama.
`cv.threshold()`	SEGMENTASYON	Nesneyi arka plandan ayırmak (Maske oluşturma).	Beyaz bir kağıt üzerindeki siyah yazıları ayıklama.
`cv.GaussianBlur()`	FİLTRELEME	Gürültüyü azaltmak ve detayları yumuşatmak.	Kenar algılama öncesi hatalı çizgileri engelleme.
`cv.Canny()`	ÖZELLİK ÇIKARIMI	Nesne sınırlarını en hassas şekilde bulmak.	Bir arabanın dış hatlarını tespit etme.
`cv.findContours()`	ANALİZ	Kenarları kapalı şekillere dönüştürüp gruplamak.	Görüntüdeki nesneleri sayma ve alanlarını hesaplama.
`cv.ORB_create()`	EŞLEŞTİRME	Benzersiz noktaları bulup farklı resimlerde arama.	Bir logoyu karmaşık bir sahnede bulup takip etme.

OpenCV Mimari Akış Şeması

** Her aşama bir önceki aşamanın çıktısını girdi olarak kullanır. Bu yapı "Computer Vision Pipeline" olarak adlandırılır.*

Endüstriyel Uygulama: Otonom Araçlar

01 **Giriş (Data Acquisition)**

Aracın önündeki yüksek çözünürlüklü kameralardan saniyede 60 kare video verisi alınır. → cv.VideoCapture

02 **Ön İşleme (Pre-processing)**

Yağmur veya gece sürüşündeki gürültüler temizlenir, şerit çizgilerini belirginleştirmek için gri tonlamaya geçilir. → cv.GaussianBlur, cv.cvtColor

03 **Segmentasyon & Özellik Çıkarımı**

Yol yüzeyi ile kaldırımlar ayrılır (Maskeleme). Şerit çizgileri ve trafik tabelalarının sınırları tespit edilir. → cv.threshold, cv.Canny

04 **Analiz & Karar (Decision Making)**

Şeritlerin eğimine göre direksiyon açısı hesaplanır. Yoldaki yayalar veya araçlar tespit edilerek fren kararı verilir. → cv.findContours, CascadeClassifier

GUI Özellikleri & Görselleştirme

`cv.imread()`

Görüntü verisini diskten belleğe (NumPy dizisi olarak) yükler.

`cv.imshow()`

Görüntüyü bir pencerede görüntüler. `cv.waitKey()` ile birlikte kullanılır.

`cv.imwrite()`

İşlenmiş görüntü verisini belirtilen formatta diske kaydeder.

`cv.putText()`

Görüntü üzerine metin, koordinat ve font bilgisiyle yazı ekler.

```
# Temel GUI Akışı
img = cv.imread('input.jpg')
cv.putText(img, 'OpenCV', (50,50), cv.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
cv.imshow('Output', img)
cv.waitKey(0)
```

Temel İşlemler

Core Operations

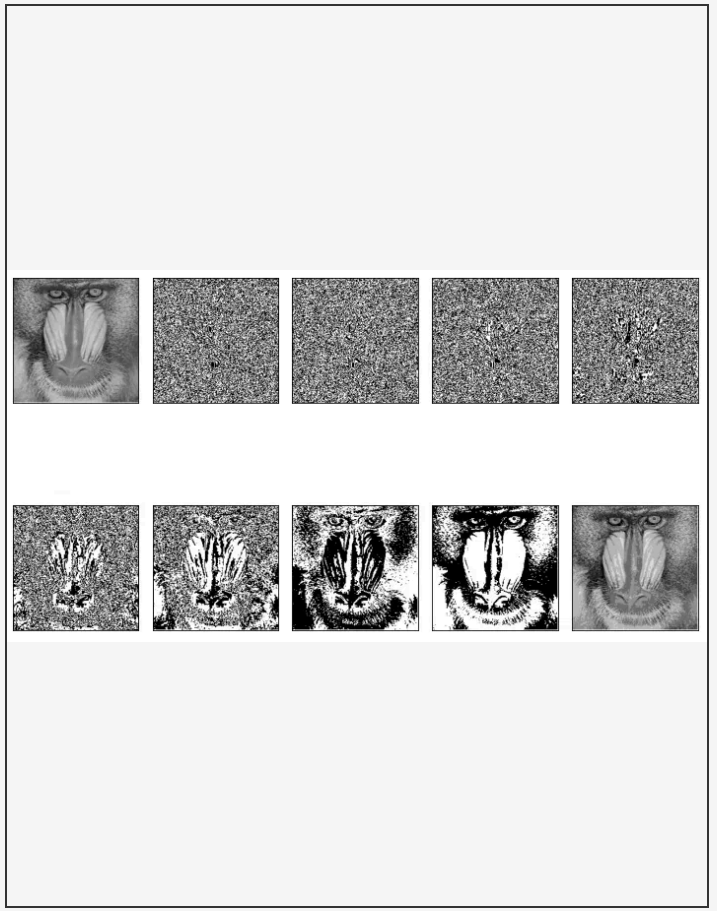
Görüntülerin aritmetik olarak toplanması, harmanlanması ve kanal yönetimi.

cv.split(): Kanallara ayırma
cv.merge(): Kanalları birleştirme

Görüntü Harmanlama

```
# İki görüntüyü harmanlama
# dst = α·img1 + β·img2 + γ

res = cv.addWeighted(
    img1, 0.7,
    img2, 0.3,
    0
)
```



Maskeleme & Bitwise İşlemleri

Bitwise AND (Ve)

Görüntü ve maskeyi çakıştırır. Sadece maskenin **beyaz (255)** olduğu pikseller korunur.

```
res = cv.bitwise_and(img, img, mask=mask)
```

Bitwise NOT (Değil)

Piksel değerlerini tersine çevirir. Beyazlar siyah, siyahlar beyaz olur. Maske tersleme için kullanılır.

```
mask_inv = cv.bitwise_not(mask)
```

Bitwise OR (Veya)

İki görüntüyü birleştirir. En az birinde piksel varsa sonuçta da olur. Şekil birleştirme için idealdir.

```
res = cv.bitwise_or(img1, img2)
```

Bitwise XOR (Özel Veya)

İki görüntü arasındaki farkları bulur. Sadece piksellerin farklı olduğu yerler beyaz kalır.

```
res = cv.bitwise_xor(img1, img2)
```

Threshold'dan Maskeye: Segmentasyon

Segmentasyon Mantığı

1. Thresholding

Görüntüyü siyah-beyaz (ikili) hale getirerek nesne sınırlarını belirleriz.

2. Maske Oluşturma

Elde edilen ikili görüntü artık bizim "maskemizdir". Beyaz alanlar "geçirgen", siyahlar "engelleyici"dir.

3. Bitwise Uygulama

Maskeyi orijinal görüntüye uygulayarak nesneyi arka plandan matematiksel olarak koparırız.

```
# 1. Maske oluştur (Threshold)
ret, mask = cv.threshold(gray, 10, 255, cv.THRESH_BINARY)

# 2. Maskeyi tersle (Arka plan için)
mask_inv = cv.bitwise_not(mask)

# 3. Nesneyi ayıkla (Bitwise AND)
img_fg = cv.bitwise_and(img, img, mask=mask)

# 4. Arka planı temizle
img_bg = cv.bitwise_and(roi, roi, mask=mask_inv)
```

Renk Uzayları & Eşikleme Analizi

Renk Uzayı Dönüşümleri

`cv.cvtColor()`

BGR → GRAY

Görüntüyü gri tonlamaya çevirir. İşlem yükünü azaltır ve kenar algılama için temel oluşturur.

`cv.cvtColor()`

BGR → HSV

Renk tabanlı nesne takibi için idealdir. Işık değişimlerinden (V kanalı) bağımsız renk analizi sağlar.

Eşikleme (Thresholding) Teknikleri

`cv.threshold()`

Global

Tüm görüntüye tek bir eşik değeri uygular. Homojen ışıkta hızlıdır.

`cv.adaptiveThreshold()`

Adaptive

Her bölge için yerel eşik hesaplar. Değişken ışıkta en iyi sonuç.

`cv.THRESH_OTSU`

Otsu

Histogramı analiz ederek en uygun eşiği otomatik belirler.

Kernel ve Konvolüsyon

Matematiksel Temel

Kernel (Çekirdek): Görüntü üzerinde kaydırılan (sliding window) küçük bir matristir. Her adımda piksellerle çarpılarak yeni bir değer üretir.

Bu işleme **Konvolüsyon** denir. Kernel içindeki değerler (ağırlıklar), işlemin türünü belirler:

- Yumuşatma (Blurring)
- Kenar Belirginleştirme (Sharpening)
- Gürültü Giderme

3x3 Gaussian Kernel Örneği

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

```
Yeni Piksel = Σ (Komşu Pikseller * Kernel Ağırlıkları)
```

Yumuşatma (Smoothing) Metotları

```
# 1. Averaging (Ortalama)
blur = cv.blur(img, (5,5))

# 2. Gaussian Blur (Gauss - En Yaygın)
g_blur = cv.GaussianBlur(img, (5,5), 0)

# 3. Median Blur (Medyan - Tuz & Biber Gürültüsü için)
m_blur = cv.medianBlur(img, 5)

# 4. Bilateral Filter (Kenar Koruyucu)
b_filter = cv.bilateralFilter(img, 9, 75, 75)
```

Metot	Kullanım Amacı	Kenar Koruma	Hız
Averaging	Genel gürültü giderme	Düşük (Bulanıklaştırır)	Çok Hızlı
Gaussian	Gauss gürültüsü giderme	Orta	Hızlı
Median	Tuz & Biber (Salt-Pepper) gürültüsü	Yüksek	Orta
Bilateral	Gürültü giderirken kenarları keskin tutma	Çok Yüksek	Yavaş

Kenar Algılama: Sobel Operatörü

Gradyan Analizi

Kenarlar, görüntüdeki piksel yoğunluğunun aniden değiştiği yerlerdir. Sobel, bu değişimi (türevi) hesaplamak için kullanılır.

```
Sobel X (Dikey Kenarlar):
[[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]

Sobel Y (Yatay Kenarlar):
[[-1, -2, -1], [0, 0, 0], [1, 2, 1]]
```

Bu operatörler görüntü üzerinde gezdirilerek X ve Y yönlü değişimler bulunur.

```python
# X yönlü gradyan (Dikey kenarlar)
sobelx = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=5)

# Y yönlü gradyan (Yatay kenarlar)
sobely = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=5)

# Gradyan büyüklüğünü birleştir
mag = cv.magnitude(sobelx, sobely)
mag = cv.convertScaleAbs(mag)
```

Kenar Algılama: Canny

`cv.Canny()`

En popüler ve hassas kenar algılama algoritmasıdır. Parametreleri:

- **threshold1:** Histerezis eşiklemedeki alt eşik değeri.
- **threshold2:** Histerezis eşiklemedeki üst eşik değeri.
- **apertureSize:** Sobel operatörü için çekirdek boyutu (Varsayılan 3).

```
# Canny Uygulaması
edges = cv.Canny(img, 100, 200, apertureSize=3)
```

Morfolojik İşlemler

Şekil Tabanlı Dönüşümler

Erozyon (Erosion)
Nesne sınırlarını aşındırır, gürültüyü yok eder.

Genişletme (Dilation)
Nesne alanını genişletir, boşlukları doldurur.

Açma & Kapama (Opening/Closing)
Gürültü giderme ve nesne birleştirme kombinasyonları.

Uygulama

```
import numpy as np

# 5×5'lik bir kernel tanımla
kernel = np.ones((5,5), np.uint8)

# Erozyon uygula
erosion = cv.erode(img, kernel,
    iterations = 1)

# Genişletme uygula
dilation = cv.dilate(img, kernel,
    iterations = 1)
```

Kontur Analizi: Kritik Fonksiyonlar

`cv.findContours()`

İkili görüntüdeki nesne sınırlarını (konturları) ve hiyerarşiyi bulur.

`cv.drawContours()`

Bulunan konturları görüntü üzerine çizer. Renk ve kalınlık ayarlanabilir.

`cv.contourArea()`

Belirli bir konturun kapladığı alanı (piksel cinsinden) hesaplar.

```
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(img, contours, -1, (0,255,0), 3)
area = cv.contourArea(contours[0])
```

Kenar Algılama (Edge Detection) Metotları

```python
# 1. Sobel (X ve Y Gradyanları)
sobelx = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=5)
sobely = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=5)

# 2. Laplacian (İkinci Türev)
laplacian = cv.Laplacian(img, cv.CV_64F)

# 3. Canny (Çok Aşamalı - En Hassas)
edges = cv.Canny(img, 100, 200)
```

Metot	Çalışma Prensibi	Gürültü Hassasiyeti	Kullanım Alanı
Sobel	Birinci türev (Gradyan)	Yüksek	Basit kenar yönü tespiti
Laplacian	İkinci türev	Çok Yüksek	Hızlı kenar tespiti (Gürültüsüz)
Canny	Gradyan + Histerezis Eşikleme	Düşük (Dahili Gauss filtresi)	Hassas nesne sınır tespiti

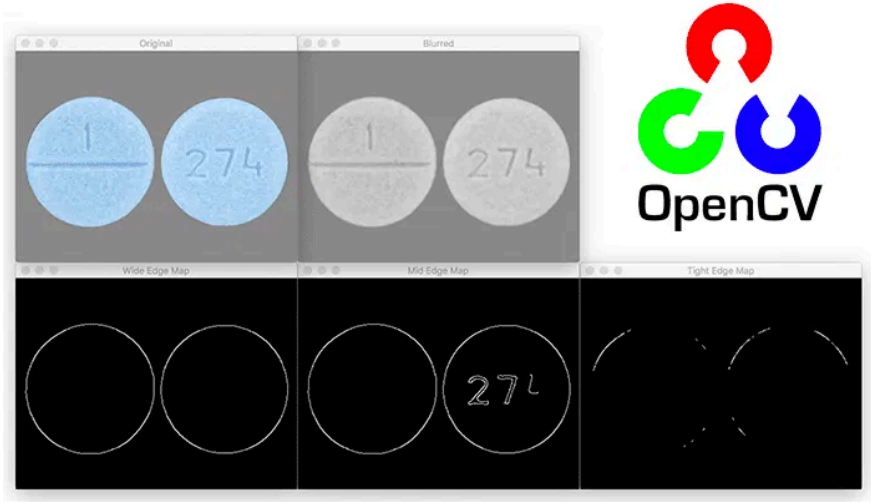
Kritik Not: Canny, gürültü giderme, gradyan hesaplama ve eşikleme adımlarını birleştirdiği için en güvenilir yöntemdir.

Konturlar (Contours)

Şekil Analizi ve Tespit

Konturlar, aynı renk veya yoğunluğa sahip tüm kesintisiz noktaları birleştiren eğrilerdir. Nesne tespiti ve şekil analizi için temel araçtır.

- **cv.findContours():** İkili görüntüden konturları bulur.
- **cv.drawContours():** Tespit edilen konturları çizer.
- **Hiyerarşi:** İç içe geçmiş nesnelerin ilişkisini tanımlar.
- **Yaklaşım:** Bellek tasarrufu için nokta sayısını optimize eder.




```python
# Konturları bul
contours, hierarchy = cv.findContours(
    thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE
)

# Tüm konturları yeşil renkte çiz
cv.drawContours(img, contours, -1, (0, 255, 0), 3)
```

Video Analizi: Arka Plan Çıkarma

`cv.createBackgroundSubtractorMOG2()`

Gaussian Mixture tabanlı arka plan çıkarıcı oluşturur. Işık değişimlerine duyarlıdır.

`cv.createBackgroundSubtractorKNN()`

K-Nearest Neighbors tabanlı arka plan çıkarıcı oluşturur. Gürültü yönetimi farklıdır.

`backSub.apply()`

Her yeni video karesi için arka plan modelini günceller ve ön plan maskesini döner.

```
backSub = cv.createBackgroundSubtractorMOG2(detectShadows=True)
fgMask = backSub.apply(frame)
```

Video Analizi: Optik Akış (Optical Flow)

Seyrek (Sparse) Akış

Lucas-Kanade: Sadece Shi-Tomasi köşeleri gibi "iyi" noktaları takip eder. Piramit yapısı ile büyük hareketleri yakalar.

```
# Lucas-Kanade Uygulaması
p1, st, err = cv.calcOpticalFlowPyrLK(
    old_gray, gray, p0, None,
    winSize=(15,15), maxLevel=2
)
```

Yoğun (Dense) Akış

Farneback: Görüntüdeki her bir piksel için hareket vektörü hesaplar. Sahnenin genel hareket yapısını anlamak için idealdir.

```
# Farneback Uygulaması
flow = cv.calcOpticalFlowFarneback(
    prev, next, None,
    0.5, 3, 15, 3, 5, 1.2, 0
)
# flow[...,0] → x yönlü hareket
# flow[...,1] → y yönlü hareket
```

Özellik Algılama (Feature Detection) Metotları

```
# 1. SIFT (Scale-Invariant Feature Transform)
sift = cv.SIFT_create()
kp, des = sift.detectAndCompute(img, None)

# 2. ORB (Oriented FAST and Rotated BRIEF)
orb = cv.ORB_create()
kp, des = orb.detectAndCompute(img, None)
```

Algoritma	Hız	Ölçek Değişmezliği	Patent Durumu
SIFT	Yavaş	Çok İyi	Ücretsiz (Eskiden Patentliydi)
SURF	Orta	İyi	Patentli (Dikkat!)
ORB	Çok Hızlı	Orta	Ücretsiz (Açık Kaynak)

Kritik Not: Gerçek zamanlı video işleme ve mobil cihazlar için **ORB** en verimli seçenektir.

Nesne Algılama

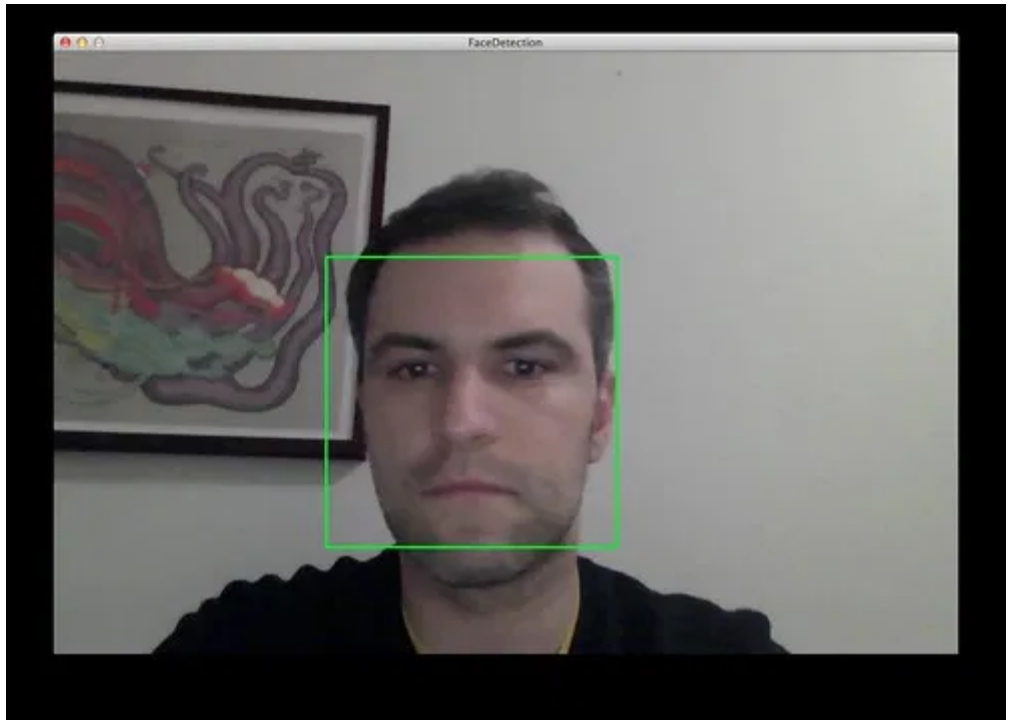
Haar Cascade Sınıflandırıcıları

Makine öğrenimi tabanlı bir yaklaşımdır. Önceden eğitilmiş XML dosyaları kullanılarak yüz, göz ve gülümseme gibi nesneler gerçek zamanlı olarak tespit edilebilir.

```
# Sınıflandırıcıyı yükle
face_cascade = cv.CascadeClassifier('face.xml')

# Çok ölçekli algılama yap
faces = face_cascade.detectMultiScale(
    gray,
    scaleFactor=1.3,
    minNeighbors=5
)

# Tespit edilen yüzleri kutu içine al
for (x,y,w,h) in faces:
    cv.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
```



Video Analizi: Derinlemesine Bakış

Optik Akış (Optical Flow) Karşılaştırması

Lucas-Kanade (Sparse): Sadece belirli özellik noktalarını (köşeler) takip eder. Çok hızlıdır.

```
# Belirli noktaları takip et
p1, st, err = cv.calcOpticalFlowPyrLK(old_gray, gray, p0, None,
**lk_params)
```

Farneback (Dense): Görüntüdeki tüm piksellerin hareketini hesaplar. Daha detaylı ama yavaştır.

```
# Tüm piksellerin akışını hesapla
flow = cv.calcOpticalFlowFarneback(prev, next, None, 0.5, 3, 15, 3, 5,
1.2, 0)
```

Arka Plan Çıkarma (Background Subtraction)

MOG2: Gaussian Mixture tabanlıdır. Gölgeleri tespit edebilir.

```
# MOG2 oluştur ve uygula
backSub = cv.createBackgroundSubtractorMOG2()
fgMask = backSub.apply(frame)
```

KNN: K-Nearest Neighbors tabanlıdır. Düşük çözünürlükte daha iyi sonuç verebilir.

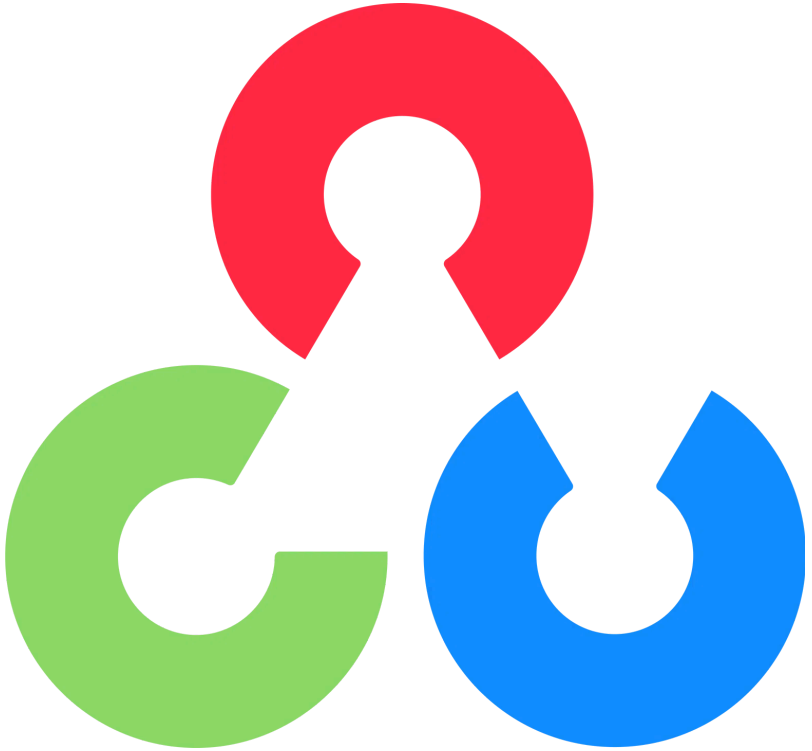
```
# KNN oluştur ve uygula
backSub = cv.createBackgroundSubtractorKNN()
fgMask = backSub.apply(frame)
```

Kritik Not: Gerçek zamanlı sistemlerde **MOG2** gölge yönetimi sayesinde genellikle tercih edilir.

Sonuç

- → OpenCV, bilgisayarlı görü dünyasının temel taşıdır ve endüstri standartlarını belirler.
- → Python entegrasyonu, karmaşık algoritmaların hızlıca prototiplenmesini sağlar.
- → Geniş topluluk desteği ve sürekli güncellenen kütüphanesiyle geleceğin teknolojilerine hazırdır.

Dinlediğiniz için teşekkürler! Sorularınız?



Kaynakça

OpenCV-Python Tutorials Root

docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

Ana eğitim dizini ve konu başlıkları hiyerarşisi.

Core Operations in OpenCV

docs.opencv.org/4.x/d7/d16/tutorial_py_table_of_contents_core.html

Temel işlemler, aritmetik ve performans optimizasyonu detayları.

Image Processing in OpenCV

docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html

Filtreleme, kenar algılama ve morfolojik işlemler dökümantasyonu.