Ar-Ge eğitim

"Merhaba arkadaşlar, bugün sizlerle görüntü işleme dünyasına adım atacağız ve OpenCV kütüphanesini detaylı bir şekilde inceleyeceğiz. Görüntü işleme, günümüzde yapay zeka, robotik ve bilgisayarla görü alanlarında devrim niteliğinde uygulamalara olanak tanıyor. Bu eğitimde, hem teoriyi anlayacak hem de pratik projelerle öğrendiklerimizi pekiştireceğiz.

OpenCV, açık kaynaklı ve çok güçlü bir kütüphane. Görüntü yükleme, işleme, şekil algılama, QR kod tespiti ve hatta gerçek zamanlı uygulamalara kadar birçok kullanım alanı var. Sizlere bu teknikleri hem teorik olarak açıklayacağım hem de kodlarla uygulamalı olarak göstereceğim.

Bu eğitim sonunda, bir görüntüyü nasıl analiz edeceğinizi, çeşitli filtreler uygulamayı, şekil algılamayı ve kameradan alınan görüntülerle çalışmayı öğreneceksiniz. Hedefimiz, öğrendiklerimizi bir mini proje ile birleştirerek görüntü işleme alanında güçlü bir temel oluşturmak.

Haydi, görüntü işleme dünyasını keşfetmeye başlayalım!"

1. Görüntü Nelerden Oluşur?

"Görüntü işleme dünyasına adım atmadan önce, bir görüntünün temel bileşenlerini anlamamız gerekiyor.

Bir dijital görüntü, küçük karelerden oluşur ve bu karelere **piksel** denir. Her piksel, görüntünün rengini ve parlaklığını temsil eden bir değer içerir.

- **Gri Tonlamalı Görüntüler**: Her piksel, 0 (siyah) ile 255 (beyaz) arasında bir değer alır.
- **Renkli Görüntüler**: Her piksel, RGB (Kırmızı, Yeşil, Mavi) bileşenlerinden oluşur. Örneğin, bir pikselin değeri (255, 0, 0) ise bu piksel kırmızı renktedir.
- **Çözünürlük**: Görüntünün genişlik ve yüksekliği, toplam piksel sayısını belirler. Çözünürlük ne kadar yüksekse, detaylar o kadar fazladır.

Görüntü işleme teknikleri ile piksel değerlerini değiştirebilir, analiz edebilir ve görüntü üzerinde farklı işlemler gerçekleştirebiliriz."

2. Görüntü Yükleme ve Kaydetme (OpenCV ile)

"Görüntü işleme projelerine başlamak için öncelikle bir görüntüyü nasıl yükleyip işleyebileceğimizi öğrenmemiz gerekiyor. OpenCV, görüntüleri kolayca yüklememizi, kaydetmemizi ve ekranda görselleştirmemizi sağlayan işlevler sunar."

```
import cv2

# Görüntü yükleme
img = cv2.imread('ornek.jpg')  # 'ornek.jpg' dosyasını aynı dizi
if(img is None):
    return 0
#Dosya yolu yanlışsa veya görüntü yüklenemezse, cv2.imread() fo
# Görüntüyü ekranda gösterme
cv2.imshow('Yüklenen Görüntü', img)
cv2.waitKey(0)  # Pencereyi kapatmak için herhangi bir tuşa bas
cv2.destroyAllWindows()

# Görüntüyü kaydetme
cv2.imwrite('kaydedilen_ornek.jpg', img)
print("Görüntü başarıyla kaydedildi!")
```

1. Renk Uzayları Hakkında Kısa Bilgi

"Dijital görüntülerde, renklerin farklı şekillerde temsil edildiği yapılar vardır ve bunlara **renk uzayları** denir. Farklı renk uzayları, görüntü işleme görevlerinde belirli avantajlar sağlar.

En yaygın renk uzayları:

- **RGB (Red, Green, Blue)**: Görüntünün üç temel renkten (kırmızı, yeşil, mavi) oluştuğu en yaygın renk uzayıdır.

- **Grayscale (Gri Tonlama)**: Görüntünün renk bilgilerinin kaldırılıp yalnızca parlaklık bilgisiyle temsil edildiği bir renk uzayıdır.
- **HSV (Hue, Saturation, Value)**: Renk, doygunluk ve parlaklık bileşenlerine ayrılan bir renk uzayıdır. Bu, özellikle renk segmentasyonu gibi işlemlerde avantajlıdır.

Renk uzayı dönüşümleri, belirli görevler için görüntüleri uygun forma dönüştürmemizi sağlar. Örneğin, bir görüntüyü gri tonlamaya çevirmek, kenar algılama işlemleri için bir ön adımdır."

2. Görüntü Dönüşümleri: Renk Uzaylarını Değiştirme

"OpenCV, bir görüntüyü kolayca farklı renk uzaylarına dönüştürmemizi sağlayan güçlü fonksiyonlar sunar.

- `cv2.cvtColor()`: Bu fonksiyon, renk uzayını değiştirmek için kullanılır.
  - Gri tonlama için: `cv2.COLOR_BGR2GRAY`
  - RGB'den HSV'ye dönüşüm için: `cv2.COLOR_BGR2HSV`

Ayrıca, renk uzayı dönüşümleri ile piksel değerlerini analiz edebilir veya değiştirebiliriz."

```
import cv2

# Görüntü yükleme
img = cv2.imread('ornek.jpg')

# Gri tonlamaya dönüştürme
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# RGB'den HSV'ye dönüştürme
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

```
# Sonuçları görselleştirme
cv2.imshow('Orijinal Görüntü', img)
cv2.imshow('Gri Tonlama Görüntüsü', gray_img)
cv2.imshow('HSV Görüntüsü', hsv_img)

cv2.waitKey(0)  # Pencereleri kapatmak için herhangi bir tuşa ba
cv2.destroyAllWindows()
```

4. Kodun Açıklaması

- **Gri Tonlama**: `cv2.COLOR_BGR2GRAY` ile renk bilgisi kaldırılarak görüntü gri tonlamaya dönüştürülür. Piksel değerleri sadece parlaklık bilgisini taşır (0-255).
- **HSV Dönüşümü**: `cv2.COLOR_BGR2HSV` ile görüntü, renk (Hue), doygunluk (Saturation) ve parlaklık (Value) bileşenlerine ayrılır.

5. Kullanım Alanları

- **Gri Tonlama**: Kenar algılama ve görüntü segmentasyonu gibi işlemlerde ön hazırlık olarak kullanılır.
- **HSV**: Renk bazlı segmentasyon, nesne takibi ve renk filtreleme gibi işlemlerde kullanılır.

1. Filtreleme Teknikleri Hakkında Kısa Bilgi

"Filtreleme teknikleri, görüntülerin piksellerine uygulanan işlemlerle, belirli özellikleri vurgulamak veya gürültüyü azaltmak amacıyla kullanılır.

Başlıca filtreleme türleri:

- **Bulanıklaştırma (Blurring)**: Görüntüyü yumuşatır ve gürültüyü azaltır.
  - Gaussian Blur: Görüntüyü doğal bir şekilde yumuşatır.
  - Median Blur: Gürültü azaltmada etkili bir yöntemdir, özellikle tuz ve biber gürültüsü için.

- **Keskinleştirme (Sharpening)**: Görüntünün kenarlarını vurgular ve detayları belirginleştirir.
- **Kenarlık Kaldırma (Edge Removal)**: Görüntüyü daha homojen hale getirmek için kullanılan yöntemlerdir.

Bu teknikler, görüntü işleme öncesinde veya belirli özellikleri ortaya çıkarmak için önemli adımlardır."

---

2. Filtreleme Teknikleri: Bulanıklaştırma ve Keskinleştirme

"OpenCV, filtreleme tekniklerini kolayca uygulamamızı sağlayan güçlü fonksiyonlar sunar:

- `cv2.GaussianBlur()`: Gaussian filtresi ile bulanıklaştırma.
- `cv2.medianBlur()`: Median filtresi ile gürültü azaltma.
- `cv2.filter2D()`: Özelleştirilmiş çekirdeklerle (kernel) filtreleme işlemleri."

```
import cv2
import numpy as np

# Görüntü yükleme
img = cv2.imread('ornek.jpg')

# Gaussian Blur ile bulanıklaştırma
gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0)

# Median Blur ile bulanıklaştırma
median_blur = cv2.medianBlur(img, 5)

# Keskinleştirme için özelleştirilmiş çekirdek
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
sharpened = cv2.filter2D(img, -1, kernel)

# Sonuçları görselleştirme
```

```
cv2.imshow('Orijinal Görüntü', img)
cv2.imshow('Gaussian Blur', gaussian_blur)
cv2.imshow('Median Blur', median_blur)
cv2.imshow('Keskinleştirilmiş Görüntü', sharpened)

cv2.waitKey(0)  # Pencereleri kapatmak için herhangi bir tuşa ba
cv2.destroyAllWindows()
```

4. Kodun Açıklaması

- **Gaussian Blur**: Görüntüyü yumuşatarak gürültüyü azaltır. `(5, 5)` boyutunda bir çekirdek kullanılmıştır.
- **Median Blur**: Gürültüyü ortalama yerine medyan değere göre azaltır, özellikle tuz-biber gürültüsüne karşı etkilidir.
- **Keskinleştirme**: Çekirdek matris ile kenarları vurgulayan bir işlem yapılır. Bu işlem, görüntünün daha detaylı görünmesini sağlar.

---

5. Kullanım Alanları

- **Bulanıklaştırma**: Gürültülü görüntüler üzerinde işlem yapmadan önce kullanılır.
- **Keskinleştirme**: Görüntünün önemli detaylarını belirginleştirir ve analiz için daha net hale getirir.

Uygulama: Görüntü Yükleme, Dönüşümleri ve Filtreleme Teknikleri

```
import cv2
import numpy as np

# 1. Görüntüyü yükleyin
img = cv2.imread('ornek.jpg')  # Görüntüyü aynı dizine ekleyin
if img is None:
    print("Görüntü yüklenemedi. Lütfen dosya yolunu kontrol edi
    exit()
```

```
# 2. Renk Dönüşümleri
# Gri tonlamaya dönüştürme
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# HSV renk uzayına dönüştürme
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# 3. Filtreleme Teknikleri
# Gaussian Blur
gaussian_blur = cv2.GaussianBlur(img, (5, 5), 0)

# Median Blur
median_blur = cv2.medianBlur(img, 5)

# Keskinleştirme filtresi
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
sharpened = cv2.filter2D(img, -1, kernel)

# 4. Sonuçları Görselleştirme
cv2.imshow('Orijinal Görüntü', img)
cv2.imshow('Gri Tonlama', gray_img)
cv2.imshow('HSV Görüntü', hsv_img)
cv2.imshow('Gaussian Blur', gaussian_blur)
cv2.imshow('Median Blur', median_blur)
cv2.imshow('Keskinleştirilmiş Görüntü', sharpened)

# 5. Görüntüyü Kaydetme
cv2.imwrite('gray_image.jpg', gray_img)
cv2.imwrite('sharpened_image.jpg', sharpened)
print("Görüntüler başarıyla kaydedildi!")
```

```
cv2.waitKey(0)  # Pencereleri kapatmak için bir tuşa basın
cv2.destroyAllWindows()
```

Kodun Açıklaması

1. **Görüntü Yükleme**: `cv2.imread()` ile orijinal görüntüyü yükleriz. Eğer görüntü yüklenemezse bir hata mesajı gösterilir.
2. **Renk Dönüşümleri**:
   - Gri tonlama (`cv2.COLOR_BGR2GRAY`) ile renk bilgisi kaldırılarak sadece parlaklık bilgisi elde edilir.
   - HSV dönüşümü (`cv2.COLOR_BGR2HSV`) ile renk, doygunluk ve parlaklık bileşenleri elde edilir.
3. **Filtreleme Teknikleri**:
   - Gaussian Blur ile yumuşatma yapılır.
   - Median Blur ile gürültü azaltılır.
   - Keskinleştirme filtresi ile görüntüdeki detaylar vurgulanır.
4. **Sonuçları Görselleştirme**: `cv2.imshow()` ile işlenmiş görüntüler ekranda gösterilir.
5. **Görüntüyü Kaydetme**: İşlenmiş görüntüler `cv2.imwrite()` ile belirtilen isimlerde kaydedilir.

Ders Notu: Kontur Tespiti

1. Kontur Nedir?

"Kontur, bir görüntüdeki nesnenin veya bir alanın sınırlarını temsil eden eğridir. Konturlar, genellikle nesnelerin şekillerini tespit etmek ve analiz etmek için kullanılır.

Kontur tespiti şu durumlarda sıkça kullanılır:

- Nesne segmentasyonu

- Şekil tanıma
- Görüntüdeki alanların veya objelerin özelliklerini ölçme

OpenCV'de konturlar, ikili (binary) görüntüler üzerinde çalışır. Bu nedenle, kontur tespiti öncesinde görüntü genellikle gri tonlamaya çevrilir ve ardından eşikleme veya kenar algılama gibi işlemler uygulanır."

---

2. Kontur Tespiti (OpenCV ile)

"OpenCV'de kontur tespiti için kullanılan ana fonksiyon `cv2.findContours()` fonksiyonudur.

- `cv2.findContours()` bir görüntüdeki konturları bulur ve bir liste olarak döndürür.
- Konturları görselleştirmek için `cv2.drawContours()` kullanılır.

Kontur tespiti için gerekli adımlar:

1. Görüntüyü gri tonlamaya çevirin.
2. Eşikleme veya kenar algılama uygulayın.
3. Konturları bulun ve görselleştirin."

```
import cv2

# Görüntü yükleme
img = cv2.imread('ornek.jpg')
if img is None:
    print("Görüntü yüklenemedi. Lütfen dosya yolunu kontrol edi
    exit()

# Görüntüyü gri tonlamaya çevirme
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Eşikleme işlemi
_, threshold_img = cv2.threshold(gray_img, 127, 255, cv2.THRESH_

# Konturları bulma
contours, hierarchy = cv2.findContours(threshold_img, cv2.RETR_T
```

```
# Konturları orijinal görüntüye çizme
contour_img = img.copy()
cv2.drawContours(contour_img, contours, -1, (0, 255, 0), 2)

# Sonuçları görselleştirme
cv2.imshow('Orijinal Görüntü', img)
cv2.imshow('Eşiklenmiş Görüntü', threshold_img)
cv2.imshow('Konturlar', contour_img)

cv2.waitKey(0)  # Pencereyi kapatmak için bir tuşa basın
cv2.destroyAllWindows()
```

4. Kodun Açıklaması

1. **Gri Tonlama**: `cv2.cvtColor()` ile görüntü gri tonlamaya çevrilir.
2. **Eşikleme**: `cv2.threshold()` ile görüntüde parlaklık seviyesine göre bir eşik belirlenir. Bu işlem, ikili bir görüntü oluşturur.
3. **Kontur Tespiti**:
   - `cv2.findContours()` konturları bulur ve bir liste döndürür.
   - `cv2.RETR_TREE`: Tüm kontur hiyerarşisini alır.
   - `cv2.CHAIN_APPROX_SIMPLE`: Kontur noktalarını sadeleştirir.
4. **Kontur Çizimi**: `cv2.drawContours()` ile konturlar orijinal görüntüye çizilir.

---

5. Kullanım Alanları

- **Şekil Tanıma**: Görüntüdeki nesnelerin geometrik şekillerini tespit etme.
- **Nesne Takibi**: Görüntüdeki nesnelerin sınırlarını takip etme.
- **Özellik Analizi**: Kontur alanı, çevresi gibi özellikleri hesaplama.

Ders Notu: Şekil Çizimi

"OpenCV, şekil çizme işlemleri için bir dizi fonksiyon sunar. Bu işlemler, beyaz bir tuval (tahta) üzerinde şekiller oluşturmak ve temel çizim tekniklerini öğrenmek için harika bir yoldur.

Kullanılabilecek Çizim Fonksiyonları

- **Çizgi Çizme**: `cv2.line()`
- **Dikdörtgen Çizme**: `cv2.rectangle()`
- **Daire Çizme**: `cv2.circle()`
- **Çokgen Çizme**: `cv2.polylines()`
- **Metin Yazma**: `cv2.putText()`

Bu fonksiyonlar, beyaz bir tahta üzerinde çeşitli şekiller çizmek için kullanılabilir."

```
import cv2
import numpy as np

# Beyaz bir tuval (tahta) oluşturma
canvas = np.ones((500, 500, 3), dtype='uint8') * 255  # 500x500

# Çizgi Çizme
cv2.line(canvas, (50, 50), (450, 50), (0, 0, 255), 3)  # Kırmızı

# Dikdörtgen Çizme
cv2.rectangle(canvas, (100, 100), (400, 300), (0, 255, 0), 5)  #

# Daire Çizme
cv2.circle(canvas, (250, 400), 50, (255, 0, 0), -1)  # Mavi dolu

# Çokgen Çizme
pts = np.array([[200, 200], [300, 200], [350, 250], [250, 350],
pts = pts.reshape((-1, 1, 2))
cv2.polylines(canvas, [pts], isClosed=True, color=(0, 0, 0), thi
```

```
# Metin Yazma
cv2.putText(canvas, 'OpenCV Drawing', (100, 450), cv2.FONT_HERSH

# Sonucu Görselleştirme
cv2.imshow('Tahta', canvas)
cv2.waitKey(0)  # Pencereyi kapatmak için bir tuşa basın
cv2.destroyAllWindows()
```

Kodun Açıklaması

1. **Beyaz Tahta**: `np.ones()` ile beyaz bir arka plan oluşturuldu. Her pikselin rengi `(255, 255, 255)` (beyaz).
2. **Çizim Fonksiyonları**:
   - `cv2.line()`: Başlangıç ve bitiş noktaları verilen bir çizgi çizer.
   - `cv2.rectangle()`: Belirtilen iki köşesi arasına bir dikdörtgen çizer.
   - `cv2.circle()`: Merkez noktası ve yarıçapı verilen bir daire çizer.
   - `cv2.polylines()`: Birden fazla nokta ile tanımlanan bir çokgen çizer.
   - `cv2.putText()`: Belirtilen konuma metin ekler.

Ders Notu: Konturları Tespit Ederek Şekilleri İşaretlemek

"Kontur tespiti, görüntüdeki nesnelerin sınırlarını bulmamızı sağlar. Bu sınırları belirledikten sonra, bu nesnelerin çevresine şekiller veya işaretler ekleyebiliriz. Örneğin, bir dikdörtgen içine almak veya merkez noktalarını işaretlemek gibi işlemler yapabiliriz."

```
import cv2

# Görüntüyü yükleme
img = cv2.imread('ornek.jpg')
if img is None:
    print("Görüntü yüklenemedi. Lütfen dosya yolunu kontrol edi
    exit()
```

```
# Görüntüyü gri tonlamaya çevirme
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Eşikleme işlemi
_, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Kontur bulma
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_

# Konturları işaretleme ve şekilleri çerçeveleme
for contour in contours:
    # Konturu çevreleyen dikdörtgen hesaplama
    x, y, w, h = cv2.boundingRect(contour)
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Konturun merkezini işaretleme
    M = cv2.moments(contour)
    if M["m00"] != 0:  # Bölme hatasını önlemek için
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])
        cv2.circle(img, (cx, cy), 5, (255, 0, 0), -1)  # Mavi m

# Sonuçları görselleştirme
cv2.imshow('Kontur Tespiti ve Şekil İşaretleme', img)
cv2.waitKey(0)  # Pencereyi kapatmak için bir tuşa basın
cv2.destroyAllWindows()
```

Kodun Açıklaması

1. **Görüntüyü Gri Tonlama**: Kontur tespiti öncesinde renkli görüntüyü gri tonlamaya çevirdik.
2. **Eşikleme**: `cv2.threshold()` ile ikili bir görüntü elde ettik. Parlaklık eşik değerine göre pikseller siyah veya beyaz olur.
3. **Kontur Tespiti**: `cv2.findContours()` ile görüntüdeki nesnelerin konturları tespit edildi.

4. **Şekilleri Çerçeveleme**:
    - `cv2.boundingRect()`: Her konturu çevreleyen dikdörtgeni hesaplar.
    - `cv2.rectangle()`: Hesaplanan dikdörtgeni görüntüye çizer.
5. **Merkez Noktalarını İşaretleme**:
    - `cv2.moments()` ile konturun ağırlık merkezi hesaplandı.
    - `cv2.circle()` ile merkez noktası işaretlendi.

Ders Notu: Video Akışı İşleme ve Renk Uzayı Dönüşümü

"Gerçek zamanlı video akışı işleme, bir kameradan alınan görüntülerin anlık olarak işlenmesi anlamına gelir. Bu, genellikle görüntüdeki nesneleri analiz etmek, efektler eklemek veya farklı renk uzayları arasında dönüşüm yapmak için kullanılır. OpenCV, kameradan gelen video akışını almak ve işlemek için `cv2.VideoCapture()` fonksiyonunu sunar."

```
import cv2

# Kameradan video akışını başlatma (0, varsayılan kamerayı temsi
cap = cv2.VideoCapture(0)

# Video akışı başarılıysa devam et
if not cap.isOpened():
    print("Kamera açılamadı.")
    exit()

while True:
    # Kameradan bir kare (frame) okuma
    ret, frame = cap.read()

    if not ret:
        print("Kare alınamadı.")
```

```
            break

        # Gri tonlama dönüşümü
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # HSV dönüşümü
        hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # Sonuçları görselleştirme
        cv2.imshow('Orijinal Görüntü', frame)
        cv2.imshow('Gri Tonlama', gray_frame)
        cv2.imshow('HSV Görüntü', hsv_frame)

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

# Video akışını serbest bırakma ve pencereleri kapatma
cap.release()
cv2.destroyAllWindows()
```

Kodun Açıklaması

1. **Kamera Bağlantısı**: `cv2.VideoCapture(0)` ile varsayılan kameradan video akışını başlatıyoruz. `0` değeri, bilgisayara bağlı olan ilk kamera cihazını belirtir.
2. **Kare Okuma**: `cap.read()` ile her bir video karesini alıyoruz. `ret` değişkeni, kare başarılı şekilde alındığında `True` değerini döner.
3. **Renk Uzayı Dönüşümleri**:
    - **Gri Tonlama**: `cv2.cvtColor()` ile renkli görüntüyü gri tonlamaya dönüştürüyoruz.
    - **HSV**: `cv2.cvtColor()` ile renkli görüntüyü HSV renk uzayına dönüştürüyoruz.
4. **Sonuçları Görselleştirme**: `cv2.imshow()` ile orijinal görüntü, gri tonlama ve HSV görüntülerini ayrı pencerelerde gösteriyoruz.

5. **Çıkış**: 'q' tuşuna basıldığında video akışını sonlandırıyoruz ve pencereleri kapatıyoruz.

Ders Notu: Anlık Görüntü İşleme Teknikleri - Kenar Algılama, Şekil Tespiti ve Filtreleme

"Gerçek zamanlı görüntü işleme, bir video akışındaki her kare üzerinde işlem yaparak anında sonuç elde etmeyi sağlar. Kenar algılama, şekil tespiti ve filtreleme gibi temel görüntü işleme teknikleri, bu tür uygulamalarda sıklıkla kullanılır.

- **Kenar Algılama: Görüntüdeki kenarları tespit etmek için genellikle Canny Edge Detection kullanılır.**
- **Şekil Tespiti: Görüntüdeki belirli şekilleri (dikdörtgen, çember vb.) tespit etmek için kontur bulma yöntemleri kullanılır.**
- **Filtreleme: Gürültü azaltma ve görüntüyü yumuşatma için Gaussian Blur gibi filtreler kullanılır."**

```
import cv2
import numpy as np

# Kameradan video akışını başlatma (0, varsayılan kamerayı temsi
cap = cv2.VideoCapture(0)

# Video akışı başarılıysa devam et
if not cap.isOpened():
    print("Kamera açılamadı.")
    exit()

while True:
    # Kameradan bir kare (frame) okuma
    ret, frame = cap.read()
```

```
    if not ret:
        print("Kare alınamadı.")
        break

    # 1. Gri tonlama dönüşümü
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # 2. Gaussian Blur (Bulanıklaştırma)
    blurred_frame = cv2.GaussianBlur(gray_frame, (5, 5), 0)

    # 3. Kenar Algılama (Canny)
    edges = cv2.Canny(blurred_frame, 100, 200)

    # 4. Şekil Tespiti (Konturlar)
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2

    # 5. Konturları çizme
    frame_with_contours = frame.copy()
    for contour in contours:
        if cv2.contourArea(contour) > 100:  # Küçük alanları fil
            cv2.drawContours(frame_with_contours, [contour], -1,

    # 6. Sonuçları görselleştirme
    cv2.imshow('Orijinal Görüntü', frame)
    cv2.imshow('Kenar Algılama (Canny)', edges)
    cv2.imshow('Şekil Tespiti ve Konturlar', frame_with_contours

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

# Video akışını serbest bırakma ve pencereleri kapatma
cap.release()
cv2.destroyAllWindows()
```

Kodun Açıklaması

1. **Gri Tonlama**: `cv2.cvtColor()` fonksiyonu ile renkli görüntüyü gri tonlamaya çeviriyoruz. Kenar algılama ve şekil tespiti işlemleri genellikle gri tonlamalı görüntüler üzerinde yapılır.
2. **Gaussian Blur**: `cv2.GaussianBlur()` fonksiyonu ile görüntüyü yumuşatıyoruz. Bu, kenar algılama sırasında gürültüyü azaltarak daha net kenarlar elde etmemize yardımcı olur.
3. **Kenar Algılama (Canny)**: `cv2.Canny()` fonksiyonu ile görüntüdeki kenarları tespit ediyoruz. Bu, kenarları belirginleştiren bir algoritmadır.
4. **Şekil Tespiti (Konturlar)**: `cv2.findContours()` fonksiyonu ile Canny kenar algılama sonucu elde edilen kenarları kullanarak konturları buluyoruz. Konturları çizmek için `cv2.drawContours()` kullanıyoruz.
5. **Sonuçları Görselleştirme**: `cv2.imshow()` ile üç farklı pencere açıyoruz:
   - Orijinal görüntü
   - Canny kenar algılama sonucu
   - Konturlar ve şekil tespiti sonucu
6. **Çıkış**: 'q' tuşuna basıldığında video akışını durduruyoruz ve pencereleri kapatıyoruz.