GİRİŞ

Görüntü işleme teknolojisi günümüzde hemen hemen her alanda etkin olarak rol almaya başlamıştır. Günlük hayatta telefonlarda, bilgisayarlarda, araçlarda ve diğer çoğu teknolojik alette bu teknoloji kullanılmaktadır.

Telefonlarda ve bilgisayarda insan yüzünün tanıtılması ile ekran kilidinin açılması, droneların bilgisayar yardımı ile kullanılması gibi örneklerde görüntü işleme teknolojisi aktif bir rol oynamaktadır.

Görüntü işleme, genel olarak resimsel bilgilerin analizine yönelik bir yöntem olarak tanımlanabilir. Görüntü işleme genel konumunda resimsel bilgilerin değişmesine ve analizlerine bağımlıdır ve bunlarla ilişkilidir. Günlük yaşamda görüntü işlemenin örnekleriyle pek çok kez karşılaşmaktayız. Belki de en yaygın örneği gözlüklerdir. Düzeltici gözlükler (numaralı), gözlenen görüntüleri, gözdeki bazı sapmalara göre, değiştirip, kompanze edip, görüntünün göz ile temasından önce görüntünün düzeltilmesi için görev görürler. Bir diğer yaygın görüntü işleme örneği ise televizyondaki parlaklık ve kontrast ayarlamalarıdır. Bunu yaparak, subjektif görüntüyü bizim için en cazip imaj haline gelene dek ayarlayabiliriz. Küçük bir havuzdaki su bile, görüntünün formunu değiştirebilir. Yansıyan görüntü yalnızca tersine çevrilmekle kalmayıp, bir de genelde suyun hareketine bağlı olarak bükülmeler, değişmeler göstermektedir. Her gün karşılaştığımız ve en güçlü görüntü işleme sisteminin örneği herhalde, insan beyin ve gözünden ibarettir. Bu biyolojik sistem görüntüleri inanılmaz hızla alır, güçlendirir, keser, analiz

eder ve saklar. Bu sistem, diğer sistemlere göre en kabul edilendir. Tüm bunlar, doğal olarak kabul edilen ancak düşünüldüğünde ne kadar muhteşem oldukları algılanan çok yaygın görüntü işleme örnekleridir. [1]

Drone yardımı ile görüntü işleme insansız hava araçlarının yaygınlaşması ile birlikte yoğun ilgi gören alanlardan biridir.

1. GÖRÜNTÜ İŞLEME

"Görüntü İşleme" terimi, günümüzün uygulamalı bilgisayar bilimlerinin en sıcak "anahtar kelimelerinden" birisi olmuştur. Görüntü işleme genel konumunda resimsel bilgilerin değişmesine ve analizine bağımlıdır ve bunlarla ilişkilidir. Günlük yaşamda görüntü işlemenin örnekleriyle pek çok kez karşılaşmaktayız. Yaygın görüntü örneklerinden biri televizyondaki parlaklık ve konstrat ayarlamalarıdır. Bunu yaparak, subjektif görüntüyü bizim için en cazip imaj haline gelene dek ayarlayabiliriz. [1]

Teknolojinin ilerlemesiyle birlikte insanlar git gide kendi yapabildiği işleri bilgisayarlara da yaptırmaya çalışma başladı. İnsanların görüp algıladığı gibi bilgisayarların da algılamasını, öğrenmesini hatta bazı noktalarda düşünmesini sağlamak için birçok çalışma yapıldı ve yapılmaya devam ediyor.

İnsanlar gözleri ile çevresindeki şeyleri algılar ve bu bilgiyi beyne iletir. Bu aşamada veri henüz anlamsızdır, beyne gelen bilgi burada işlenir ve anlam kazanır. Görüntü işleme de buna benzer bir yapıdadır. Bilgisayara verilen görüntünün bilgisayar tarafından anlaşılması yani işlenmesidir.

Görüntü işleme ile bir görüntü üzerinden istediğimiz bilgileri alabiliriz. Bu bir görüntü üzerindeki belirli nesneleri bulmak olabilir, bir görüntüdeki insanları bulmak olabilir. Bu tarz veriler görüntü üzerinden elde etmek istendiğinde görüntü işleme algoritmaları kullanılır.

Görüntü işleme algoritmaları gün geçtikçe daha da gelişmeye devam etmektedir. Çoğu uygulamada veri toplamayı kolaylaştıran bu algoritmalar sık sık tercih edilir olmuşlardır.

Her gün karşılaştığımız ve en güçlü görüntü işleme sisteminin örneği herhalde, insan beyin ve gözünden ibarettir. Bu biyolojik sistem görüntüleri inanılmaz hızla alır,

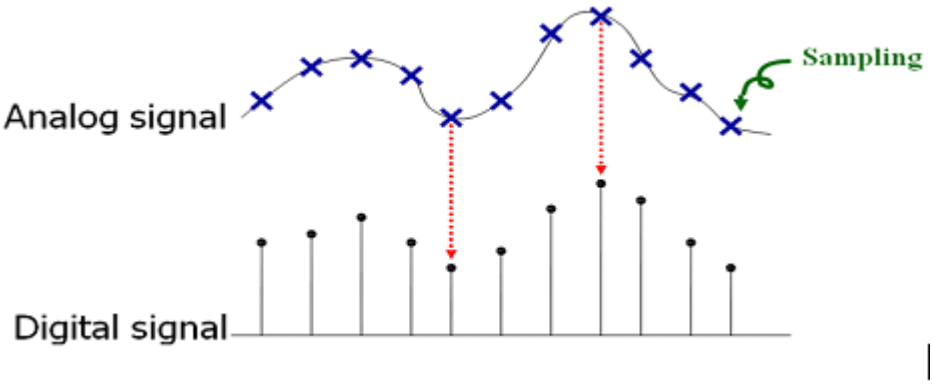
güçlendirir, keser, analiz eder ve saklar. Bu sistem diğer sistemlere göre en kabul edilendir. Tüm bunlar doğal ve kabul edilen ancak düşünüldüğünde ne kadar muhteşem oldukları algılanan çok yaygın görüntü işleme örnekleridir. [2,3]

1.1 Temel Kavramlar

Görüntü: Üç boyutlu nesnelerin iki boyutlu düzlem üzerinde haritalanmasıdır. Bu haritalamada her bir noktanın konum bilgisi (f(x,y)) ve renk bilgisi tutulur. Görüntü iki farklı başlık altında incelenebilir.

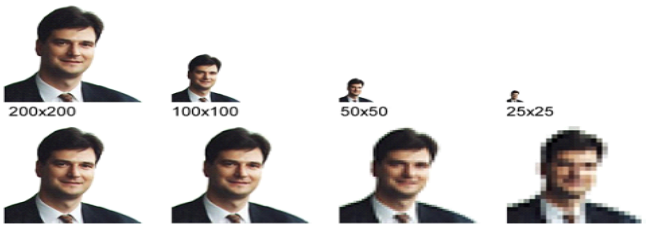
Analog Görüntü: Analog görüntüde görüntüyü oluşturan fonksiyonun f(x,y) değişkenleri reel değerler alıyorsa bu görüntü analog görüntüdür. Sayısal bilgisayarlar sürekli fonksiyonları/parametreleri işleyemezler. Bu fonksiyonların sayısallaştırılması gerekir. [4]

Dijital(Sayısal) Görüntü: f(x,y) şeklinde temsil edilen sürekli görüntüyü(analog görüntü) ayrık örnekler cinsinden ifade edilmesidir. Gösterimi f[x,y] şeklindedir. [4]



Şekil 1.1 Analog sinyalin dijital sinyale dönüştürülmesi.

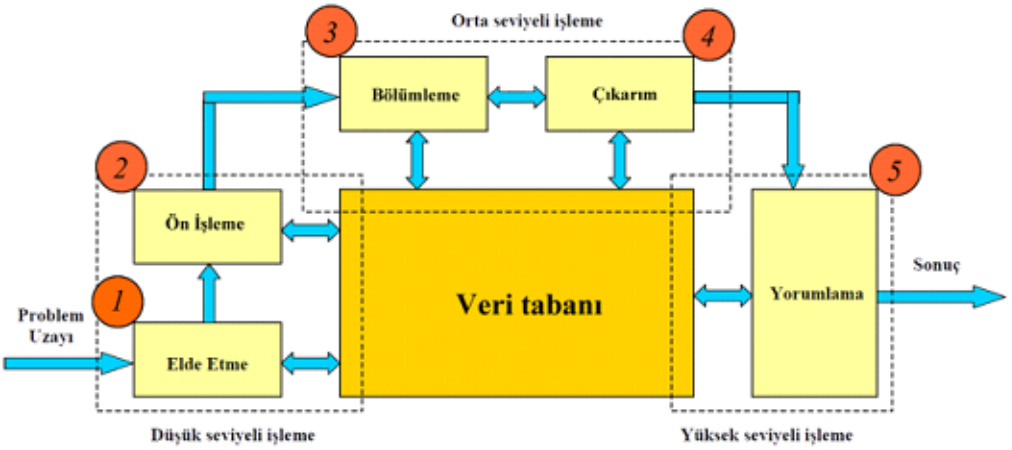
Piksel: Dijital bir görüntünün iki boyutlu dizi şeklindeki her bir elemanına piksel denir.



Şekil 1.2: Bir fotoğrafın farklı piksel boyutları ile gösterimi.

1.1.1 Sayısal Görüntü İşleme

Sayısal görüntü işleme, analog bir görüntünün sayısal biçime dönüştürülmesi ve daha sonra sayısal bilgisayarlarla işlenmesidir.



Şekil 1.3: Sayısal görüntü işleme adımları.

Sırasıyla sayısal görüntü işleme adımları;

Elde Etme: Sayısal görüntünün sayısal kamera ile elde etme işlemidir. [4]

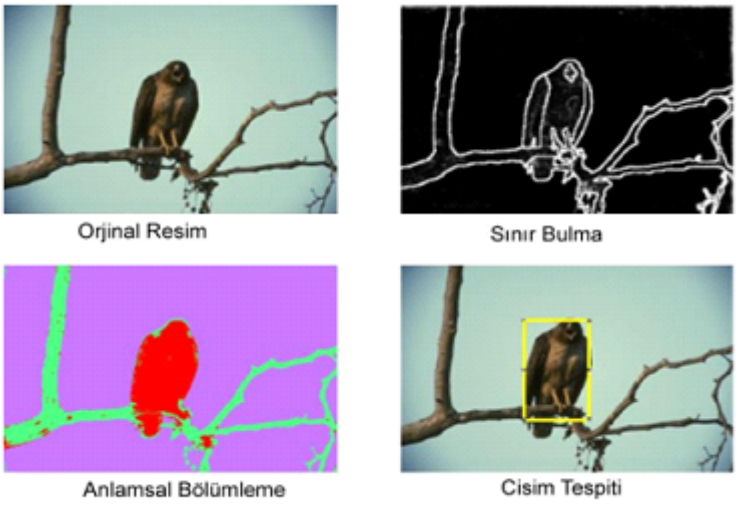
Ön İşleme: Ön-işleme, elde edilen sayısal görüntüyü kullanmadan önce daha başarılı bir sonuç elde edebilmek amacıyla görüntünün bazı ön işlemlerden geçirilmesidir. Bu işlemler temel olarak; görüntü iyileştirme (image enhancement), görüntü onarma (image restoration) ve görüntü sıkıştırma (image compression)'dır. [4]

Bölümleme: Bir görüntüdeki nesne ve arka planın veya görüntü içerisindeki ilgilenilen değişik özelliklere sahip bölgelerin birbirinden ayrıştırılması işlemidir. Bölümleme; bir görüntüdeki nesnenin sınırları ve alanlarını tespit ederek şekli üzerinde ham bilgiler üretir. Eğer nesnelerin şekilleriyle ilgileniyorsak, bölümleme bize o nesnenin kenarları, köşeleri ve sınırları hakkında bilgiler verir. Diğer taraftan görüntü içerisindeki nesnelerin yüzey kaplaması, alanı, renkleri, iskeleti gibi iç özellikleriyle ilgileniliyorsa bölgesel bölümleme yapılmalıdır. Karakter veya genel

olarak örüntü (pattern) tanıma gibi oldukça karmaşık problemlerin çözümü için her iki bölümleme yönteminin (sınırlar ve alanlar) bir arada kullanılması gerekebilir. [4]

Çıkarım: Görüntüden elde edilen ham bilgilerin, ilgilenilen ayrıntıların ön plana çıkarılmasıdır. Yani aranan özellikli alanların arka plandan ve birbirinden ayrıştırılmasıdır. [4]

Yorumlama: Yüksek seviyeli görüntü işleme grubuna giren bu aşamada, çeşitli karar verme mekanizmaları (yapay zeka algoritmaları gibi) ile görüntüdeki arka plandan çıkarılmış nesnelerin veya bölgelerin etiketlendirilmesi, sınıflandırılması yapılır. [4]



Şekil 1.4: Bir resimdeki cismin tespit aşamaları.

1.2 Görüntü İşleme Teknikleri

Dijital bir görüntü, bilgisayar dilinde sınırlı sayıda bit tarafından temsil edilen bir dizi gerçek sayıdan ibarettir. Dijital görüntü işleme yöntemlerinin temel avantajı; çok yönlü tarama kabiliyeti, tekrarlanabilirliği ve orijinal veri hassasiyetinin korunmasıdır. Başlıca görüntü işleme teknikleri; görüntü önişleme, görüntü geliştirme, görüntü ayırma, özellik çıkartma ve görüntü sınıflandırılmasıdır. [5]

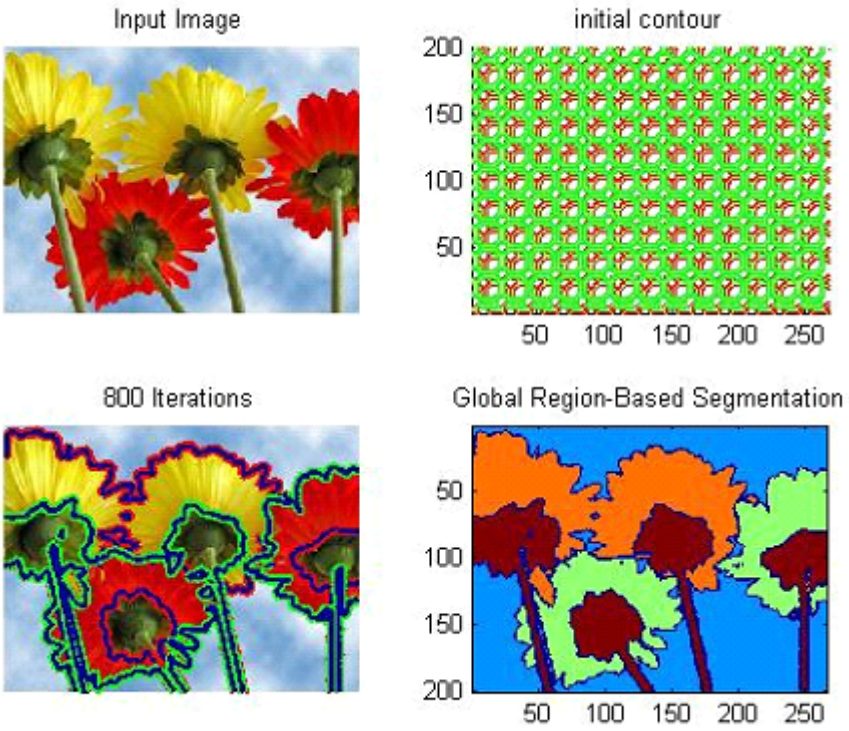
Görüntü Önişleme ve Geliştirme: Uydulardan veya dijital kameralardan elde edilen görüntüler, görüntü yakalarken görüntüleme alt sistemlerinin ve aydınlatma koşullarının sınırlamaları nedeniyle kontrast ve parlaklık hataları yaratır, görüntüler

farklı türde seslere sahip olabilir. Görüntü önişleme piksellerin geometrisi ve parlaklık değerleri ile ilgili hataların engellenmesini sağlar. Bu hatalar ampirik ya da teorik matematik modeller kullanılarak düzeltilir. [5]

Görüntü geliştirme ise, görselin görünümünü iyileştirerek görüntünün insan veya makine yorumu için daha uygun forma dönüştürülmesini sağlar veya sonraki analizler için belirli görüntü özelliklerini vurgular. Görsel geliştirme temel olarak piksel parlaklık değerleri üzerindeki oynamalarla sağlanır. Kontrast ve kenar geliştirme, sözde renklendirme, gürültü filtreleme, keskinleştirme ve büyütme uygulamalara örnektir. [5]

Şekil 1.5: Görüntü Önişleme ile işlenmiş bir resim.

Görüntü Bölütleme: Görüntü bölütleme görüntüyü kendini oluşturan bileşenlere ve nesnelere ayırma işlemidir. Uygulama konusuna göre görüntü bölütleme işleminin ne zaman durdurulacağı belirlenir. Aranan nesne veya bileşen görüntüden elde edildiği anda bölütleme işlemi durdurulur. Örneğin, havadan hedef tespitinde ilk olarak yol ve araçlar bütünü olan fotoğraf, yol ve araç olarak ayrılır. Her bir araç ve yol tek başına bu görüntünün bölünmüş parçalarını oluşturur. Hedef kriterlerine göre uygun olan alt resme ulaşıldığında görüntü bölütleme işlemi durdurulur. Görüntü bölütleme için görüntü eşiği oluşturma yöntemleri kullanılır. Bu sınırlama yöntemlerinde nesne pikselleri bir gri seviyesine arka plan ise farklı piksellere ayarlanır. Genellikle nesne pikselleri siyah ve arka plan beyazdır. Piksel farkından kaynaklanan bu ikili görüntüler gri ölçeğe göre değerlendirilerek görüntü ayrımı yapılır. [5]



Şekil 1. 6: Görüntü Bölütleme örneği.

Özellik Çıkartma: Görüntü bölütlemenin bir üst versiyonu olarak düşünülebilir. Bu kez amaç sadece nesneleri arka plandan ayırmak değil; nesnenin boyutu, şekli, kompozisyonu, konumu gibi özelliklere göre nesneyi tanımlamaktır. Matematiksel ifadeyle, sınıf içi örüntü değişkenliğini en aza indirirken sınıf örüntüsünün değişkenliğini arttırmak için ham veri bilgisinden çıkarım yapma işlemidir. Bu sayede nesnenin kantitatif özellik ölçümleri, sınıflandırılması ve tanımlanması kolaylaşır. Tanıma sisteminin verimliliği üzerinde gözlemlenebilir bir etkiye sahip olduğu için özellik çıkarma safhası görsel işleme aşamalarında önemli bir basamaktır. [5]

Görüntü Sınıflandırma: Sınıflandırma en sık kullanılan bilgi çıkarma yöntemlerinden biridir. En basit haliyle bir piksel ya da piksel grubunun gri değerine bağlı olarak etiketlenmesidir. Etiketleme sırasında bir nesnenin çoklu özellikleri kullanır ve bunun için veri tabanında aynı nesneye ait birçok görüntünün olması sınıflandırma işlemini kolaylaştırır. Uzaktan algılama teknolojisini kullanan araçlarda

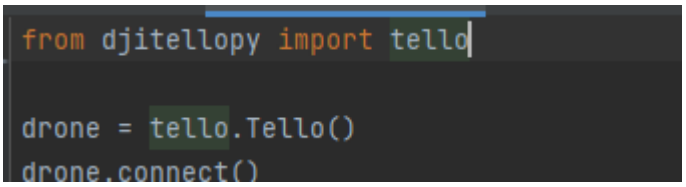
bilgi çıkarım tekniklerinin çoğu görüntülerin spektral yansıma özelliklerini analiz eder. Spektral analizler özelleştirilmiş bazı algoritmalar tarafından yapılır ve genel olarak, denetimli ve denetimsiz multispektral sınıflandırma olmak üzere iki farklı sistemle yapılır. [5]

2. TELLO DRONE ve BİLGİSAYAR BAĞLANTISI

Proje içerisinde tello drone yardımı ile insan tanıma adımı, insan yüzünü takip etme ve en son olarak da çoklu nesne tanıma adımlarını adımları gerçekleştirildi.

2.1. Tello Drone'a Bağlanma

Tello Drone'a bağlantı için djitellopy kütüphanesi kullanıldı.
djitellopy ve djitellopy2 olarak aslında 2 farklı sürüm mevcut. İlk aşamada djitellopy kütüphanesi basit işlemlerimiz karşılasa da daha karmaşık adımlar için djitellopy2 kullanımı daha ideal bir seçimdir.



Şekil 2.1: Djitellopy kütüphanesinin proje import kodu.

2.2. Tello Drone Hareket Kontrolü

Drone hareketi için djitellopy kütüphanesinde aslında sabit tanımlar mevcut ama bu hareket fonksiyonları drone'un sürümüne göre veya djitellopy2 kullanılmasına, hatta opencv-python sürümüne göre bile farklılık gösterebiliyor. Başlangıç için djitellopy içindeki methodları kullanılarak gerçekleştirildi, bunun en temel sebebi ise djitellopy'nin daha basit işler için tasarlanmış olup sürümler arası methodlarda aşırı farklılıklar bulunmamasından kaynaklıdır.

```python
from djitellopy import tello
from time import sleep

drone = tello.Tello()
drone.connect()

print(drone.get_battery())

drone.takeoff()
#(y,x,y,y) -> x forward,backward
drone.send_rc_control(0, 50, 0, 0)
sleep(2)
#(x,y,y,y) -> x left,right
drone.send_rc_control(50, 0, 0, 0)
sleep(2)
drone.send_rc_control(0, -50, 0, 0)
sleep(2)
drone.send_rc_control(-50, 0, 0, 0)
sleep(2)
#(y,y,y,x) -> x rotate
drone.send_rc_control(0, 0, 0, 30)
sleep(2)
drone.send_rc_control(0, 0, 0, 0)
drone.land()
```

Şekil 2.2: Drone'un hareketleri için kullanılan kod parçası.

Sırasıyla kod satırlarını şu şekilde açıklanabilir:

drone.get_battery(): Drone'un ne kadar şarjı olduğunu takip edebilmek için kullanılır. 20'den az olduğundan dronedan streaming yapılamaz ve görüntü alınamaz.

drone.takeoff(): Drone'un kalkışı için kullanılır. Eğer custom bir değer verilmezse default değeri 100'dür.

drone.send_rc_control(0,50,0,0):((left,right),(forward,backward),(up,down),(rotate)) olacak şekilde bu parametleri değiştirerek hareketi sağlıyor. Hareketleri net görebilmek için time.sleep() methoduyla aralara bekleme süresi koyulabilir.

drone.land(): Drone'un inişi için kullanılır.

2.1.1. Tello Drone'un Klavye ile Kontrolü ve Kendi Kendine Hareket Edebilmesinin Sağlanması

Pygame kütüphanesi ile klavye inputlarının alınabilmesi sağlandı. Sonrasında klavyeden alınan girdileri initialize edebilmesi için gerekli modül hazırlandı.

```python
import pygame


def init():
    pygame.init()
    win = pygame.display.set_mode((400, 400))


def getKey(keyName):
    ans = False
    for eve in pygame.event.get(): pass
    keyInput = pygame.key.get_pressed()
    mkey = getattr(pygame, 'K_{}'.format(keyName))
    if keyInput[mkey]:
        ans = True
    pygame.display.update()

    return ans
```

Şekil 2.3: Klavye girdilerinin initialize edilmesini sağlayan modül.

Bu şekilde klavye değerleri ile drone kontrolü sağlandı.

Sonrasında Drone'un kendi kendine hareket edebilmesi için bir dikdörtgen oluşturularak bu alan içerisinde hareket edilmesi sağlandı. Bunun için Pygame pencere boyutu ayarlandı.

Sonrasında işlemin daha kolay gerçekleştirilebilmesi için method hazırlandı.

Bu methodun temel amacı Pygame'de girilen input isimleri K_UP veya K_DOWN gibi isimler olması ve bunlar Pygame'in kendi sabit input isim tanımlamalarıdır, bunu getattr() satırında formatlanması sağlandı.
Fonksiyonun çalışma biçimi ise özetle eğer bir event var ise döngünün içine girerek hangi girdinin veya tuşun basıldığınn algılanması sağlanıyor. Daha sonra girdi ismi formatlanıyor ve return ediliyor. Ve aslında en alttada main içinde bu modülü bu .py dosyası çağrıldığında initlemesini sağlanıyor.

```python
import KeyPressModule as kp
from djitellopy import tello
import cv2
import time

kp.init()
drone = tello.Tello()
drone.connect()

drone.streamon()
```

Şekil 2.4: Modül importu ve initleme işlemi.

Bir drone nesnesi oluşturup Tello Drone'a bağlanıyor ve video kaydı oluşturuluyor.

drone.streamon(): Drone'dan video yakalama işlemini salar.

getKeyboardInput(): Methodunda hangi tuş basıldıysa Drone'a bir hareket yollayabilmek için değerleri returnlüyoruz.

lr: left/right
fb: forward/backward
ud: up/down
yv: velocity

Başlangıç değerleri 0 olarak ayarlanıyor. Eğer Pygame penceresi üzerinde herhangi bir tuşa basılmazsa Drone sabit kalıyor.

Eğer herhangi bir girdi değeri gelirse KeyPressModule.py yukarıda açıklanan modül üzerinden getKey() methodu initleniyor ve hangi tuşa basıldıysa ona göre speed değeri çıktı olarak Drone'a gönderiliyor. Speed değeri elli olarak ayarlandı.

cv2.imwrite(f'Resources/Images/ {time.time()}.jpg', img): Bu method "Z" tuşuna basıldığı andaki Frame'i .jpg olarak belirtilen dosya yoluna kaydetme işlemi sağlanır.

2.3. Tello Drone'un Kapalı Alan İçerisinde Konum Tespiti ve Hareketlerinin Çizimi

Burada açısal ve ilerleme hızını deneme yanılma yollarıyla öncelikle tespit edildi ve daha sonradan belli hata oranlarıyla atamalarını gerçekleştirildi.

Başlangıç koordinatları atandı ve başlangıç noktasını da drone'un ilk bulunduğu konumu 0,0 olarak sabitlendi.

```
import math

import KeyPressModule as kp
from djitellopy import tello
import cv2
import numpy as np
import time

#####PARAMETERS#####
fspeed = 117 / 10  # forward speed in cm/s
aspeed = 360 / 10  # angular speed in degrees/s

interval = 0.25
dInterval = fspeed * interval
aInterval = aspeed * interval

kp.init()
drone = tello.Tello()
drone.connect()

print(drone.get_battery())

points = [(0, 0), (0, 0)]
```

Şekil 2.5: Konum için hazırlanmış kod parçası.

```python
def drawPoints(img, points):
    for point in points:
        cv2.circle(img, point, 5, (0, 0, 255), cv2.FILLED)

    cv2.circle(img, points[-1], 8, (0, 255, 0), cv2.FILLED)
    cv2.putText(img, f'{(points[-1][0] - 500) / 100},{(points[-1][1] - 500) / 100}m',
                (points[-1][0] + 10, points[-1][1] + 30), cv2.FONT_HERSHEY_PLAIN, 1, (255, 0, 255), 1)


while True:
    vals = getKeyboardInput()
    drone.send_rc_control(vals[0], vals[1], vals[2], vals[3])
    img = np.zeros((1000, 1000, 3), np.uint8)
    if (points[-1][0] != vals[4] or points[-1][1] != vals[5]):
        points.append((vals[4], vals[5]))
    drawPoints(img, points)
    cv2.imshow("Output", img)
    cv2.waitKey(1)
```

Şekil 2.6: Drone hareketlerinin çizimi için kullanılan kod parçası.

drawPoints() methodunun amacı Drone'un her hareketinde bir önceki noktaya bir daire çizebilmek.

Klavye girdilerimiz ile aldığımız değerleri aslında oluşturulan 0'lar matrisinde 1 ile değiştirerek hareket için bir yol çizebilmek için kullanıldı.

Gidilen yönün tespiti de aslında açısal hızın sürekli kontrol ederek değişikliklere göre bir üst veya alt(veya yanlar) matrise 1 inputunu vererek gerçekleştirildi.

3. OPENCV ile GÖRÜNTÜ İŞLEME

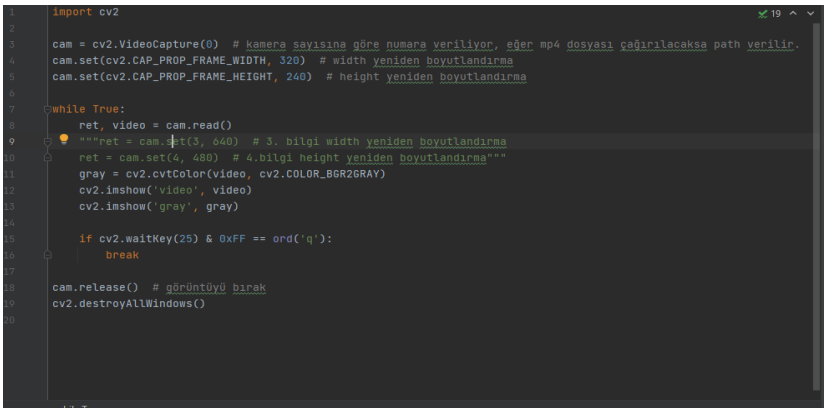
Drone ile görüntü işlemek için Drone'un kamerası ile yakaladığımız video kayıtları ve fotoğraf görüntüleri temel alındı. Burada ilk adım insan tespiti ve sonrasında ise insan yüzü takibiydi. Bundan sonraki adım ise çoklu obje tespitiydi.

3.1 Basit Görüntü İşleme

Basit Görüntü İşleme işlemi içerisinde temel olarak 4 farklı yapı kullandık.

- Görüntü Yakalama
- Görüntü Kaydetme
- Görüntü Yeniden Boyutlandırma
- Aritmetik İşlemler
    - Maskeleme
        - Resim Okuma-Yazma
        - Resim Çerçeveleme
        - Resim Format Değiştirme

Cam.py adlı bir dosya oluşturarak içerisinde görüntü yakalama işlemini gerçekleştirdik. Klavyeden “Q” harfi tetiklenene kadar görüntü yakalama işlemi sürdürüldü.



```python
import cv2

cam = cv2.VideoCapture(0)  # kamera sayısına göre numara veriliyor, eğer mp4 dosyası çağırılacaksa path verilir.
cam.set(cv2.CAP_PROP_FRAME_WIDTH, 320)  # width yeniden boyutlandırma
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)  # height yeniden boyutlandırma

while True:
    ret, video = cam.read()
    """ret = cam.set(3, 640)  # 3. bilgi width yeniden boyutlandırma
    ret = cam.set(4, 480)  # 4.bilgi height yeniden boyutlandırma"""
    gray = cv2.cvtColor(video, cv2.COLOR_BGR2GRAY)
    cv2.imshow('video', video)
    cv2.imshow('gray', gray)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cam.release()  # görüntüyü bırak
cv2.destroyAllWindows()
```

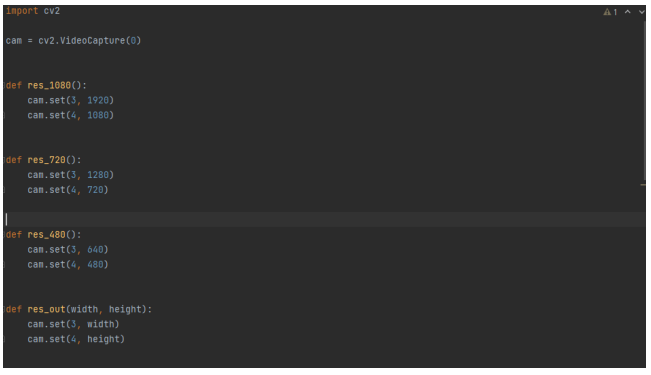
Şekil 3.1: Görüntü yakalama işlemi için yazılmış kod parçası.

Cam_save.py dosyası içerisinde yakalanan görüntülerin kaydedilmesi sağlandı.

```python
import cv2

cam = cv2.VideoCapture(0)

fourcc = cv2.VideoWriter_fourcc(*'XVID')  # hangi formatta kaydedilsin

save = cv2.VideoWriter('save.avi', fourcc, 20.0, (640, 480))  # 20.0 = saniye kaç görüntü, 4. parametre video boyutu

while (cam.isOpened()):
    ret, frame = cam.read()
    if ret == True:
        frame = cv2.flip(frame, 0)
        save.write(frame)
        cv2.imshow('save', frame)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

cam.release()
save.release()
cv2.destroyAllWindows()
```

Şekil 3.2: Yakalanan görüntünün kaydedilme işlemi.

Cam_scale.py dosyası içerisinde görüntünün tekrar boyutlandırılması sağlandı.



```python
import cv2

cam = cv2.VideoCapture(0)


def res_1080():
    cam.set(3, 1920)
    cam.set(4, 1080)


def res_720():
    cam.set(3, 1280)
    cam.set(4, 720)


def res_480():
    cam.set(3, 640)
    cam.set(4, 480)


def res_out(width, height):
    cam.set(3, width)
    cam.set(4, height)
```

Şekil 3.3: Görüntü boyutlandırma işleminin kod parçası.

Aritmetik_islemler.py dosyasının altında gerekli aritmetik işlemler yapıldı.

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

image = cv2.imread('saat.png')

image[80, 80] = [0, 255, 255]

rectangle = image[75:150, 75:150]
image[0:75, 0:75] = rectangle
# image[75:150, 75:150] = [255, 255, 0]
cv2.rectangle(image, (75, 75), (150, 150), (0, 255, 0), 2)
cv2.imshow('clock', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Şekil 3.4: Pikselleri çerçeveleme işlemi.

cv2.rectangle() methodu belli bir aralıktaki pikselleri çerçeve içinealma işlemi yapıyor.
cv2.rectangle(source,dim(hangi aralık),renk,kalınlık) parametrelerini alır.

```python
import cv2
import numpy as np

img1 = cv2.imread('image1.png')
img2 = cv2.imread('image2.png')

# sum = cv2.add(img1, img2)
weighted = cv2.addWeighted(img1, 0.7, img2, 0.3,0)
cv2.imshow('sum', weighted)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Şekil 3.5: Resim toplama işlemi.

cv2.add() method ile 2 resim toplanabilir veya
cv2.addWeighted() methoduyla oransal bir şekildetoplanabilir.

```python
import cv2
from matplotlib import pyplot as plt

img1 = cv2.imread('image.png')
img2 = cv2.imread('opencvdark.png')

scale = 30
width = int(img2.shape[1] * (scale / 100))
height = int(img2.shape[0] * (scale / 100))

dim = (width, height)
img2 = cv2.resize(img2, dim, interpolation=cv2.INTER_AREA)

print(img2.shape)  # (1024, 831, 3) row,column,channel
row, column, channel = img2.shape
roi = img1[0:row, 0:column]

img2gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)  # image2'yi gri tona çevirdik.
# cv2.imshow('img2gray', img2gray)
ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)  # 10-255 eşik değerleri arasını alıyoruz.
print(ret)
# cv2.imshow('mask', mask)
```

Şekil 3.6: Yeniden boyutlandırma işlemi.

Alınan image 2 %30 oranında scale ediliyor ve yeniden boyutlandırılıyor.

Daha sonra roi adlı bir değişkende image2'nin width ve height'ini saklanıyor. Bu değer image1 üzerinde işlem yapılırken kullanılacaktır.

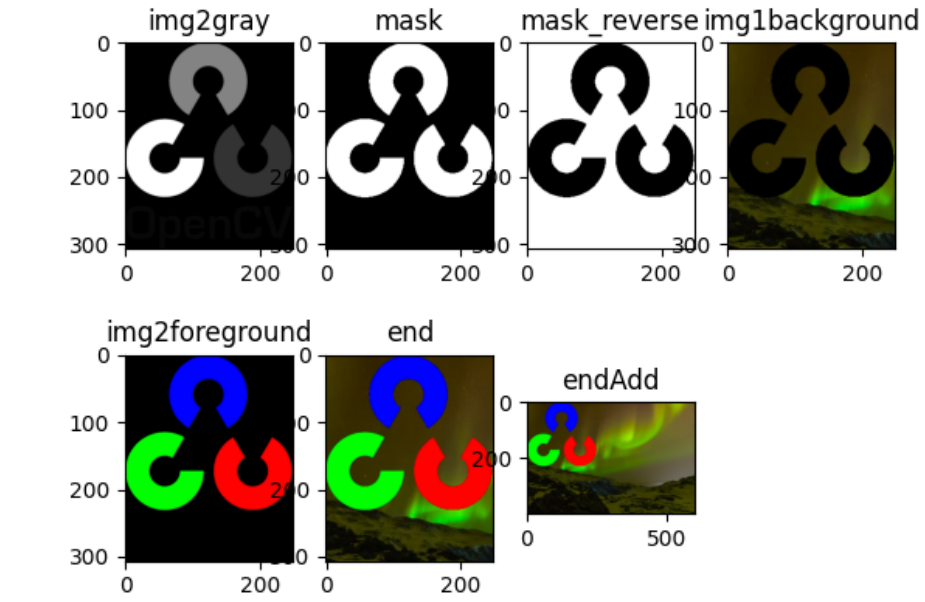
cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY): Methoduyla aslında 2 işlem yapılıyor.
Birincisi img2'nin RGB renk paletini BGR formatına dönüştürülmesi daha sonrasında ise tüm pikseller gri tona çeviriliyor.

Ret,mask = cv2.threshold(img2gray,10,255,cv2.THRESH_BINARY): Methoduyla ise bir eşik aralığı belirleniyor. Aslında img2gray resminde renk değerleri 10-255 arasındaki piksellerin alınması sağlanıyor. Bu bir maskeleme işlemidir.

```python
mask_reverse = cv2.bitwise_not(mask)
# cv2.imshow('mask_reverse', mask_reverse)

img1background = cv2.bitwise_and(roi, roi, mask=mask_reverse)
# cv2.imshow('img1background', img1background)

img2foreground = cv2.bitwise_and(img2, img2, mask=mask)
# cv2.imshow('img2foreground', img2foreground)

end = cv2.add(img1background, img2foreground)
# cv2.imshow('end', end)

img1[0:row, 0:column] = end
# cv2.imshow('endAdd', img1)

plt.subplot(241), plt.imshow(img2gray, 'gray'), plt.title('img2gray')
plt.subplot(242), plt.imshow(mask, 'gray'), plt.title('mask')
plt.subplot(243), plt.imshow(mask_reverse, 'gray'), plt.title('mask_reverse')
plt.subplot(244), plt.imshow(img1background, 'gray'), plt.title('img1background')
plt.subplot(245), plt.imshow(img2foreground, 'gray'), plt.title('img2foreground')
plt.subplot(246), plt.imshow(end, 'gray'), plt.title('end')
plt.subplot(247), plt.imshow(img1, 'gray'), plt.title('endAdd')
```

Şekil 3.7: Maskeleme işlemi.

Son olarak bu kısımda ise aritmetik işlemler test ediliyor.



Şekil 3.8: Aritmetik işlem test çıktısı.

Aslında böylece iki resim birleştirilmiş olunuyor(foreground-background olarak).

Resim_okuma_yazma.py dosyası içerisinde resim okuma ve yazma işlemleri gerçekleştirildi.

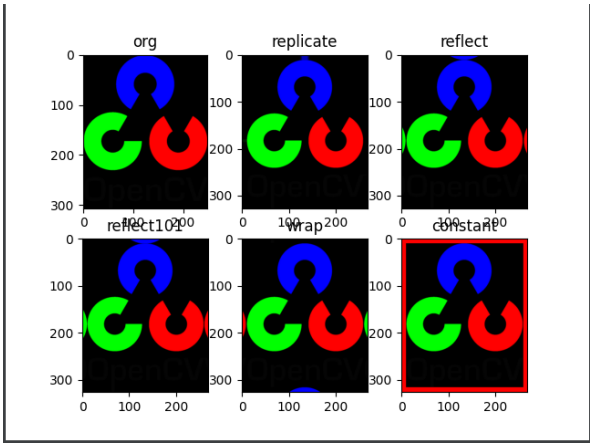
```
# image = cv2.imread('image.png', 0)  # image okuma methodu //0 parametresi gri tonlama için kullanıl
image = cv2.imread('image.png')
# cv2.imshow('PNG', image)  # show methodu
cv2.imwrite('gray.png', image)  # image'ı kaydetme methodu

print(str(image.item(100, 100, 0)))  # (100,100) aralığında //0 mavi, 1 yeşil , 2 kırmızı piksel değerini döndürür
image.itemset((100, 100, 1), 0)  # 100-100 noktasında mavi kırmızı veya yeşil değerini değiştirir.
print(
    str(image.shape))  # eğer resim renkliyse renk parametresi de gelir(3 parametre ),# eğer griyse renk parametresi Y
print(str(len(image.shape)))  # //len methoduyla gri mi renkli mi olduğu anlaşılabilir.(3 mü 2 mi?)
print(
    str(image.size))  # kaç pixel olduğunun bilgisini verir # 3ton olduğu için(kırmızı, mavi, yeşil)pixel size renkli
print(image.dtype)  # veri tipi

ROI = image[210:290, 172:230]  # [y,x] parametrelerin yerlerine dikkat!!
image[210:290, 72:130] = ROI  # pixel eşitleme//roideki pixelleri buraya yapıştır.
b, g, r = cv2.split(image)  # renklere ayrıştırma
cv2.merge((b, g, r))  # ayrıştırılan renkleri tekrar mergeleme

blue = image[:, :, 0]  # renk erişimi bu şekilde de yapılabilir.0=blue,1=green,2=red
image[:, :, 0] = 255
cv2.imshow('cut', image)
cv2.waitKey(0)  # kullanıcı bir tuşa basana kadar bekle
```

Şekil 3.9: Resim okuma yazma işlemi.

Resim_çerçeveleme.py dosyası içerisinde resim çerçeveleme işlemi gerçekleştirildi.

```
image = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)

replicate = cv2.copyMakeBorder(image, 10, 10, 10, 10,
                               cv2.BORDER_REPLICATE)  # It replicates the last element. Suppose, if image contains le
reflect = cv2.copyMakeBorder(image, 10, 10, 10, 10,
                             cv2.BORDER_REFLECT)  # The border will be mirror reflection of the border elements. Supp
reflect101 = cv2.copyMakeBorder(image, 10, 10, 10, 10,
                                cv2.BORDER_REFLECT101)  # It does the same works as cv2.BORDER_REFLECT but with sligh
wrap = cv2.copyMakeBorder(image, 10, 10, 10, 10,
                          cv2.BORDER_WRAP)  # this reflects the pixel from the opposite boundary as cdefgh|abcdefgh|a
constant = cv2.copyMakeBorder(image, 10, 10, 10, 10, cv2.BORDER_CONSTANT,
                              value=blue)  # It adds a constant colored border. The value should be given as a keywor
```

Şekil 3.10: Resim çerçeveleme işlemi.



Şekil 3.11: Resim çerçeveleme işleminin çıktısı.

Resim_format_degistirme.py dosyası içerisinde format değiştirme işlemi yapıldı.

```
def save(path, image, jpeg_quality=None, png_compress=None):
    if jpeg_quality:
        cv2.imwrite(path, image,
                    [int(cv2.IMWRITE_JPEG_QUALITY), jpeg_quality])  # jpg kalitesi 0-100 arasında 100 max kalite
    elif png_compress:
        cv2.imwrite(path, image, [int(cv2.IMWRITE_PNG_COMPRESSION),
                                  png_compress])  # sıkıştırma 0-9 arasında yüksek değer küçük boyut fazla sıkıştırma
    else:
        cv2.imwrite(path, image)
```

Şekil 2.12: Resim formatı değiştirme işlemi.

Buradaki 2 önemli nokta kaydetme biçimi:

- jpg kalitesi 0-100 arasında ve 100 maximum kalite.
- sıkıştırma 0-9 arasında yüksek değer küçük boyut fazla sıkıştırma demek.

3.2 Detaylı Maskeleme

Detaylı maskeleme işlemi altında altı farklı yapı kullanıldı.

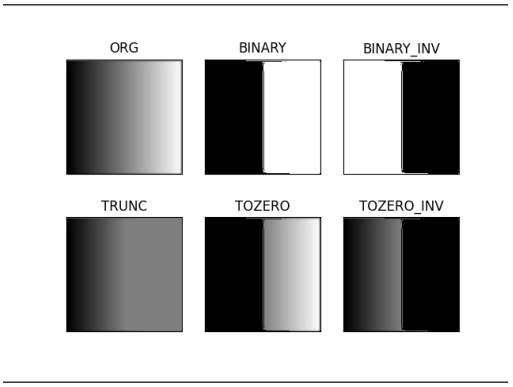
- Basit Eşik Değerle Maskeleme
- Blur ve Renk Filtreleme
- Canny
- Gauss-Mean Adaptif Filtreleme
- Gradyan-Canny
- Otsu Filtreleme

Basic_thresholding.py altında basit eşik değerle maskeleme işlemi yapıldı.

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

img = cv2.imread('gradient.JPG')

_, threshold1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
_, threshold2 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)
_, threshold3 = cv2.threshold(img, 127, 255, cv2.THRESH_TRUNC)
_, threshold4 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO)
_, threshold5 = cv2.threshold(img, 127, 255, cv2.THRESH_TOZERO_INV)

titles = ['ORG', 'BINARY', 'BINARY_INV', 'TRUNC', 'TOZERO', 'TOZERO_INV']
imgs = [img, threshold1, threshold2, threshold3, threshold4, threshold5]

for i in range(6):
    plt.subplot(2, 3, i + 1), plt.imshow(imgs[i], 'gray'), plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```

Şekil 3.13: Basit eşik değerle maskeleme işlemi.

Bu işlemde farklı threshold metodlarıyla farklı sonuçlar elde edilebilir.



Şekil 3.14: Farklı threshold metodlarının kullanımıyla elde edilen çıktılar.

Blur.py ve color_filtering.py dosyaları içerisinde blur ve renk filtreleme yapıldı.

```python
import cv2
import numpy as np

cam = cv2.VideoCapture(0)

while True:
    ret, frame = cam.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_white = np.array([0, 0, 140])
    upper_white = np.array([256, 60, 256])

    mask = cv2.inRange(hsv, lower_white, upper_white)
    end_frame = cv2.bitwise_and(frame, frame, mask=mask)

    kernel = np.ones((15, 15), np.float32) / 255
    blur = cv2.filter2D(end_frame, -1, kernel)
    gaussian_blur = cv2.GaussianBlur(end_frame, (15, 15), 0)
    median = cv2.medianBlur(end_frame, 15)
    bilateral = cv2.bilateralFilter(end_frame, 15, 75, 75)

    cv2.imshow('end_frame', end_frame), cv2.imshow('blur', blur), cv2.imshow('gaussian_blur', gaussian_blur)
    cv2.imshow('median', median), cv2.imshow('bilateral', bilateral)
```

Şekil 3.15: Blur ve renk filtreleme işlemi.

Blur işlemi çoğu zaman eğer görüntüde gürültü fazlaysa kullanılır ve farklı renk aralıklarını bulanıklaştırmak da mümkündür.
Blur işleminde cvtColor() yaparken bu sefer BGR2GRAY değil BGR2HSV kullanılmıştır.

HSV nedir?
Hue(renk tonu), Saturation(doygunluk) and Value(değer) renk uzayıdır.
Ton => rengi, Doygunluk =>griliği, Değer =>parlaklığı temsil eder.
BGR veya RGB renk uzayına kıyasla daha iyi performans gösterir.
Hue(renk tonu) aralığı => [0,179]
Saturation(doygunluk) aralığı => [0,255]
Value(değer) aralığı => [0,255]'dir.

Daha sonradan numpy kütüphanesi ile bir array aralığı oluşturulmuştur. Bu aralık aslında bu örnek için beyaz tonu filtrelemek için gereken değer aralığını kapsamaktadır. Başka renkler için bu değerler değişir.

```python
low_red = np.array([150, 30, 30])
up_red = np.array([190, 255, 255])
mask_red = cv2.inRange(hsv, low_red, up_red) # hsv'ye dönüştürülmüş
# cv2.imshow('mask_red', mask_red)
end_red = cv2.bitwise_and(frame, frame, mask=mask_red)
# cv2.imshow('end_red', end_red)
```

```python
low_blue = np.array([100, 60, 60])
up_blue = np.array([140, 255, 255])
mask_blue = cv2.inRange(hsv, low_blue, up_blue)
# cv2.imshow('mask_blue', mask_blue)
end_blue = cv2.bitwise_and(frame, frame, mask=mask_blue)
# cv2.imshow('end_blue', end_blue)
```

Şekil 3.16 ve 3.17: Numpy ile array aralığının oluşturulması ve beyaz tonun filtrelenmesi işlemi.

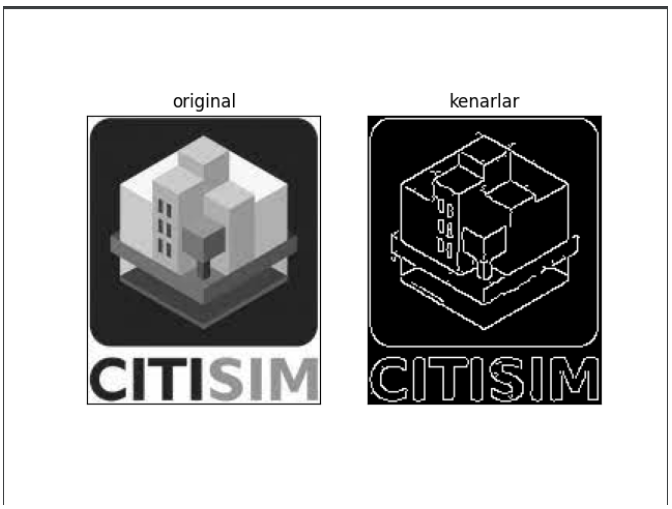
cv2.inRange()-> methodunda aslında lower ve upper white olarak tanımladığımız değerler, hsv aralığındaki karşılıklarıyla eşleştirilir ve bir mask oluşturulmuş olur.Bu mask bitwise_and() işlemi ile frame üzerine filtrelenir.

Blur işlemi için ise farklı farklı blur methodları mevcuttur. Bu noktadan sonrakı aslında geri kalan işlem bizim istediğimiz sonuçları vericek şekilde bir blur işlemini yapabilecek methodu seçmektir.
Özellikle blur işlemi içerisinde bahsedilmesi gereken son konu da kerneldir. Kernel dediğimiz yapı aslında 1'lerden oluşan bir matristir. Buradaki olay resimdeki piksellerin konvülasyonel şekilde bu kernelle çarpılıp yeni değerler elde ederek blurlanma işlemidir.

Canny.py dosyası içerisinde kenar bulma işlemi yapıldı.

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

img = cv2.imread('image1.png', 0)
canny = cv2.Canny(img, 50, 200)
plt.subplot(121), plt.imshow(img, cmap='gray'), plt.title('original'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(canny, cmap='gray'), plt.title('kenarlar'), plt.xticks([]), plt.yticks([])

plt.show()
```

Şekil 3.18: Kenar bulma işlemi için kullanılan kod parçası.

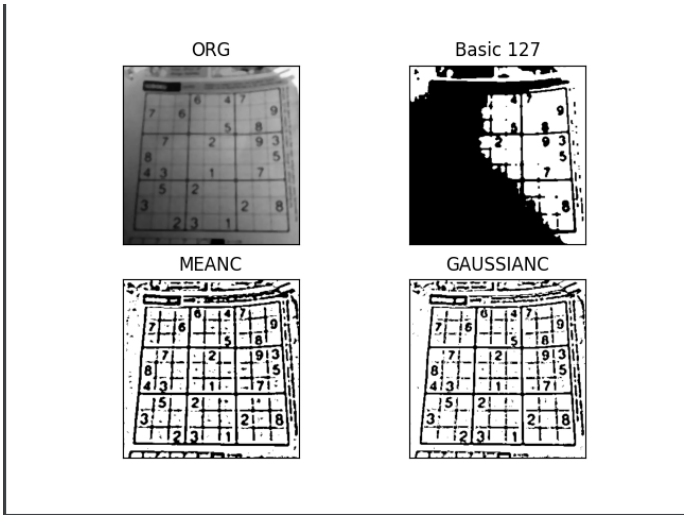


Şekil 3.19: Kenar bulma işlemi örnek üzerinde.

Gauss_mean_adaptif_filtering.py dosyası içerisinde Gauss-Mean Adaptif filtreleme işlemi yapıldı.

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

img = cv2.imread('sudoku.JPG', 0)
img = cv2.medianBlur(img, 5)

_, threshold = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
adaptiveThresholdMeanC = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2) # orta
adaptiveThresholdGaussianC = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)

titles = ['ORG', 'Basic 127', 'MEANC', 'GAUSSIANC']
imgs = [img, threshold, adaptiveThresholdMeanC, adaptiveThresholdGaussianC]

for i in range(4):
    plt.subplot(2, 2, i + 1), plt.imshow(imgs[i], 'gray'), plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```

Şekil 3.20: Gauss-Mean Adaptif filtreleme işlemi kod parçacığı.



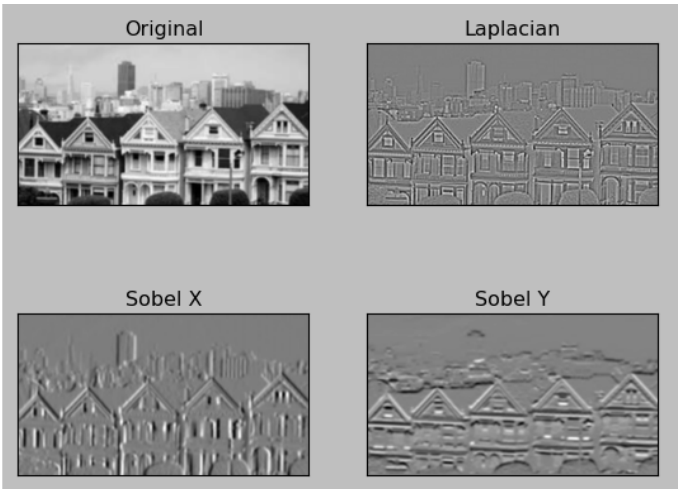
Şekil 3.21: Gauss-Mean Adaptif filtreleme işlemi örneği.

Gradyan_canny.py dosyası içerisinde laplacian, sobelX ve sobelY filtreleme işlemlerinden elde edilen görüntüde Canny ile kenar bulunuyor.

```python
cam = cv2.VideoCapture(0)

while (1):
    ret, frame = cam.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    low_white = np.array([0, 0, 140])
    up_white = np.array([256, 60, 256])

    laplacian = cv2.Laplacian(frame, cv2.CV_64F)
    sobelX = cv2.Sobel(frame, cv2.CV_64F, 1, 0, ksize=5)
    sobelY = cv2.Sobel(frame, cv2.CV_64F, 0, 1, ksize=5)

    mask_white = cv2.inRange(hsv, low_white, up_white)
    # cv2.imshow('mask_white', mask_white)
    end_white = cv2.bitwise_and(frame, frame, mask=mask_white)
    # cv2.imshow('end_white', end_white)

    canny = cv2.Canny(frame, 100, 200)

    cv2.imshow('laplacian', laplacian)
    cv2.imshow('sobelX', sobelX)
    cv2.imshow('sobelY', sobelY)
```

Şekil 3.22: Gradyan-Canny işlemi.

Sobel fonksiyonun 3. ve 4. parametresi dx ve dy'dir. Eğer dx'e 1, dy'ye 0 verirsek bu horizontal (yatay) kenarları algılayacak. dx'e 0, dy'ye 1 verirsekte vertical (dikey) kenarları algılar, ksize ise kenarlaştırma oranıdır.

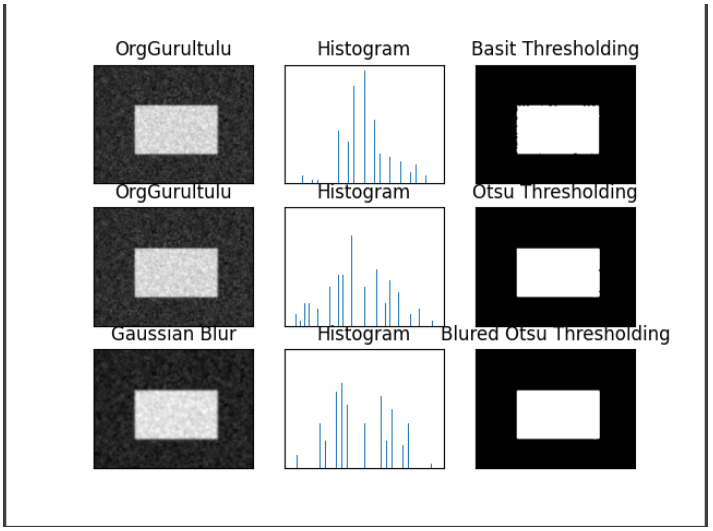


Şekil 3.23: Gradyan-Canny işlemi.

Otsu.py dosyası içerisinde Otsu Filtreleme işlemi yapıldı.

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

img = cv2.imread('gurultuluresim.JPG', 0)

_, basicThreshold = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
_, otsuThreshold2 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

blur = cv2.GaussianBlur(img, (5, 5), 0)
_, otsuBlurThreshold2 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

imgs = [img, 0, basicThreshold,
        img, 0, otsuThreshold2,
        blur, 0, otsuBlurThreshold2]

titles = ['OrgGurultulu', 'Histogram', 'Basit Thresholding',
          'OrgGurultulu', 'Histogram', 'Otsu Thresholding',
          'Gaussian Blur', 'Histogram', 'Blured Otsu Thresholding']
```

Şekil 3.24: Otsu Filtreleme işlemi kod parçacığı.



Şekil 3.25: Otsu Filtreleme işlemi örneği.

3.3 Morfolojik İşlemler

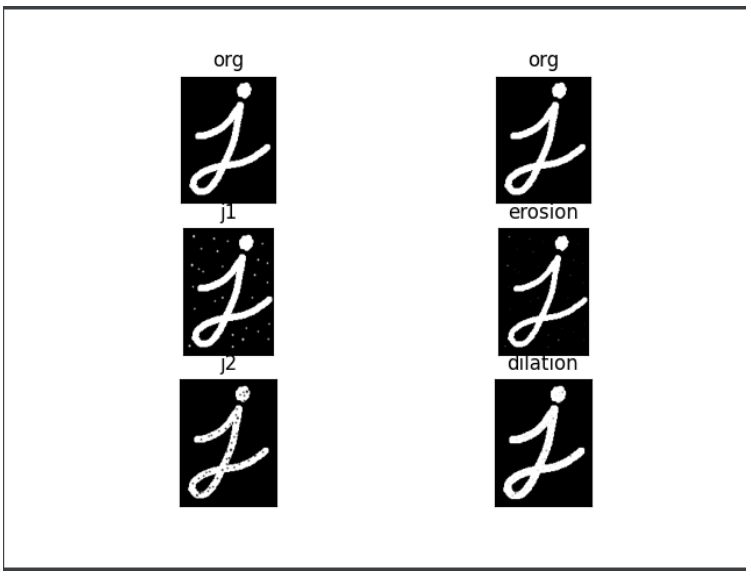
Morfolojik İşlemler işleminin altında iki ana işlem gerçekleştirildi. Bunlar:

- Erasion-Dilation Filtreleme
- Açık-Kapalı Filtreleme

Erasion_dilation.py dosyasının altında gerekli filtreleme işlemleri gerçekleştirildi.

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

j = cv2.imread('j.png')
j1 = cv2.imread('j2.png')
j2 = cv2.imread('j3.png')

kernel = np.ones((2, 2), np.uint8) / 255

erosion = cv2.erode(j1, kernel, iterations=1)
dilation = cv2.dilate(j2, kernel, iterations=1)

plt.subplot(321), plt.imshow(j), plt.title('org'), plt.xticks([]), plt.yticks([])
plt.subplot(322), plt.imshow(j), plt.title('org'), plt.xticks([]), plt.yticks([])
plt.subplot(323), plt.imshow(j1), plt.title('j1'), plt.xticks([]), plt.yticks([])
plt.subplot(324), plt.imshow(erosion), plt.title('erosion'), plt.xticks([]), plt.yticks([])
plt.subplot(325), plt.imshow(j2), plt.title('j2'), plt.xticks([]), plt.yticks([])
plt.subplot(326), plt.imshow(dilation), plt.title('dilation'), plt.xticks([]), plt.yticks([])

plt.show()
```

Şekil 3.26: Erasion-Dilation Filtreleme işlemi kod parçacığı.

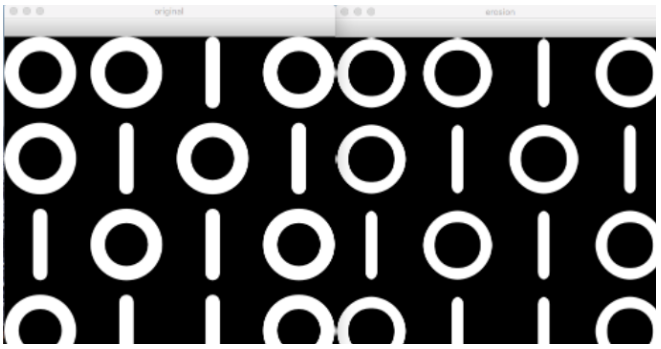


Şekil 3.27: Erasion-Dilation Filtreleme işlemi örneği.

Acik_kapali_filtreleme.py dosyası altında açık kapalı filtreleme işlemi yapıldı.

```python
import cv2
import numpy as np

cam = cv2.VideoCapture(0)

while True:
    _, frame = cam.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    lower_green = np.array([36, 0, 0])
    upper_green = np.array([86, 255, 255])

    mask = cv2.inRange(hsv, lower_green, upper_green)
    end_frame = cv2.bitwise_and(frame, frame, mask=mask)

    kernel = np.ones((5, 5), np.float32) / 25

    opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN,
                               kernel=kernel)  # filtrede uymayan kısımları(gürültü) belirginleştirir.
    cv2.imshow('opening', opening)
    closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel=kernel)  # gürültüyü kapatır.
    cv2.imshow('closing', closing)
```

Şekil 3.28: Açık Kapalı Filtreleme işlemi kod parçacığı.



Şekil 3.29: Erosion örneği.

Şekil 3.30: Dilation örneği.

4. TELLO DRONE İLE GÖRÜNTÜ İŞLEME

Drone kamerasından alınan görüntüler üzerinde basit obje tespiti yapılması ve tek bir obje takibi yapılması adımları bu aşamada işlenmiştir.

İlk olarak drone üzerinden resmi çekilen bir nesnenin takibi gerçekleştirildi.

```
import cv2
import numpy as np
from djitellopy import tello

#####################################################################
width = 640  # WIDTH OF THE IMAGE
height = 480  # HEIGHT OF THE IMAGE
deadZone = 100
#####################################################################

startCounter = 0

# CONNECT TO TELLO
me = tello.Tello()
me.connect()
me.for_back_velocity = 0
me.left_right_velocity = 0
me.up_down_velocity = 0
me.yaw_velocity = 0
me.speed = 0

print(me.get_battery())
```

Şekil 4.1: Çekilecek resmin boyutlarının ayarlanması ve dronun hareketsiz kalması işlemi.

Drone üzerinden görüntü almaya başlandı ve frame aralıkları belirlendi.

```
def empty(a):
    pass


cv2.namedWindow("HSV")
cv2.resizeWindow("HSV", 640, 240)
cv2.createTrackbar("HUE Min", "HSV", 20, 179, empty)
cv2.createTrackbar("HUE Max", "HSV", 40, 179, empty)
cv2.createTrackbar("SAT Min", "HSV", 148, 255, empty)
cv2.createTrackbar("SAT Max", "HSV", 255, 255, empty)
cv2.createTrackbar("VALUE Min", "HSV", 89, 255, empty)
cv2.createTrackbar("VALUE Max", "HSV", 255, 255, empty)

cv2.namedWindow("Parameters")
cv2.resizeWindow("Parameters", 640, 240)
cv2.createTrackbar("Threshold1", "Parameters", 166, 255, empty)
cv2.createTrackbar("Threshold2", "Parameters", 171, 255, empty)
cv2.createTrackbar("Area", "Parameters", 1750, 30000, empty)
```

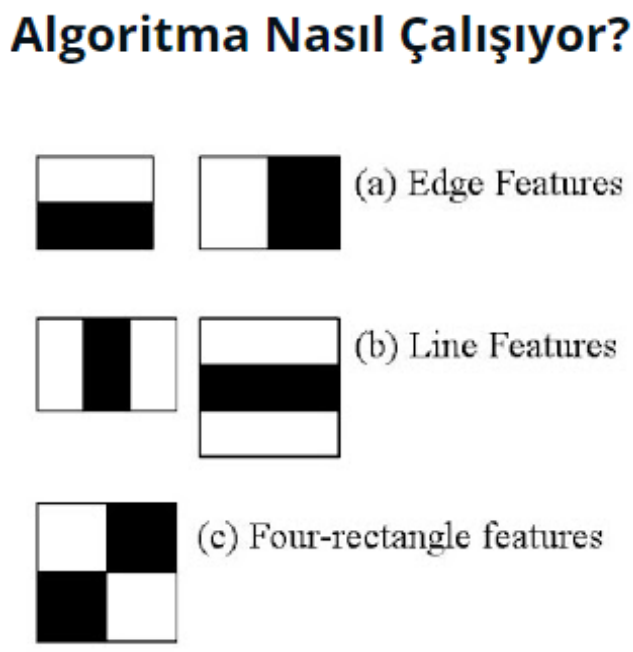
Şekil 4.2: Görüntü alınma işlemi.

Empty() metodunda ise hsv alt üst sınır değerlerini, alınacak resmin thresholdlarını ve kaplayabileceği max alanı atadık.

StackImages() methodunda drone'un resmin etrafında toplam 360 derece dönecek şekilde fotoğrafların alınması sağğlandı ve bir arrayde bu değerler tutuldu.
Daha sonradan OpenCV kısmında anlatıldığı birçok filtreleme yöntemini kullanarak alınan resimler filtrelendi ve maskelendi. Bu kısımda bir öğrenme kullanılmadı. Drone üzerinden yakalanan resimlerden birine denk düşecek şekilde bi an yakaladığında objeyi takip etmesi sağlandı.

4.1 Drone ile İnsan Takibi

Haar Cascade, Paul Viola ve Michael Jones tarafından 2001 yılında yayınlanan makine öğrenmesi nesne algılama algoritmasıdır. Birçok pozitif yani algılanacak nesnenin bulunduğu yerler ve negatif yani nesnenin bulunmadığı görüntüden eğitildiği makine öğrenimi tabanlı bir yaklaşımdır. OpenCV kütüphanesi bize, eğitildikleri görüntülere bağlı olarak kategoriler (yüzler, gözler vb.) halinde düzenlenmiş önceden eğitilmiş Haar Cascade algoritmalar sunar. [6]



Şekil 4.3: Haar Cascade Algoritması çalışma şekli.

Haar cascade, yukarıdaki filtreleri kullanarak görüntülerden öznitelik çıkarmaktadır. Bu filtrelere "Haar özellikleri" denir.

Filtre özniteliğin çıkarılacağı görsel üzerinde adım adım ilerleyerek beyaz ve siyah kısımların piksel yoğunlukları toplanır. Sonrasında bu iki yoğunluğun toplanıp

çıkarılmasıyla oluşan değer özniteliğin değeridir. Sonuç ne kadar yüksekse (yani siyah ve beyaz toplamlar arasındaki fark), o pencerenin ilgili bir özellik olma olasılığı o kadar yüksek olur.

Cascade sınıflandırma, tüm özellikleri bir pencereye uygulamak yerine, özellikleri sınıflandırıcıların farklı aşamalarına göre gruplandırıyor ve tek tek uyguluyor. Bir pencere başarısız olursa (beyaz ve siyah toplamlar arasındaki fark düşüktür) ilk aşama (normalde birkaç özellik içerir), algoritma onu atar: üzerinde kalan özellikleri dikkate almaz. Geçerse algoritma, özelliklerin ikinci aşamasını uygular ve işleme devam eder.[6]

Bu çalışmamızda HaarCascade algoritmasını kullanarak yüz tespitini gerçekleştirdik.

```python
from Utils import *
import cv2
import time

w, h = 360, 240
pid = [0.5, 0.5, 0]
pError = 0
startCounter = 0

drone = initializeTello()

drone.takeoff()
drone.send_rc_control(0, 0, 25, 0)
time.sleep(2.2)

while True:
    # if startCounter == 0:
    # startCounter = 1
    img = telloGetFrame(drone, w, h)
    img, info = findFace(img)
    pError = trackFace(drone, info, w, pid, pError)
    cv2.imshow('Image', img)
    if cv2.waitKey(1) and 0xFF == ord('q'):
        drone.land()
```

Şekil 4.4: Haar Cascade algoritması kod parçacığı.

```python
def findFace(img):
    faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(imgGray, 1.2, 4)

    faceListC = []
    faceListArea = []

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cx = x + w // 2
        cy = y + h // 2
        area = w * h
        cv2.circle(img, (cx, cy), 5, (0, 0, 255), cv2.FILLED)
        faceListC.append([cx, cy])
        faceListArea.append(area)

    if len(faceListArea) != 0:
        i = faceListArea.index(max(faceListArea))
        return img, [faceListC[i], faceListArea]
    else:
        return img, [[0, 0], 0]
```

Şekil 4.5: Haar Cascade algoritması kod parçacığı.

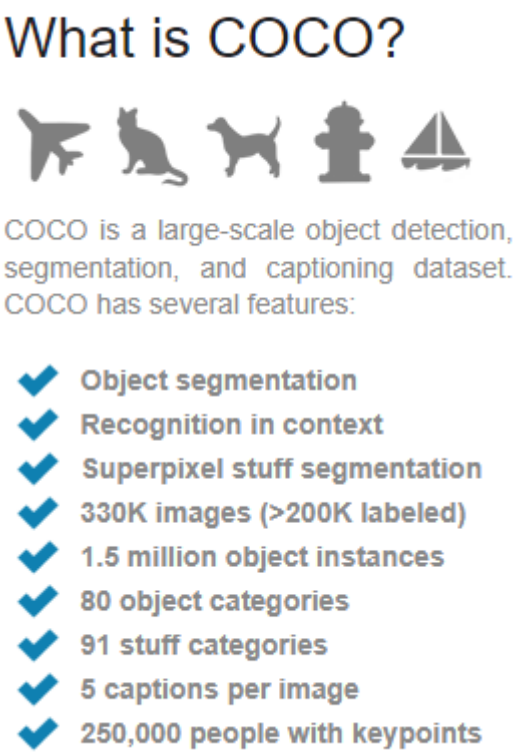
```python
def trackFace(drone, info, w, pid, pError):
    area = info[1]
    x, y = info[0]
    fb = 0

    error = x - w // 2
    speed = pid[0] * error + pid[1] * (error - pError)
    speed = int(np.clip(speed, -100, 100))
    if area > fbRange[0] and area < fbRange[1]:
        fb = 0
    elif area > fbRange[1]:
        fb = -20
    elif area < fbRange[0] and area != 0:
        fb = 20
    if x == 0:
        speed = 0
        error = 0

    drone.send_rc_control(0, fb, 0, speed)
    return error
```

Şekil 4.6: Haar Cascade algoritması kod parçacığı.

Yukarıdaki şekillerde verilen Haar Cascade algoritması kod parçacıklarında findFace() methoduyla Cascade Classifier'ı çağırılıyor. Eğer bir yüz tespit edilirse dikdörtgen içine alıyor ve ekrana bu yansıtılıyor. Cx,cy aslında yüzün bulunduğu noktadan itibaren orta noktasını bulup dikdörtgeni doğru bir şekilde çizdirmek için kullanılıyor. trackFace() metodu ile ise dikdörtgeni takip edilecek, daha doğrusu drone'un görüş açısında kalacak şekilde takibi sağlanıyor.

4.2. Drone ile Çoklu Nesne Tespiti

MobileNet-SSD (Single Shot Detector) v2: Gerçek zamanlı nesne tespitinde yaygın kullanılan bir modeldir. Modele "Single Shot" denilmesinin nedeni, nesne tespitinin ve sınıflandırmanın tek bir ileri yönlü hesaplamada olmasıdır. Ayrıca inception tarzında iç içe bloklardan oluşan çoklu-kutu (multibox) üzerinden sınıflandırma yapabilmesi SSD'nin önemli özelliklerindendir. SSD ağı taban olarakönerilen ve transfer öğrenimi çalışmalarında yoğun olarak kullanılan VGG-16 ağını kullanmaktadır. Fakat, VGG-16 gerçek zamanlı nesne tespiti için elverişli olmadığı için real-time çalışmasında önerilen MobileNet-V2 dir. MobileNet, ag tasarımında kullanılan ayrıştırılabilir evrişimsel katmanı (separable convolutional layer) ile hesaplama yükünü önemli derecede azaltırken, baharım olarak parametre açısından büyük ölçekli diger ağlarla aynıdır. Kayıp fonksiyonu olarak sınıflandırmada softmax kullanılırken yerelleştirme (localization) için Huber kaybı kullanılmıştır. [7]



Şekil 4.7: COCO nedir.

```
# cap.set(4, 480)  # set height #bu kisim drone kullanilirken silinece
thres = 0.6
nmsThres = 0.2

classNames = []
classFile = 'coco.names'

with open(classFile, 'rt') as f:
    classNames = f.read().split('\n')
print(classNames)
configPath = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
weightsPath = "frozen_inference_graph.pb"

net = cv2.dnn_DetectionModel(weightsPath, configPath)
net.setInputSize(320, 320)
net.setInputScale(1.0 / 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
```

Şekil 4.8: Nesne tanıma işlemi.

Threshold ve multiple object threshold değerleri atandı. Weight ve config yolları atadıktan sonra model cv2.dnn_DetectionModel içerisine verildi.

5. SONUÇLAR VE ÖNERİLER

Yapılan çalışmada, öncelikle farklı veri setleri denendi ve en uygunu seçilmeye çalışıldı. İlk adım olarak Drone ile bağlantı kurulması aşaması denendi bu süreçte farklı kütüphaneler ve fonksiyonlar denendi. Sonrasında ise ilk olarak insan tespiti yapılmaya çalışıldı. İnsan yüzünün tespiti kısmında en uygun olarak Haar Cascade Algoritması seçildi.

Bu şekilde yüz tespitini gerçekleştirdik sonrasında ise Tello Drone ile yüz takibi yapılmaya başlandı.

Object Detection için COCO veri setinden yararlanılarak CNN model ile object detection gerçekleştirdik. Real-Time Object Detection yaparken karşılaştığımız en büyük problem drone üzerinde büyük kütüphanelerin(YoloV4-YoloV3 ve YoloV3tiny)fps kayıplarına sebebiyet vermesiydi. Bunun için daha minimalistik yaklaşımlarda bulunarak OpenCV kullanarak eğitilmiş modeller üzerinde çalıştık ve kendimiz de bir model oluşturup eğitip entegre ettik.

Tüm bu çalışmaların sonunda hem insan takibi yapabilen hem de yaklaşık 30 farklı nesneyi %70-%80 doğrulukla tespit edebilen bir drone hazırlamış bulunduk.Görüntü işleme, nesne tespiti ve drone hareket kabiliyeti fonksiyonları hakkında da en azından yazılım düzeyinde fikir sahibi olmuş olduk.

KAYNAKLAR

[1] Görüntü İşleme,İ. Ü. Diş Hekimliği Fakültesi Dergisi Cilt : 27, Sayı: 4, Aralık 1993,Adalet Erdem Aytan ,Yusuf Öztürk, Emin Kazım Örgev'

[2] Baxes GA: Digital Image Processing: A practical Primec, Cascade Press Denver, 1984.

[3] Robert J. Schalkoff: Digital Image Processing and Computer Vision, John Wiley & Sons, Inc, 1989.

[4] http://www.ibrahimcayiroglu.com/Sayfalar/Ders_GoruntuIsleme.aspx

[5] https://medium.com/türkiye/görüntü-işleme-tekniklerinde-yapay-zeka-kullanımı-24101616cc97

[6] https://spacing.itu.edu.tr/pdf/tansel.pdf

[7] https://ibrahimcanerdogan.com/2021/06/25/python-ile-yuz-tespiti-nasil-yapilir-haar-cascade-algoritmalari/

[8] Wayne Niblack: An Introduction to Digital Image Processing, Prentice Hall International, 1986.

[9] Erişti E. 2010. Görüntü İşlemede Yeni Bir Soluk, OPENCV, Akademik Bilişim'10 - XII. Akademik Bilişim Konferansı Bildirileri, 10-12 Şubat, Muğla.

[10] Bradski, G. and Kaehler, A., "Learning OpenCV: Computer Vision with the OpenCV Library", O'Reilly Media, Amerika Birleşik Devletleri, 16-17 (2008).

[11] http://opencv.willowgarage.com/wiki/

[12] INTEL CORPORATION: Intel researchers teach computers to 'read lips' to improve accuracy of speech recognition software. M2 Presswire, Coventry, Apr 28,2003, pg1.

[13] http://ilkeraksoy.net/opencv-nedir/

[14] https://ogrencidergisi.erzurum.edu.tr/Documents/KerimOzturketal-b9a26e2a-d330-41c3-99c6-a13be0629901.pdf

ÖZGEÇMİŞ

Feyza Demirel 9 Nisan 2000 tarihinde Ankara'da doğdu. 2014 yılında Nevzat Ayaz Anadolu Lisesine girdi ve 2018 yılında mezun oldu. 2018 yılında 19 Mayıs Üniversitesi Bilgisayar Mühendisliği bölümünü kazandı ve 2019 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümüne yatay geçiş yaptı. Burada hala öğrenimine devam etmektedir.

Yasin Ömer Kara 20 Eylül 1999 tarihinde İzmir'de doğdu. 2013 yılında Sakarya Cevat Ayhan Fen Lisesine girdi ve 2017 yılında mezun oldu. 2017 yılında Dokuz Eylül Üniversitesi Makine Mühendisliği bölümünü kazandı ve 2018 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümüne merkezi yatay geçiş yaptı. Burada hala öğrenimine devam etmektedir.