Teknoloji Fakültesi

MARMARA ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ 2024-2025 DÖNEMİ
BİLGİSAYAR MÜHENDİSLİĞİ
GÖMÜLÜ SİSTEMLER DERSİ PROJE RAPORU

Hazırlayanlar :

Safa Kılıçkaya - 170422044

Ahmet Önder Önlü - 170422030

Zaid Almoughrabi - 170422991

İçindekiler Tablosu

BİLGİSAYAR GÖRÜ TABANLI

KİŞİ SAYMA VE YÜZ TANIMA SİSTEMİ

PROJE BİLGİLERİ

Proje Adı: Bilgisayar Görü Uygulamaları - Kişi Sayma ve Yüz Tanıma Sistemi

Ders: Gömülü Sistemler

Geliştirme Dili: Python

Arayüz: Tkinter GUI

1. PROJE AMACI VE KAPSAMI

Bu proje, modern bilgisayar görü teknolojilerini kullanarak gerçek zamanlı kişi sayma ve yüz tanıma işlevlerini yerine getiren entegre bir sistem geliştirmeyi amaçlamaktadır. Proje, gömülü sistemler alanında yaygın olarak kullanılan görüntü işleme ve yapay zeka teknolojilerinin pratik uygulamalarını içermektedir.

1.1. Ana Hedefler

- Gerçek zamanlı kamera görüntüsü üzerinde kişi tespiti ve sayımı
- Önceden tanımlanmış kişilerin yüz tanıma ile belirlenmesi
- Kullanıcı dostu grafik arayüz ile sistem kontrolü
- GPU hızlandırması ile performans optimizasyonu
- Modüler yazılım mimarisi ile genişletilebilirlik

1.2. Uygulama Alanları

- Güvenlik sistemleri

- Akıllı bina yönetimi
- Katılım takip sistemleri
- Müşteri analiz sistemleri

2. KULLANILAN TEKNOLOJİLER VE KÜTÜPHANELER

2.1. Yapay Zeka ve Görüntü İşleme

- **YOLOv8n (Ultralytics):** Kişi tespiti için ultra-hafif nesne tanıma modeli olup, gerçek zamanlı uygulamalar için optimize edilmiştir. Yüksek doğruluk oranı ile düşük hesaplama maliyetini birleştirerek mobil ve gömülü sistemlerde etkili performans sağlar.

- **InsightFace:** Yüz tanıma için gelişmiş derin öğrenme kütüphanesi olup, state-of-the-art yüz embedding algoritmaları içerir. Yüksek doğruluk oranları ve hızlı işlem süreleri ile endüstriyel uygulamalarda yaygın olarak kullanılmaktadır.

- **OpenCV:** Bilgisayar görü uygulamaları için temel görüntü işleme ve kamera operasyonları sağlayan kapsamlı bir kütüphanedir. Gerçek zamanlı video işleme, görüntü manipülasyonu ve çeşitli görüntü formatları desteği ile projede kritik rol oynamaktadır.

- **scikit-learn:** Makine öğrenmesi algoritmaları içeren Python kütüphanesi olup, projede cosine similarity hesaplamaları için kullanılmaktadır. Yüz tanıma modülünde farklı yüz vektörleri arasındaki benzerlik oranlarını hesaplamak için optimize edilmiş matematiksel fonksiyonlar sağlar.

2.2. Arayüz ve Sistem

- **Tkinter:** Python'un standart GUI framework'ü olup, masaüstü uygulamaları için kullanıcı arayüzü oluşturmaya olanak sağlar. Cross-platform uyumluluğu ve Python ile entegre yapısı sayesinde hızlı prototipleme ve gelişim süreçlerinde tercih edilmektedir.

- **PIL (Pillow):** Python için güçlü bir görüntü işleme ve formatı dönüşüm kütüphanesi olup, çeşitli görüntü formatlarını destekler. Projede kamera görüntülerinin Tkinter arayüzünde gösterilmesi için format dönüşümleri ve boyutlandırma işlemleri gerçekleştirir.

- **NumPy:** Çok boyutlu array'ler ve matematiksel işlemler için optimize edilmiş Python kütüphanesi olup, bilimsel hesaplamalarda temel araçtır. Görüntü verilerinin matris olarak işlenmesi ve yüz embedding vektörlerinin matematiksel operasyonları için kritik önem taşır.

- **Threading:** Python'da paralel programlama için kullanılan modül olup, asenkron işlem yürütme imkanı sağlar. Kamera görüntülerinin işlenmesi sırasında kullanıcı arayüzünün donmaması için background thread'lerde video işleme görevlerini yürütür.

2.3. Veri Yönetimi ve Sistem Araçları

- **Pickle:** Python objelerinin binary formatta serilization işlemi için kullanılan standart kütüphanedir. Projede yüz embedding vektörlerinin disk üzerinde saklanması ve hızlı erişim için kullanılır, böylece uygulama her başlatıldığında embedding'lerin yeniden hesaplanması gerekmez.

- **Glob:** Dosya sistemi üzerinde pattern matching ile dosya arama işlemi yapan Python modülüdür. Proje klasörlerindeki resim dosyalarını otomatik olarak tarayarak farklı formatları (jpg, png, bmp) destekler ve batch işlem imkanı sağlar.

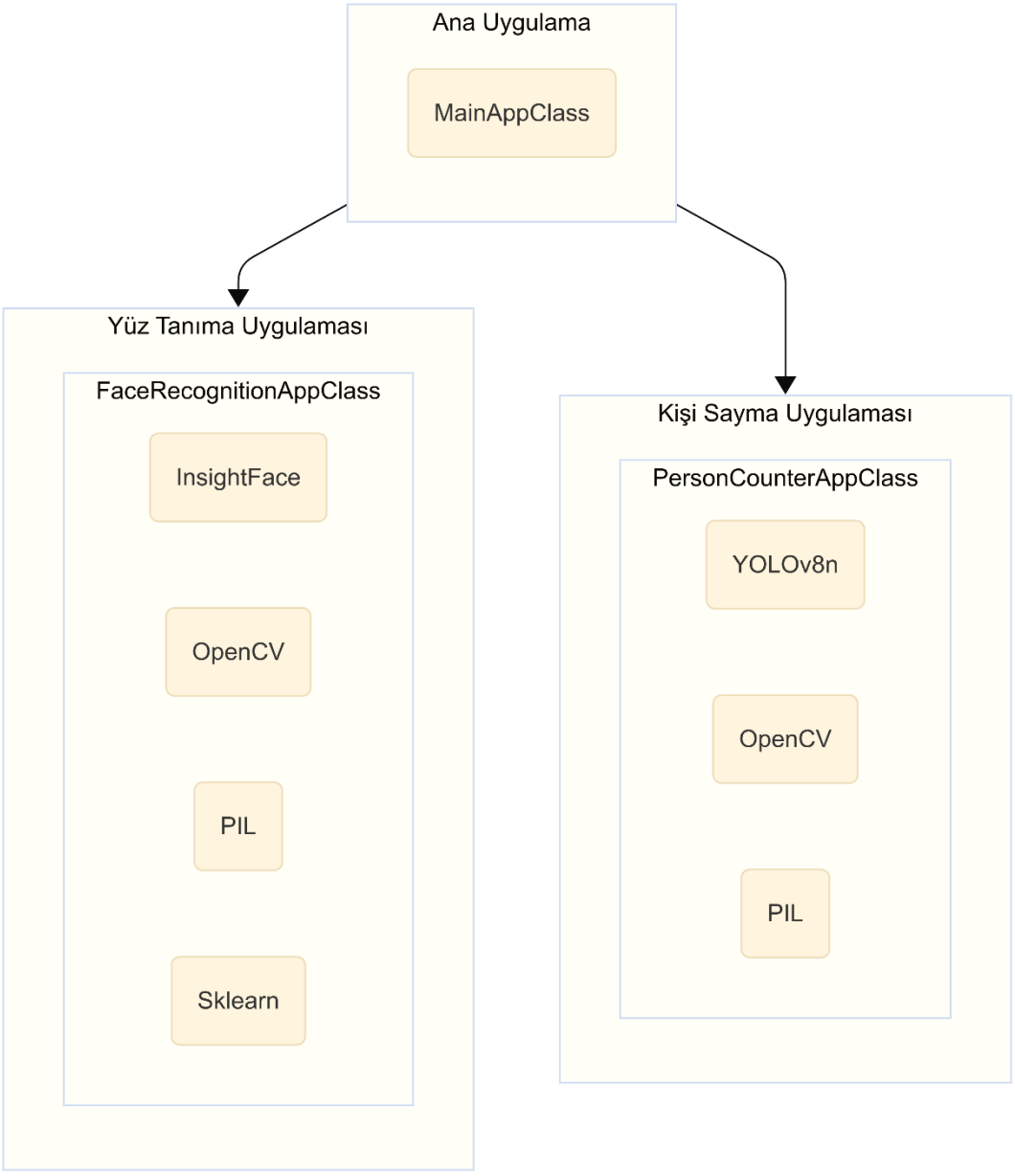
- **Time:** Zaman ölçümü ve gecikme işlemleri için kullanılan Python standart kütüphanesidir. FPS hesaplama, performans ölçümü ve frame işleme süreçlerindeki zamanlama kontrolü için kritik rol oynar.

2.3. Performans Optimizasyonları

- **CUDA GPU Desteği:** NVIDIA GPU hızlandırması
- **Frame Skipping:** Gereksiz frame işlemelerinin atlanması
- **Buffer Optimizasyonu:** Kamera buffer boyut ayarlamaları
- **Çoklu Provider Desteği:** CPU/GPU otomatik seçimi

3. SİSTEM MİMARİSİ VE MODÜLLER

Sistem, projenin ana hedeflerine ulaşmak için modüler bir mimari üzerine kurulmuştur. Bu yapı, farklı işlevsellikleri (ana kontrol, kişi sayma, yüz tanıma) ayrı modüllere ayırarak geliştirme, bakım ve genişletilebilirliği kolaylaştırır. Ana uygulama modülü (main_app.py), sistemin merkezi olup, kullanıcı arayüzünü yönetir ve kişi sayma (person_counter.py) ile yüz tanıma (face_recognition_app.py) modüllerini başlatır. Kamera girdisi bu modüller tarafından işlenir ve ilgili yapay zeka modelleri (YOLOv8n, InsightFace) kullanılarak tespit ve tanıma işlemleri gerçekleştirilir. İşlem sonuçları kullanıcı arayüzünde gösterilir ve yüz embedding verileri yönetilir. Sistemin genel mimari yapısı Şekil 1'de gösterilmektedir.



Şekil-1. Sistem mimari şeması

3.1. Ana Uygulama Modülü (main_app.py)

Sistemin giriş noktası ve ana menü kontrolcüsüdür. Bu modül ana menü arayüzünü oluşturur, alt uygulamaları başlatır ve yönetir, sistem bağımlılıklarını kontrol eder ve pencere yaşam döngüsünü yönetir.

MainApp Sınıfı Fonksiyonları:

__init__(self, root)

- Ana uygulama sınıfının kurucu fonksiyonu
- Tkinter root penceresini alır ve temel yapılandırmaları yapar
- Pencere boyutunu 800x600 olarak ayarlar
- Aktif pencere referansını (active_window) başlatır
- Ana GUI'yi oluşturmak için create_main_gui() fonksiyonunu çağırır

create_main_gui(self)

- Ana menü arayüzünün tüm görsel bileşenlerini oluşturur
- Başlık etiketi, uygulama butonları ve bilgi etiketlerini yerleştirir
- "Kişi Sayma" ve "Yüz Tanıma" butonlarını tasarlar
- Çıkış butonu ve sistem gereksinimleri hakkında bilgi etiketleri ekler
- Modern ve kullanıcı dostu bir arayüz tasarımı uygular

open_person_counter(self)

- Kişi sayma uygulamasını yeni bir pencerede başlatır
- Mevcut aktif pencereyi kapatır ve yeni Toplevel penceresi oluşturur
- Pencere kapanma olayını yönetmek için protokol ayarlar
- PersonCounterApp sınıfının bir örneğini oluşturur
- Hata durumlarını yakalayarak kullanıcıya bilgi verir

open_face_recognition(self)

- Yüz tanıma uygulamasını yeni bir pencerede başlatır
- open_person_counter ile benzer mantıkta çalışır
- FaceRecognitionApp sınıfının bir örneğini oluşturur
- Pencere yönetimi ve hata kontrolü sağlar

exit_app(self)

- Uygulamadan güvenli çıkış sağlar
- Kullanıcıdan onay alarak çıkış işlemini gerçekleştirir
- Aktif pencereleri kapatır ve kaynakları serbest bırakır
- Tkinter event loop'unu sonlandırır

Yardımcı Fonksiyonlar:

check_dependencies()

- Gerekli Python kütüphanelerinin kurulu olup olmadığını kontrol eder
- opencv-python, ultralytics, insightface, scikit-learn, numpy, Pillow kontrolü yapar
- Eksik kütüphaneler için kullanıcıya kurulum komutları önerir
- Boolean değer döndürerek sistem hazırlık durumunu belirtir

main()

- Uygulamanın başlangıç fonksiyonu
- Bağımlılık kontrolü yapar ve sistem durumunu raporlar
- Ana Tkinter penceresi oluşturur ve MainApp sınıfını başlatır
- Exception handling ile beklenmeyen hataları yakalar
- Keyboard interrupt (Ctrl+C) durumunu yönetir

Ana uygulama modülünün **create_main_gui** metodu ile oluşturulan arayüzün görseli Şekil 2'de gösterilmiştir.

Şekil-2. Ana sayfa ekran görüntüsü

3.2. Kişi Sayma Modülü (person_counter.py)

YOLOv8n modeli kullanarak gerçek zamanlı kişi tespiti ve sayımı yapar. Kamera görüntüsünden kişi tespiti, güven eşiği ayarlanabilir parametre, FPS sayacı ile performans gösterimi ve bounding box ile tespit edilen kişilerin işaretlenmesi özelliklerini içerir.

PersonCounterApp Sınıfı Fonksiyonları:

__init__(self, window)

- Kişi sayma uygulamasının kurucu fonksiyonu
- Tkinter penceresi, YOLO modeli ve kamera referanslarını başlatır
- FPS sayacı ve çalışma durumu değişkenlerini ayarlar
- GUI oluşturma ve model yükleme işlemlerini başlatır

create_gui(self)

- Kişi sayma uygulamasının grafiksel arayüzünü oluşturur
- Sol panel: kontrol butonları, ayarlar ve durum bilgileri
- Sağ panel: kamera görüntüsü için video frame
- Güven eşiği slider'ı ve FPS göstergesi ekler
- Modern ve işlevsel bir arayüz tasarımı uygular

load_model(self)

- YOLOv8n modelini yükler ve hazır hale getirir
- Model yükleme durumunu kullanıcı arayüzünde gösterir
- Hata durumlarında kullanıcıya bilgi verir
- Model başarıyla yüklendiğinde durum etiketini günceller

start_camera(self)

- Kamera cihazını başlatır ve konfigüre eder
- Kamera çözünürlüğü ve FPS ayarlarını optimize eder
- Video işleme thread'ini başlatır
- Buton durumlarını günceller ve durum bilgilerini gösterir

process_camera(self)

- Ana video işleme döngüsü (thread'de çalışır)
- Her frame'i okur ve YOLO modeli ile işler
- FPS hesaplama ve kişi sayma işlemlerini gerçekleştirir
- Tespit edilen kişiler için bounding box çizer
- İşlenen görüntüyü GUI'de gösterir

update_gui(self, imgtk, person_count)

- GUI'yi güncellemek için ana thread'de çalışır
- İşlenen görüntüyü ve kişi sayısını arayüze yansıtır
- Thread-safe GUI güncellemesi sağlar

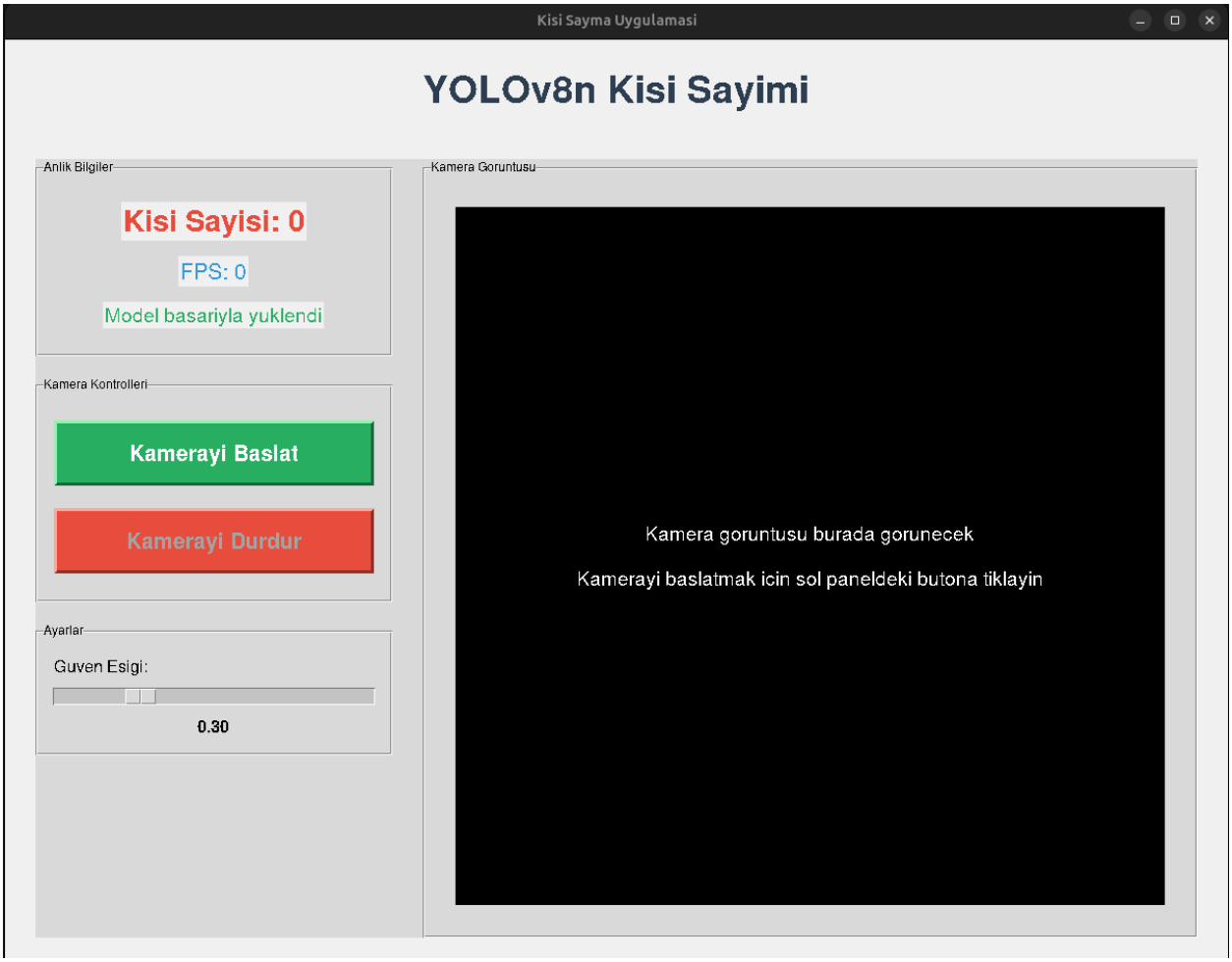
stop_camera(self)

- Kamera cihazını durdurur ve kaynakları serbest bırakır
- Video işleme thread'ini sonlandırır
- GUI elementlerini başlangıç durumuna getirir
- Buton durumlarını günceller

__del__(self)

- Sınıf destructor'ı - temizlik işlemleri
- Kamera kaynaklarını serbest bırakır
- OpenCV pencerelerini kapatır

Kişi sayma modülünün **create_gui** metodu ile oluşturulan arayüz ekran görüntüsü Şekil 3'de verilmiştir.



Şekil-3. YoloV8n ile kişi sayma uygulaması ekran görüntüsü

3.3. Yüz Tanıma Modülü (face_recognition_app.py)

InsightFace kütüphanesi ile gelişmiş yüz tanıma işlevselliği sağlar. Yüz tespiti ve embedding çıkarımı, önceden kaydedilmiş yüzlerle karşılaştırma, cosine similarity ile benzerlik hesaplama ve otomatik veri yönetimi özelliklerini içerir.

FaceRecognitionApp Sınıfı Fonksiyonları:

__init__(self, root)

- Yüz tanıma uygulamasının kurucu fonksiyonu
- Temel değişkenleri (kamera, embedding'ler, eşik değerleri) başlatır
- Veri yollarını ve performans parametrelerini ayarlar
- InsightFace kütüphanesi kontrol ederek GUI oluşturur
- Model yükleme ve embedding işlemlerini başlatır

create_gui(self)

- Yüz tanıma uygulamasının detaylı arayüzünü oluşturur
- Sol panel: kamera kontrolleri, ayarlar, veri yönetimi, durum bilgileri
- Sağ panel: gerçek zamanlı video görüntüsü
- Benzerlik eşiği slider'ı ve FPS göstergesi
- Veri klasörü seçimi ve embedding güncelleme butonları

load_models(self)

- InsightFace modellerini GPU optimizasyonlu olarak yükler
- CUDA ve CPU provider'ları arasında otomatik seçim yapar
- Model başarım durumunu kullanıcı arayüzünde gösterir
- Optimum tespit boyutu (320x320) ayarlar

load_or_create_embeddings(self)

- Mevcut yüz embedding'lerini yükler veya yenilerini oluşturur
- Pickle dosyasından kayıtlı embedding'leri okur
- Dosya bulunamazsa create_embeddings_from_data() çağırır
- Yüklenen kişi sayısını GUI'de gösterir

create_embeddings_from_data(self)

- Veri klasöründeki resimlerden yüz embedding'leri oluşturur
- Her kişi klasöründeki tüm resimleri işler
- Birden fazla resim varsa ortalama embedding hesaplar
- Oluşturulan embedding'leri pickle formatında saklar

recognize_face(self, face_embedding)

- Gelen yüz embedding'ini kayıtlı yüzlerle karşılaştırır

- Cosine similarity kullanarak benzerlik hesaplar
- Eşik değerini aşan en yüksek benzerliği döndürür
- Tanınmayan yüzler için "Unknown" etiketi verir

select_data_folder(self)

- Kullanıcıdan yeni veri klasörü seçmesini ister
- Dosya diyalogu açarak klasör seçim arayüzü sağlar
- Seçilen klasörü kayderir ve embedding'leri günceller

refresh_embeddings(self)

- Mevcut embedding'leri siler ve yeniden oluşturur
- Veri klasöründeki güncel resimlerle çalışır
- Manuel güncelleme için kullanıcı arayüzünden çağrılır

toggle_camera(self)

- Kamera durumunu kontrol ederek başlatır veya durdurur
- Buton durumuna göre uygun işlemi çağırır

start_camera(self)

- Kamera cihazını performans optimizasyonlu başlatır
- Kamera parametrelerini (çözünürlük, FPS, codec) ayarlar
- Video işleme thread'ini başlatır
- GPU destekli çalışma modunu etkinleştirir

stop_camera(self)

- Kamera cihazını durdurur ve kaynakları temizler
- Video thread'ini güvenli şekilde sonlandırır
- GUI durumunu başlangıç haline getirir

update_video(self)

- Ana video işleme döngüsü (thread'de çalışır)
- Frame okuma ve FPS hesaplama işlemleri
- Frame işleme ve GUI güncelleme koordinasyonu
- Hata durumlarında güvenli sonlandırma

process_frame(self, frame)

- Her video frame'ini GPU optimizasyonlu işler
- Frame atlama tekniği ile performans artırır
- Yüz tespiti ve tanıma işlemlerini gerçekleştirir
- Bounding box, isim etiketi ve güven skoru çizer

draw_previous_faces(self, frame)

- Frame atlama sırasında önceki frame bilgilerini kullanır
- Performans için aynı yüz bilgilerini tekrar çizer
- Tutarlı görüntü akışı sağlar

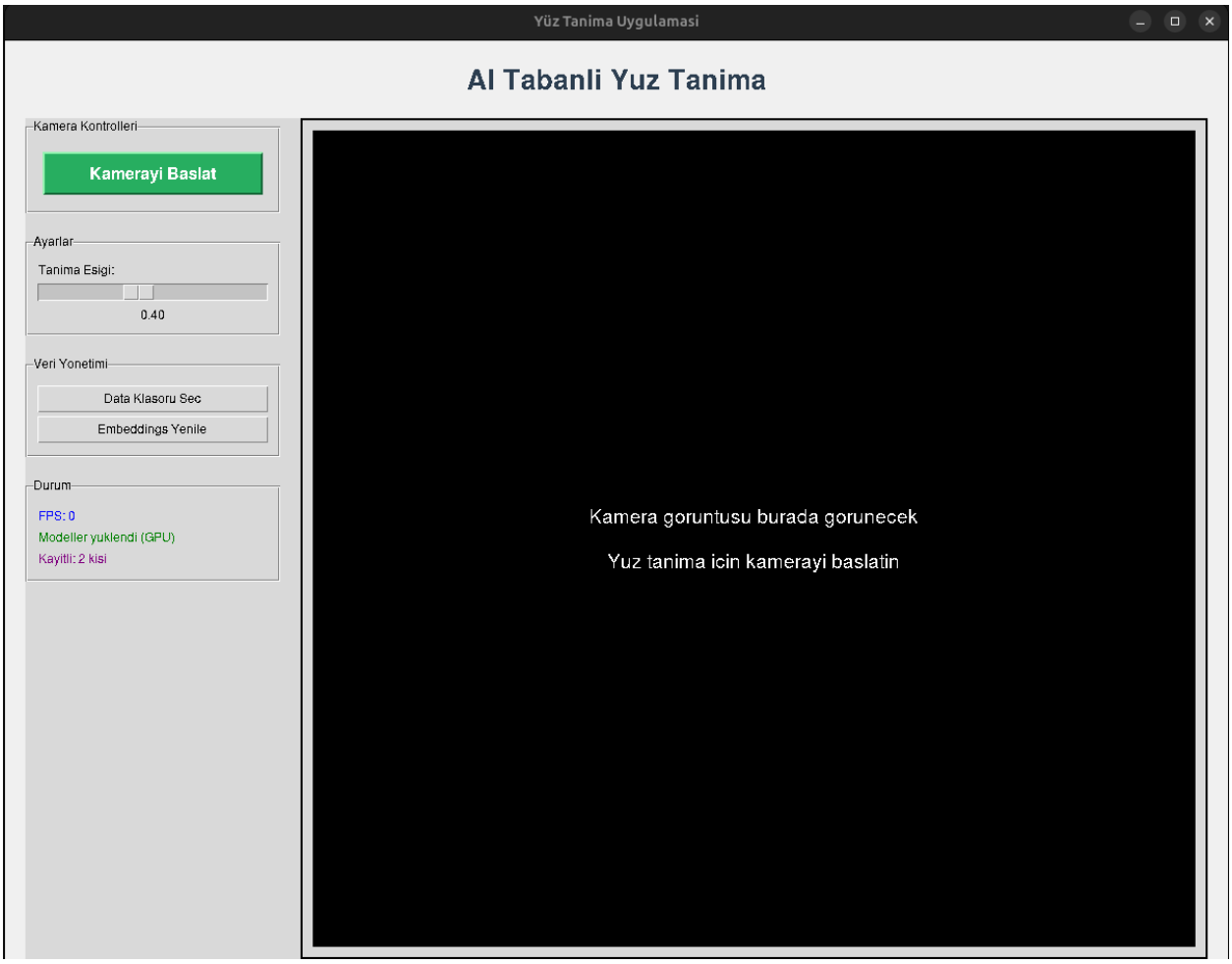
update_video_frame(self, photo)

- Ana thread'de GUI video frame'ini günceller
- Thread-safe görüntü güncellemesi sağlar
- PhotoImage referansını korur

__del__(self)

- Sınıf destructor'ı - kaynak temizliği
- Kamera kaynaklarını serbest bırakır
- OpenCV pencerelerini kapatır

Yüz tanıma modülünün **create_gui** metodu ile oluşturulan arayüz ekran görüntüsü Şekil 4'de verilmiştir.



Şekil-4. Yüz tanıma uygulaması

4. PERFORMANS OPTİMİZASYONLARI

4.1. GPU Hızlandırması

Sistem, CUDA destekli GPU'larda çalışacak şekilde optimize edilmiştir:

```
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
providers = ['CUDAExecutionProvider', 'CPUExecutionProvider']
```

4.2. Frame İşleme Optimizasyonları

- **Frame Skipping:** Her 2. frame işlenerek işlem yükü azaltılır
- **Resolution Scaling:** Tespit için küçük çözünürlük kullanılır

- **Buffer Management:** Kamera buffer boyutu minimize edilir

4.3. Memory Management

- Embedding vektörleri pickle formatında saklanır
- Gereksiz referanslar temizlenir
- Thread-safe programlama uygulanır

5. KULLANICI ARAYÜZÜ VE ETKİLEŞİM

5.1. Ana Menü

- Modern ve sade tasarım
- Büyük, kolay erişilebilir butonlar
- Sistem durumu göstergeleri
- Bağımlılık kontrol mesajları

5.2. Kişi Sayma Arayüzü

- Gerçek zamanlı kamera görüntüsü
- Anlık kişi sayısı göstergesi
- FPS performans sayacı
- Ayarlanabilir güven eşiği

5.3. Yüz Tanıma Arayüzü

- Çoklu yüz tespiti ve tanıma
- Benzerlik oranı gösterimi
- Veri klasörü yönetimi
- Embedding güncelleme araçları

6. VERİ YÖNETİMİ VE DEPOLAMA

6.1. Yüz Verisi Organizasyonu

```
data/
├── kisi1/
│   ├── foto1.jpg
│   ├── foto2.jpg
```

```
│   └── ...
├── kisi2/
│   ├── foto1.jpg
│   └── ...
```

6.2. Embedding Sistemi ve Organizasyonu

- Her kişi için ortalama embedding vektörü hesaplanır
- Pickle formatında binary olarak saklanır
- Otomatik güncelleme ve yenileme özelliği

```
embeddings/
├── face_embeddings.pkl
```

7. HATA YÖNETİMİ VE GÜVENİLİRLİK

7.1. Exception Handling

- Try-catch blokları ile hata yakalama
- Kullanıcı dostu hata mesajları
- Sistem durumu bildirimleri

7.2. Resource Management

- Kamera kaynaklarının güvenli serbest bırakılması
- Thread'lerin düzgün sonlandırılması
- Memory leak önleme

8. KURULUM VE ÇALIŞTIRMA

8.1. Gerekli Kütüphaneler

```
pip install opencv-python ultralytics insightface scikit-learn numpy Pillow
```

8.2. Sistem Gereksinimleri

- Python 3.8+

- OpenCV uyumlu kamera
- CUDA destekli GPU (opsiyonel)
- Minimum 4GB RAM

8.3. Çalıştırma

python main_app.py

9. PROJE SONUÇLARI VE DEĞERLENDİRME

9.1. Başarı Kriterleri

- ✓ Gerçek zamanlı kişi tespiti
- ✓ Yüz tanıma fonksiyonalitesi
- ✓ Kullanıcı dostu arayüz
- ✓ GPU performans optimizasyonu
- ✓ Modüler kod yapısı

9.2. Performans Metrikleri

- YOLOv8n: ~30 FPS (GPU'da)
- InsightFace: ~15 FPS (GPU'da)
- Yüz tanıma doğruluğu: >%90 (uygun koşullarda)

9.3. Geliştirme Önerileri

- Mobil cihaz desteği eklenmesi
- Web tabanlı arayüz geliştirme
- Veritabanı entegrasyonu
- REST API desteği

10. SONUÇ

Bu proje, gömülü sistemler alanında bilgisayar görü teknolojilerinin başarılı bir uygulamasını temsil etmektedir. YOLOv8n ve InsightFace gibi modern derin öğrenme modellerinin entegrasyonu ile yüksek performanslı, gerçek zamanlı bir görüntü işleme sistemi geliştirilmiştir.

Proje, teorik bilgilerin pratik uygulamaya dönüştürülmesi açısından değerli bir deneyim sağlamış ve gömülü sistemlerde AI/ML teknolojilerinin kullanımına dair önemli içgörüler sunmuştur.

KAYNAKLAR

1. Ultralytics YOLOv8 Documentation
2. InsightFace GitHub Repository
3. OpenCV Python Tutorials
4. Python Tkinter Documentation
5. CUDA Programming Guide