T.C.
KONYA TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ

LİSANS BİTİRME PROJE RAPORU

ESP32 KONTROLLÜ VE RASPBERRY PI DESTEKLİ GÖRÜNTÜ İŞLEME TABANLI YABANİ BİTKİ TESPİT ARACI

Hazırlayan

ÖMER FARUK ACAR

211222051

Danışman

Prof. Dr. Ahmet Afşin Kulaksız

KONYA-2025

T.C.

KONYA TEKNİK ÜNİVERSİTESİ

MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ

LİSANS BİTİRME PROJESİ

Hazırlayan

ÖMER FARUK ACAR

211222051

Bu çalışma …/…/2025 tarihinde komisyonumuz tarafından Elektrik Elektronik Mühendisliği Bölümü LİSANS BİTİRME PROJESİ olarak kabul edilmiştir.

Prof. Dr. Ahmet Afşin Kulaksız

Prof. Dr. Ramazan Akkaya

Dr. Öğr. Üyesi Emre Hasan Dursun

Prof. Dr. Seral Özşen

Elk. Elt. Müh. Bölüm Başkanı

İçindekiler

1.Giriş

Tarım sektörü, artan dünya nüfusu ve iklim değişikliği baskısı altında daha sürdürülebilir, verimli ve akıllı çözümlere ihtiyaç duymaktadır. Geleneksel tarımsal yöntemler; yüksek iş gücü gereksinimi, zaman kaybı ve insan hatası gibi nedenlerle modern ihtiyaçları karşılamakta yetersiz kalmaktadır. Bu bağlamda, Nesnelerin İnterneti (IoT), robotik sistemler ve yapay zekâ tabanlı görüntü işleme teknolojileri tarımda dijital dönüşümü hızlandırarak üretim sürecinin otomasyonuna olanak tanımaktadır.

Bu tez çalışmasında, IoT tabanlı web arayüzü üzerinden kontrol edilebilen, hareket kabiliyetine sahip kameralı bir mobil tarım robotu tasarlanmış ve geliştirilmiştir. Robot üzerindeki pan-tilt mekanizmasına entegre USB web kamerası, çevredeki bitki fidelerinin görüntülerini gerçek zamanlı olarak yakalamakta ve Raspberry Pi üzerinde çalışan derin öğrenme tabanlı bir görüntü işleme modeli ile analiz etmektedir. Geliştirilen sistem, 13 farklı bitki türüne ait 5500'ü aşkın görüntüden oluşan özel bir veri seti kullanılarak eğitilen Convolutional Neural Network (CNN) modeli sayesinde, kameraya gösterilen fidelerin hem türünü hem de yabani olup olmadığını yüksek doğrulukla sınıflandırabilmektedir.

Robotun hareketi ve kamera açısı, ESP32 mikrodenetleyicisi üzerinde çalışan çok işlevli firmware ile kontrol edilen L298N motor sürücü modülü ve pan-tilt mekanizması aracılığıyla sağlanmaktadır. Kullanıcının sisteme erişimi, Flask tabanlı web sunucusu üzerinden sunulan sezgisel bir arayüz ile gerçekleşmektedir. Bu arayüz; canlı kamera görüntüsünü, anlık sınıflandırma sonuçlarını ve ileri-geri, sağ-sol gibi yön komutlarını tek bir ekranda birleştirerek operatöre kesintisiz bir deneyim sunmaktadır. Ayrıca HC-SR04 ultrasonik mesafe sensörü yardımıyla engel algılama özelliği kazandırılan sistem, tarla koşullarında otonom ya da yarı-otonom hareket edebilme potansiyeline sahiptir.

Tezin temel amacı, tarımsal alanlarda yaygın bir problem olan yabani otların erken tespiti ve ürün bitkilerinin doğru sınıflandırılmasını, düşük maliyetli ve taşınabilir bir robot platformu ile sağlamaktır. Böylece, kimyasal herbisit kullanımının azaltılması, iş gücünün daha verimli planlanması ve bitki sağlığının izlenmesi gibi kritik süreçler kolaylaştırılacaktır. Literatürde benzer örnekler bulunsa da bu çalışma, gömülü yapay zekâ yaklaşımını sahada gerçek zamanlı çalıştırması, pan-tilt özellikli görüntüleme birimini IoT kontrollü mobil şase ile birleştirmesi ve tüm bileşenleri açık kaynak yazılım-donanım altyapısı üzerinde entegre etmesi açısından özgünlük taşımaktadır.

2. Literatür Araştırması

2.1 Mobil Robot Sistemleri

Mobil robot sistemleri, fiziksel dünyada hareket edebilen ve çevreleriyle etkileşime girebilen otonom veya yarı otonom cihazlardır. Literatürde, mobil robotların çeşitli görevlerde kullanıldığı ve sürekli olarak geliştirildiği görülmektedir.

- **Uygulama Alanları:** Mobil robotlar, arama ve kurtarma operasyonlarında (afet durumlarında insan kurtarma ekiplerine destek sağlama), lojistik ve depolama, keşif ve gözetleme, askeri uygulamalar, ev hizmetleri ve tarım gibi geniş bir yelpazede kullanılmaktadır. Özellikle tehlikeli veya erişimi zor alanlarda insan müdahalesinin yerini alarak riskleri azaltma potansiyeline sahiptirler.
- **Temel Bileşenler:** Bir mobil robot sistemi genellikle şu bileşenleri içerir:
    - **Algılayıcılar (Sensörler):** Çevreyi algılamak için kızılötesi, ultrasonik, lidar, kamera, GPS gibi sensörler kullanılır. Bu sensörler, robotun konumunu belirlemesine, engelleri tespit etmesine ve çevresel verileri toplamasına yardımcı olur. Örneğin, ultrasonik ve kızılötesi sensörler, robotun çarpışmaları önlemesi için mesafe ölçümleri sağlar.
    - **Eyleyiciler (Aktüatörler):** Motorlar, tekerlekler, kollar gibi robotun hareketini ve fiziksel etkileşimini sağlayan bileşenlerdir. Enkoderli motorlar, robotun konumunu ve hareketini hassas bir şekilde takip etmek için önemlidir.
    - **Kontrol Ünitesi:** Algılayıcılardan gelen verileri işleyen ve eyleyicileri yönlendiren bir mikrodenetleyici veya bilgisayar sistemidir.
    - **İletişim Modülleri:** Robotun diğer sistemlerle (örneğin, merkezi bir bilgisayar veya operatör) kablosuz olarak iletişim kurmasını sağlar.
- **Haritalama ve Konumlandırma (SLAM):** Mobil robotların karmaşık ve bilinmeyen ortamlarda başarılı bir şekilde gezinmesi için eşzamanlı konumlandırma ve haritalama (SLAM) algoritmaları kritik öneme sahiptir. Bu algoritmalar, robotun kendi konumunu belirlerken aynı anda çevresinin bir haritasını oluşturmasına olanak tanır.
- **Gelecek Perspektifi:** Mobil robot teknolojileri, yapay zeka ve makine öğrenimi ile birleşerek daha otonom, uyarlanabilir ve akıllı hale gelmektedir. Bu gelişmeler, mobil robotların daha karmaşık görevleri daha etkin bir şekilde yerine getirmesine olanak sağlayacaktır.

2.2 Raspberry Pi Tabanlı Robotik Uygulamalar

Raspberry Pi, düşük maliyetli, kredi kartı boyutunda bir tek kart bilgisayar olup, robotik ve otomasyon projelerinde oldukça popüler bir platform haline gelmiştir. Geniş GPIO (General Purpose Input/Output) pin setleri, Linux tabanlı işletim sistemi desteği ve güçlü işlem yeteneği sayesinde birçok farklı robotik uygulamanın geliştirilmesine imkan tanır.

- **Avantajları:**
    - **Düşük Maliyet:** Raspberry Pi'nin uygun fiyatı, hobi projelerinden eğitim amaçlı uygulamalara kadar geniş bir kitleye hitap etmesini sağlar.
    - **Çok Yönlülük:** Python, C++, Java gibi çeşitli programlama dillerini destekler. Ayrıca geniş bir sensör ve aktüatör ekosistemine sahiptir.
    - **Topluluk Desteği:** Geniş bir geliştirici ve kullanıcı topluluğu, sorun giderme ve proje geliştirmede önemli bir kaynak sunar.
    - **Yüksek İşlem Gücü:** Küçük boyutuna rağmen, görüntü işleme ve yapay zeka algoritmaları gibi işlem yoğun görevler için yeterli gücü sağlar.
- **Uygulama Örnekleri:**
    - **Akıllı Ev Sistemleri:** Raspberry Pi, sensörler (sıcaklık/nem, hareket, titreşim) ve aktüatörler (servo motorlar, ışıklar) ile entegre edilerek akıllı ev otomasyon sistemleri oluşturmak için kullanılır.
    - **Mobil Robot Kontrolü:** Robotun motorlarını kontrol etmek, sensör verilerini işlemek ve kablosuz iletişim kurmak için merkezi bir işlem birimi olarak işlev görür.
    - **Görüntü İşleme Tabanlı Projeler:** Özellikle kamera modülü ile birlikte kullanılarak nesne tanıma, yüz tanıma ve takip gibi uygulamalarda yer alır.
    - **Eğitim ve Araştırma:** Öğrencilerin ve araştırmacıların robotik ve gömülü sistemler hakkında pratik bilgi edinmeleri için ideal bir platformdur.

2.3 Kamera Tabanlı Görüntü İşleme ve WiFi İletişimi

Kamera tabanlı görüntü işleme ve WiFi iletişimi, mobil robot sistemlerinin çevreleriyle etkileşimini ve uzaktan kontrolünü sağlayan kritik teknolojilerdir.

- **Kamera Tabanlı Görüntü İşleme:**

- **Algılama ve Tanıma:** Kameralar, robotların çevredeki nesneleri, insanları veya belirli özellikleri algılaması ve tanıması için kullanılır. Bu, robotun navigasyonu, nesne manipülasyonu ve karar verme süreçleri için hayati önem taşır.
- **Bilgisayar Görüsü Algoritmaları:** Görüntü işleme genellikle OpenCV gibi kütüphanelerle gerçekleştirilir. Kenar tespiti, nesne takibi, renk ayrımı, desen tanıma ve derin öğrenme tabanlı nesne algılama (YOLO gibi) algoritmaları yaygın olarak kullanılır.
- **Uygulama Alanları:** Tarımda bitki hastalıklarının tespiti, endüstride kalite kontrol, güvenlik sistemlerinde yüz tanıma ve mobil robotların engel kaçınması gibi birçok alanda görüntü işleme teknolojileri temel rol oynar.

- **WiFi İletişimi:**
  - **Uzaktan Kontrol ve Veri Aktarımı:** WiFi, mobil robotların uzaktan kontrol edilmesi ve sensör verilerinin (özellikle kamera akışının) gerçek zamanlı olarak merkezi bir sunucuya veya operatöre iletilmesi için güvenilir bir kablosuz iletişim yöntemi sunar.
  - **Ağ Yapısı:** Robotlar genellikle bir WiFi ağına bağlanarak komutları alabilir ve verileri gönderebilirler. Bu, özellikle karmaşık görevlerin uzaktan denetimini ve veri analizi için büyük veri setlerinin transferini kolaylaştırır.
  - **Avantajları:** Geniş kapsama alanı, yüksek bant genişliği ve yaygın erişilebilirlik, WiFi'yi robotik uygulamalar için cazip bir seçenek haline getirir.

2.4 Yapay Zeka Algoritmaları ile Bitki Analizi

Yapay zeka (YZ) algoritmaları, tarım sektöründe bitki analizi ve yönetimi için devrim niteliğinde çözümler sunmaktadır. Bu teknolojiler, verimliliği artırmak, kaynak kullanımını optimize etmek ve bitki sağlığını iyileştirmek için kullanılır.

- **Bitki Hastalık ve Zararlı Tespiti:**
  - **Derin Öğrenme Modelleri:** Evrişimsel Sinir Ağları (CNN) gibi derin öğrenme modelleri, bitki yapraklarındaki hastalık belirtilerini, lekeleri veya anormallikleri kamera görüntüleri üzerinden yüksek doğrulukla tespit etmek için eğitilir. Bu, çiftçilere erken uyarı sağlayarak hızlı müdahale imkanı sunar.

    - **Mobil Uygulamalar:** Akıllı telefon kameralarıyla çekilen bitki görselleri üzerinden çalışan YZ tabanlı uygulamalar (örneğin Agrio, Plantix), bitki hastalıklarını otomatik olarak tanımlayabilir ve tedavi önerileri sunabilir.
- **Bitki Türü ve Sağlık Analizi:**
    - **Nesne Tespiti Algoritmaları (YOLO):** Yüzlerce veya binlerce bitki görseli ile eğitilen YOLO gibi algoritmalar, belirli bitki türlerini veya sağlıklı/hastalıklı bitkileri anında tespit edebilir. Bu, bitki coğrafyası araştırmalarında veya tarım alanlarında bitki sayımı ve sınıflandırması için kullanılabilir.
    - **Besin Durumu ve Büyüme Takibi:** Yaprak analizleri (örneğin, bitki öz suyu analizi) ile elde edilen veriler, YZ algoritmaları ile işlenerek bitkinin anlık beslenme durumu ve besin ihtiyaçları hakkında detaylı raporlar oluşturulabilir. Bu raporlar, çiftçilerin gübreleme programlarını optimize etmelerine yardımcı olur.
- **Verim Tahmini ve Kaynak Optimizasyonu:**
    - Yapay zeka, çevresel faktörler (sıcaklık, nem, toprak bileşimi) ve bitki sağlığı verileri kullanılarak mahsul verimini tahmin edebilir.
    - Sulama ve gübreleme gibi kaynakların optimize edilmesi için akıllı reçeteler oluşturulabilir, bu da verimliliği artırırken maliyetleri düşürür.
- **Uzaktan Algılama Entegrasyonu:** İnsansız hava araçları (İHA'lar) veya saha robotlarına monte edilmiş sensörlerden (hiperspektral kameralar, termal kameralar) alınan veriler, YZ algoritmaları ile birleştirilerek geniş alanlarda bitki sağlığı izlemesi ve anormallik tespiti yapılabilir.

2.5 Sonuç

Bu literatür araştırması, mobil robot sistemlerinin temel prensiplerinden, Raspberry Pi'nin robotik projelerdeki esnek kullanımına, kamera tabanlı görüntü işleme ve kablosuz iletişimin kritik rolüne ve yapay zeka algoritmalarının bitki analizindeki dönüştürücü etkisine kadar geniş bir perspektif sunmaktadır. Bu alanlardaki gelişmeler, gelecekte akıllı tarım uygulamalarından otonom endüstriyel sistemlere kadar birçok alanda önemli yeniliklerin kapısını aralamaktadır. Özellikle Raspberry Pi gibi düşük maliyetli donanımların YZ ve görüntü işleme yetenekleriyle birleşmesi, bu teknolojilerin daha geniş kitleler tarafından erişilebilir olmasını sağlamaktadır.

3. Sistem Entegrasyonu

Sistemin farklı donanım ve yazılım bileşenleri arasında kurulan entegrasyon sayesinde, cihazlar arası veri alışverişi ve görev dağılımı gerçekleştirilmiştir. Bu kapsamda, ESP32 mikrodenetleyicisi ile araç üzerindeki motorlar, pan-tilt servolar ve mesafe sensörü kontrol edilmiştir. Bu kontrol, Arduino IDE üzerinden yazılan C++ tabanlı bir yazılım ile sağlanmıştır. ESP32 aynı zamanda, cep telefonundan gelen yön komutlarını Wi-Fi ağı üzerinden alarak uygun donanım tepkilerini üretmiştir.

Öte yandan Raspberry Pi, sistemin görüntü işleme merkezini oluşturmuş; Flask tabanlı bir web sunucusu üzerinden canlı kamera görüntüsü yayınlanmış ve TensorFlow/Keras tabanlı derin öğrenme modeli ile sınıflandırma işlemleri yapılmıştır. Raspberry Pi üzerindeki derin öğrenme modeli, kameradan alınan görüntüyü analiz ederek bitki türünü ve yabani olup olmadığını belirlemiş, bu sonuçları web arayüzü aracılığıyla kullanıcıya iletmiştir.

ESP32 ile Raspberry Pi arasında TCP/IP tabanlı haberleşme, ESP32'nin oluşturduğu özel Wi-Fi ağı üzerinden gerçekleştirilmiştir. Flask sunucusu ile ESP32 HTTP istekleri üzerinden senkronize çalışarak, web üzerinden verilen yön komutlarını doğrudan araca aktarmıştır. Bu bütünleşik yapı sayesinde, kullanıcı yalnızca cep telefonundaki tarayıcı üzerinden sistemi yönetebilmiş, böylece kullanıcı deneyimi ve mobilite en üst düzeye çıkarılmıştır.

Tüm bu entegrasyon sayesinde sistem, hem donanım hem de yazılım açısından modüler bir yapıya sahip olmuştur. İlerleyen süreçte farklı donanımlar veya gelişmiş modeller sisteme kolayca entegre edilebilecek şekilde tasarım yapılmıştır. Aynı zamanda bileşenlerin birbirine senkronize çalışmasıyla gecikme süreleri minimize edilmiştir. Kullanıcı deneyimini daha da iyileştirmek için sistemde geri bildirim mekanizmaları düşünülmüş; hareket komutlarının uygulandığına dair görsel veya sesli uyarılar eklenebilir hale getirilmiştir. Böylece, sistem yalnızca akademik bir prototip değil, saha uygulamaları için pratik bir çözüm altyapısına kavuşmuştur.

4. Karşılaşılan Sorunlar ve Çözümler

Projenin geliştirme sürecinde hem yazılımsal hem de donanımsal bazı zorluklarla karşılaşılmıştır. Bu sorunlar, sistemin daha kararlı ve verimli hale gelmesini sağlamak amacıyla çözülerek ilerlenmiştir.

- **ESP32 ve Kamera Haberleşmesi:** Başlangıçta, tüm sistemin ESP32 üzerinden çalıştırılması planlanmıştı ancak USB kamera bağlantısı ESP32 tarafından desteklenmediği ve bant genişliği ihtiyacını karşılayamadığı için, kamera doğrudan Raspberry Pi'ye bağlanarak bu sorun aşıldı.
- **Modelin Hızlı Çalışması:** Raspberry Pi üzerinde çalışan CNN modeli ilk başta yüksek gecikme süresi gösterdi. Bu durumu iyileştirmek için modelin boyutu küçültüldü, görüntü boyutu 64x64 piksel ile sınırlandırıldı ve yazılımda gereksiz işlem döngüleri optimize edildi. Böylece sınıflandırma sonuçları daha hızlı alınabilir hale geldi.
- **Enerji Problemleri:** Kamera, motorlar, pan-tilt servolar ve Raspberry Pi aynı anda çalıştırıldığında sistem kararsız hale geliyordu. Bu sorunu çözmek için güç kaynağı çıkışları dengelendi ve ayrı enerji dağıtımı yapıldı. Raspberry Pi için harici bir batarya paketi kullanılarak sistem daha stabil hale getirildi.
- **Pan-Tilt Mekanizmasında Senkron Hata:** Pan ve tilt servo motorlarının eş zamanlı hareketinde gecikme yaşanıyordu. Kod yapısında senkronizasyon sağlanarak, hareket komutlarının aynı anda gönderilmesi sağlandı ve daha yumuşak yön değişimi elde edildi.
- **Wi-Fi Bağlantı Sorunları:** ESP32 ile web arayüzü arasındaki bağlantı bazı durumlarda kopmaktaydı. Bu durum, ESP32 tarafında bağlantı kopmalarını tespit edip otomatik yeniden bağlanma algoritması ile düzeltildi. Ayrıca yönlendirici yerine ESP32'nin kendi ağı üzerinden doğrudan bağlantı sağlanarak parazit etkisi azaltıldı.
- **Veri Seti Dengesizliği:** Projenin başlangıç aşamasında kullanılan veri seti sınıflar arasında dengesizdi; bazı bitki türlerine ait örnek sayısı çok az, bazılarının ise fazlaydı. Bu durum, modelin bazı sınıflara karşı önyargılı tahminlerde bulunmasına neden oldu. Bu sorunu çözmek amacıyla, veri setindeki bitki türlerinin sayıları denkleştirildi. Böylece modelin doğruluk oranı arttı ve sınıflar arası denge sağlanmış oldu.

5. Alternatif Yöntemler ve Teknolojik Seçimler Üzerine Değerlendirme

Bu projede yapılan donanım ve yazılım tercihleri, düşük maliyetli ve açık kaynak temelli bir sistem geliştirmek amacıyla belirlenmiştir. Ancak farklı ihtiyaçlar, ölçekler ve performans beklentileri doğrultusunda farklı yöntem ve elemanlar da tercih edilebilirdi:

- **Mikrodenetleyici Alternatifi (ESP32 yerine Raspberry Pi Pico W veya STM32):** Daha az güç tüketen, daha deterministik zamanlama sunan mikrodenetleyiciler tercih edilerek enerji verimliliği artırılabilir ve sistemin gerçek zamanlı tepkisi iyileştirilebilirdi. Özellikle yalnızca motor/sensör kontrolü gereken uygulamalarda ESP32'nin yerine daha yalın çözümler tercih edilebilirdi.
- **Motor Sürücü Alternatifi (L298N yerine TB6612FNG veya DRV8833):** L298N gibi eski nesil sürücüler yerine daha kompakt ve verimli motor sürücüleri tercih edilerek hem enerji kayıpları azaltılabilir hem de termal performans artırılabilirdi.
- **Görüntü İşleme Donanımı (Raspberry Pi yerine NVIDIA Jetson Nano veya Google Coral Dev Board):** Modelin daha karmaşık hale gelmesi veya daha yüksek çözünürlüklü görüntülerin işlenmesi gerektiğinde, Raspberry Pi yetersiz kalabilir. Bu durumda donanım tabanlı hızlandırıcıları olan daha güçlü kartlar tercih edilebilirdi.
- **Kablosuz Haberleşme (ESP32'nin Wi-Fi'si yerine LoRa, ZigBee veya LTE):** Özellikle uzak mesafelerde veya açık alanda çalışacak sistemlerde Wi-Fi yeterli çekim alanı sunmadığı durumlarda, LoRa gibi düşük bant genişlikli ama uzun menzilli protokoller tercih edilebilirdi.
- **Web Arayüzü Alternatifi (Flask yerine Node.js veya MQTT Tabanlı Dashboard):** Flask kullanımı basit ve yeterli olsa da daha fazla eş zamanlı kullanıcı desteği veya olay tabanlı mimari istenirse Node.js gibi platformlar tercih edilebilirdi. Ayrıca IoT odaklı arayüzler için MQTT tabanlı paneller (ör. Node-RED) daha hızlı kurulum ve modülerlik sağlayabilirdi.
- **Model Mimarisi Alternatifi (CNN yerine MobilNetV2, EfficientNet):** Özellikle gömülü sistemlerde kullanılmak üzere optimize edilmiş daha hafif ve hızlı çalışan mimariler tercih edilerek hem modelin doğruluğu artırılabilir hem de daha düşük işlem gücüyle çalışması sağlanabilirdi.

6. Sonuç ve Gelecek Çalışmalar

Bu proje sayesinde, bitkileri sahada gerçek zamanlı tanıyabilen ve uzaktan kontrol edilebilen bir mobil tarım robotu başarıyla geliştirildi. Sistem, kamera ile elde edilen görüntüler üzerinden bitki sınıflandırması yapabilmekte, araç hareketlerini web arayüzü üzerinden yönetebilmekte ve sınıflandırma sonuçlarını anlık olarak kullanıcıya sunabilmektedir. Donanım ve yazılım bileşenlerinin bütünleşik yapısı, sistemin modüler ve genişletilebilir olmasına olanak tanımaktadır.

Gelecek çalışmalarda aşağıdaki iyileştirme ve genişletme hedefleri planlanmaktadır:

- **GPS Entegrasyonu ile Haritalama:** Sisteme entegre edilecek bir GPS modülü ile aracın bulunduğu konum bilgisi gerçek zamanlı olarak elde edilecek ve bu bilgilerle sınıflandırılan bitkilerin lokasyonları bir harita üzerine işlenecektir. Bu sayede tarla içinde hangi bölgelerde hangi bitki türlerinin veya yabani otların yoğun olduğu görsel olarak analiz edilebilecektir.
- **Daha Fazla Bitki Türüyle Eğitim:** Mevcut model, 13 farklı bitki türü ile sınırlıdır. Gelecek çalışmalarda veri setine yeni bitki türleri eklenecek ve model yeniden eğitilerek daha geniş kapsamlı sınıflandırmalar yapabilecek hale getirilecektir. Bu, sistemin farklı bölgelerdeki tarım türlerine kolaylıkla uyarlanmasını sağlayacaktır.
- **Otomatik İmha Mekanizması Entegrasyonu:** Yabani otlar tespit edildiğinde manuel müdahale yerine sistem üzerine eklenecek küçük bir mekanik aparat (örneğin mini püskürtme nozulu veya mikro bıçak mekanizması) ile yabani otların otomatik olarak ortadan kaldırılması hedeflenmektedir. Bu, sistemin otonom tarımsal müdahale kabiliyetini artıracaktır.
- **Güneş Paneli ile Enerji Verimliliği Artırımı:** Araç üzerine yerleştirilecek küçük boyutlu güneş panelleri ile sistemin enerji ihtiyacının bir kısmı doğrudan güneşten sağlanacaktır. Bu sayede batarya kullanım süresi uzayacak, dış ortamda daha uzun süreli çalışmalara olanak tanınacaktır.

7. Kullanılan Donanımlar

- **ESP32 Geliştirme Kartı:** ESP32, düşük enerji tüketimi, entegre Wi-Fi ve Bluetooth desteği, çift çekirdekli Tensilica Xtensa işlemcisi ve çok sayıda dijital/analog giriş-çıkış pinine sahip olması sayesinde projede merkezi kontrol birimi olarak tercih edilmiştir. PWM, I2C, SPI ve UART gibi haberleşme protokollerini desteklemesi, ESP32'nin motor sürücüleri, mesafe sensörleri ve pan-tilt kamera mekanizması gibi farklı birimlerle kolaylıkla entegre edilmesine olanak tanımıştır. Araç üzerindeki tüm donanımlar bu kart üzerinden kontrol edilmekte olup, görevler arasında çoklu işlem yönetimi sağlanmaktadır. Ayrıca kablosuz bağlantı üzerinden web tabanlı arayüz ile iletişim kurarak kullanıcının cep telefonu ya da bilgisayar aracılığıyla gönderdiği yön komutlarını gerçek zamanlı olarak alır ve işler. Bu özellikleriyle ESP32, hareketli araçlarda otonom kontrol, uzaktan erişim ve enerji verimliliği gerektiren projeler için ideal bir geliştirme platformudur. Örnek görsel Fig. 7.1'de verilmiştir.

Fig. 7.1: ESP32 Geliştirme Kartı

- **L298N Motor Sürücü Modülü:** İki adet DC motorun yön (ileri-geri) ve hız kontrolünü sağlamak amacıyla kullanılan çift H-köprüsü entegresine sahip bir motor kontrol kartıdır. Bu modül, mikrodenetleyicilerden gelen düşük güçlü kontrol sinyallerini motorlar için gerekli olan daha yüksek akım ve voltaja çevirerek motorların verimli şekilde çalışmasını sağlar. Projede ESP32 geliştirme kartı üzerinden gönderilen PWM ve dijital sinyaller ile L298N modülü sürülmekte; böylece motorların hem yönü hem de hızı hassas bir şekilde kontrol edilmektedir. Ayrıca modül üzerinde bulunan harici güç girişi sayesinde motorlar, ESP32'nin sağlayamayacağı yüksek akım ihtiyacını karşılayacak şekilde harici bir kaynaktan beslenebilir. Üzerinde yer alan soğutucu,

ısınmaya karşı koruma sağlayarak uzun süreli çalışmalarda sistemin kararlılığını artırır. Örnek görsel Fig. 7.2'de verilmiştir.

Fig. 7.2: L298N Motor Sürücü Modülü

- **Pan-Tilt Mekanizması:** Bir kameranın hem yatay (pan) hem de dikey (tilt) eksende yönlendirilmesini sağlayan iki serbestlik derecesine sahip, servo motorlarla çalışan bir hareketli platform sistemidir. Projede kullanılan bu mekanizma sayesinde, kamera sabit bir konumda durmaksızın farklı yönlere çevrilebilmekte ve böylece daha geniş bir görüş alanı elde edilmektedir. Yapısında yer alan iki adet servo motor, ESP32 geliştirme kartı üzerinden gönderilen PWM sinyalleri ile kontrol edilmekte; bu sayede kullanıcı web arayüzü üzerinden kameranın konumunu gerçek zamanlı olarak değiştirebilmektedir. Pan-tilt sistemi, özellikle hareketli araçlarda çevresel izleme, nesne takibi veya otonom karar verme gibi işlevlerde kameranın yönlendirilmesini mümkün kılar. Esnek yapısı sayesinde kamera görüş açısının ayarlanabilir olmasını sağlayarak sistemin adaptif ve etkileşimli çalışmasına olanak tanır. Örnek görsel Fig. 7.3'te verilmiştir.

Fig. 7.3: Pan-Tilt Mekanizması

- **HC-SR04 Mesafe Sensörü:** Ultrasonik dalgalar kullanarak cisimlerin uzaklığını ölçen ve genellikle engel algılama, robotik navigasyon ve çarpışma önleme sistemlerinde tercih edilen bir sensördür. Çalışma prensibi, belirli frekansta gönderilen ses dalgalarının bir nesneye çarpıp geri dönme süresinin ölçülmesine dayanır; bu süre üzerinden mesafe hesaplanır. Projede bu sensör, aracın önüne yerleştirilerek çevredeki engellerin tespiti amacıyla kullanılmıştır. ESP32 mikrodenetleyicisine bağlı olarak çalışan sensör, trig pininden gönderilen sinyalin yankısını echo pini aracılığıyla geri alır ve bu veriyi işleyerek milimetre hassasiyetinde mesafe ölçümü yapılmasını sağlar. Ölçülen bu mesafe verileri, aracın çarpışma riskini algılaması ve gerektiğinde hareket komutlarını durdurması veya yönünü değiştirmesi için kullanılır. Hafif, düşük maliyetli ve kolay uygulanabilir yapısı sayesinde HC-SR04 sensörü, projede güvenilir bir engel algılama çözümü olarak önemli bir rol üstlenmiştir. Örnek görsel Fig. 7.4'te verilmiştir.

Fig. 7.4: HC-SR04 Mesafe Sensörü

- **USB Web Kamerası:** Raspberry Pi'ye USB arabirimi üzerinden bağlanarak çevreden gelen görüntüleri yüksek çözünürlükte yakalayan ve bu verileri gerçek zamanlı olarak görüntü işleme algoritmalarına aktaran bir görüntü toplama birimidir. Projede, bitki tespiti ve sınıflandırması gibi yapay zekâ tabanlı uygulamalar için temel veri kaynağı olarak kullanılmıştır. Kamera, Python tabanlı görüntü işleme kütüphaneleri (OpenCV gibi) ile entegre çalışarak, her kareyi anlık olarak işleme sistemine aktarır ve bu sayede bitkilerin şekil, renk ve desen gibi özellikleri analiz edilebilir. Elde edilen görseller, Raspberry Pi üzerinde çalışan sınıflandırma modeli tarafından yorumlanarak bitkinin türü ve yabani olup olmadığı belirlenir. Gerçek zamanlı çalışması sayesinde, mobil araç üzerinde dinamik ortamlarda bile kesintisiz veri akışı sağlayarak sistemin otonom şekilde karar vermesine katkı sunar. Kolay entegrasyonu ve uygun maliyeti sayesinde,

bu tür yapay zekâ destekli tarım projelerinde ideal bir görüntü kaynağıdır. Örnek görsel Fig. 7.5'te verilmiştir.

Fig. 7.5: USB Web Kamerası

- **Raspberry Pi 5 - 4GB:** kompakt yapısına rağmen yüksek işlem gücüne ve gelişmiş donanım özelliklerine sahip bir minibilgisayardır. Projede, Python ortamında geliştirilen yapay zekâ modelini çalıştırmak, kamera üzerinden alınan görüntüleri işlemek ve gerçek zamanlı sınıflandırma yapmak amacıyla kullanılmıştır. USB web kamerasından gelen görüntüler bu cihazda işlenerek CNN modeline iletilir; modelden elde edilen sonuçlar ise kullanıcıya sunulmak üzere web arayüzüne aktarılır. Hem görüntü işleme hem de model çalıştırma görevlerini akıcı bir şekilde yerine getirebildiği için, Raspberry Pi 5 bu tür gömülü yapay zekâ uygulamaları için ideal bir platformdur. Örnek görsel Fig. 7.6'da verilmiştir.

Fig. 7.6: Raspberry Pi 5 - 4GB

- **4WD Taşınabilir Şase ve Batarya**: 4WD taşınabilir şase, projede kullanılan tüm elektronik bileşenlerin (ESP32, motor sürücü, sensörler, kamera vb.) monte edildiği, dört adet DC motorla donatılmış hareketli bir platformdur. Hafif ancak dayanıklı yapısı sayesinde engebeli arazilerde dahi stabil bir sürüş sunar ve dış ortam koşullarına uyum sağlar. Dört tekerlekten çekişli (4WD) yapısı, aracın daha güçlü ve dengeli hareket etmesini mümkün kılar. Sistemin enerji ihtiyacı ise üzerinde bulunan yüksek kapasiteli lityum iyon batarya ile karşılanmakta; bu da kablosuz ve uzun süreli saha uygulamaları için gerekli olan mobiliteyi sağlar. Batarya, ESP32 ve motorlar dahil tüm bileşenlere gerekli gerilim ve akımı sağlayacak şekilde seçilmiş olup, aracın otonom şekilde görev yapmasına olanak tanır. Bu yapı sayesinde sistem, tarım gibi açık alan uygulamalarında bağımsız ve verimli bir şekilde çalışabilir. Örnek görseller Fig. 7.7 ve Fig. 7.8'de verilmiştir.

Fig. 7.7: 4WD Taşınabilir Şase

Fig. 7.8: Batarya

8.Yazılım Mimarisi

Bu projede yazılım mimarisi, donanım kontrolü ve görüntü işleme süreçlerinin entegre çalışmasını sağlamak amacıyla tasarlanmıştır.

8.1 Arduino IDE ve ESP32 & Flask Tabanlı Web Arayüzü: Araç üzerindeki motorlar, pan-tilt servoları ve HC-SR04 mesafe sensörü ESP32 mikrodenetleyicisi tarafından yönetilmektedir. Bu işlemci, Arduino IDE ortamında geliştirilen C++ dilindeki program aracılığıyla hem donanım kontrolünü hem de web tabanlı iletişimi sağlar. Sisteme entegre edilmiş Flask tabanlı web arayüzü ile kullanıcı, yön komutlarını (ileri, geri, sağa, sola, dur)

belirli butonlara tıklayarak gönderir. Bu komutlar HTTP üzerinden ESP32'ye ulaşır ve burada işlenerek uygun PWM sinyalleri ile motorlara iletilir. Pan-tilt mekanizması da benzer şekilde servo açıları üzerinden kontroledilirken, HC-SR04 sensörü yardımıyla çevredeki engeller tespit edilir ve bu bilgi de web arayüzüne anlık olarak aktarılır. Arayüz ayrıca kamera görüntüsünü canlı olarak sunar ve sınıflandırma sonucunu ekranda görüntüler. HTML ve CSS ile zenginleştirilen bu kullanıcı arayüzü, mobil cihazlarla da uyumlu olacak şekilde optimize edilmiştir.

```
#include <WiFi.h>
#include <WebServer.h>
#include <ESP32Servo.h>

const char* ssid = "ESP32-CAR";
const char* password = "12345678";

#define ENA 5
#define ENB 23
#define IN1 22
#define IN2 21
#define IN3 19
#define IN4 18

#define PAN_PIN 14
#define TILT_PIN 27
Servo servoPan;
Servo servoTilt;

#define TRIG_PIN 13
#define ECHO_PIN 12

int speedVal = 150;
WebServer server(80);

#define PWM_A 0
#define PWM_B 1

long distanceCM = 0;
```

Fig. 8.1.1: Donanım kontrolü + Web arayüzü kodu (1. kısım)

Fig. 8.1.1'deki kod bloğunda, öncelikle ESP32'nin kullanacağı temel kütüphaneler eklenmiştir. WiFi.h kablosuz bağlantı oluşturmak için, WebServer.h HTTP sunucusu kurmak için ve ESP32Servo.h servo motorları kontrol etmek için gereklidir. Ardından, motor sürücüsü (ENA, ENB, IN1–IN4), pan-tilt servo motor pinleri (PAN_PIN, TILT_PIN), ultrasonik sensör pinleri (TRIG ve ECHO), PWM kanalları (PWM_A, PWM_B) ve hız gibi önemli sabitler tanımlanmıştır. WebServer server(80) ile 80 numaralı portta çalışan bir web sunucu oluşturulmuştur. Bu bölümdeki tüm tanımlar, sistemin donanım bileşenlerini kontrol edebilmesi için gereklidir ve projenin temel altyapısını oluşturur.

```
void setup() {
  Serial.begin(115200);

  WiFi.softAP(ssid, password);
  Serial.println("🚀 ESP32 Access Point oluşturuldu.");
  Serial.print("📶 Ağ Adı: "); Serial.println(ssid);
  Serial.print("🔑 Şifre: "); Serial.println(password);
  Serial.print("🌐 IP Adresi: "); Serial.println(WiFi.softAPIP());

  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);

  ledcSetup(PWM_A, 1000, 8);
  ledcAttachPin(ENA, PWM_A);
  ledcSetup(PWM_B, 1000, 8);
  ledcAttachPin(ENB, PWM_B);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  servoPan.setPeriodHertz(50);
  servoTilt.setPeriodHertz(50);
  servoPan.attach(PAN_PIN, 500, 2400);
  servoTilt.attach(TILT_PIN, 500, 2400);
  servoPan.write(90);
  servoTilt.write(90);

  stopMotors();

  server.on("/", handleRoot);
  server.on("/move", HTTP_POST, handleMove);
  server.on("/speed", HTTP_POST, handleSpeed);
  server.on("/distance", handleDistance);
  server.begin();
  Serial.println("🌐 Web sunucusu başlatıldı.");
}
```

Fig. 8.1.2: Donanım kontrolü + Web arayüzü kodu (2. kısım)

Fig. 8.1.2'deki kod bloğunda, sistemin ilk başlatıldığı anda çalışan setup() fonksiyonudur. ESP32, WiFi.softAP() komutu ile kendi WiFi ağını oluşturur. Seri port üzerinden ağ adı, şifre ve IP adresi gibi bilgiler kullanıcıya gösterilir. Sonrasında motor pinleri çıkış olarak ayarlanır ve PWM (Pulse Width Modulation) sinyali üretmek için gerekli ledcSetup ve ledcAttachPin işlemleri yapılır. Ultrasonik sensörün giriş-çıkış pinleri belirlenir. Pan-Tilt için kullanılan servo motorlar yapılandırılıp başlangıç pozisyonuna getirilir. Tüm motorlar durdurulur ve web sunucusu başlatılarak gelen HTTP isteklerine hangi fonksiyonların yanıt vereceği tanımlanır. Bu parça, donanımı hazırlar ve ESP32'nin bir web sunucu gibi çalışmasını sağlar.

```
void handleRoot() {
  String html = R"rawliteral(
<!DOCTYPE html><html><head>
  <meta charset="UTF-8"><title>ESP32 Araç Kontrol</title>
  <style>
    body { font-family: Arial; text-align: center; background: #eaf6ff; padding-top: 30px; }
    h1 { color: #0077cc; }
    .btn {
      padding: 15px 30px;
      margin: 10px;
      font-size: 18px;
      background: #0077cc;
      color: white;
      border: none;
      border-radius: 8px;
      cursor: pointer;
    }
    .btn:hover { background: #005fa3; }
    input[type=range] { width: 300px; }
  </style>
</head>
<body>
  <h1>ESP32 Motor Kontrol</h1>
  <div>
    <button class="btn" onclick="sendMove('forward')">sag</button><br>
    <button class="btn" onclick="sendMove('left')">ileri</button>
    <button class="btn" onclick="sendMove('stop')">Dur</button>
    <button class="btn" onclick="sendMove('right')">geri</button><br>
    <button class="btn" onclick="sendMove('backward')">sol</button><br><br>
    <button class="btn" onclick="sendMove('pantilt')">Pan-Tilt Hareket</button>
  </div>
  <br>
  <label for="speed">Hız: </label>
  <input type="range" id="speed" min="0" max="255" value="150" oninput="updateSpeed(this.value)">
  <span id="speedVal">150</span>
  <h2 id="distance">Mesafe: -- cm</h2>
  <script>
    function sendMove(dir) {
      fetch('/move?dir=' + dir, { method: 'POST' });
    }
    function updateSpeed(val) {
      document.getElementById("speedVal").innerText = val;
      fetch('/speed?val=' + val, { method: 'POST' });
    }
    function updateDistance() {
      fetch('/distance')
        .then(res => res.text())
        .then(data => {
          document.getElementById("distance").innerText = "Mesafe: " + data + " cm";
        });
    }
    setInterval(updateDistance, 1000);
  </script>
</body>
</html>
)rawliteral";
  server.send(200, "text/html", html);
}
```

Fig. 8.1.3: Donanım kontrolü + Web arayüzü kodu (3. kısım)

Fig. 8.1.3'teki kod bloğunda, iki önemli fonksiyon vardır. loop() fonksiyonu, sürekli olarak web istemcilerinden gelen bağlantıları kontrol eder (server.handleClient()) ve her döngüde bir kez mesafe ölçümü yapar. handleRoot() fonksiyonu ise ESP32'nin IP adresine gidildiğinde kullanıcıya sunulan web sayfasını içerir. Sayfada butonlar aracılığıyla ileri, geri, dur, sağ, sol ve pan-tilt gibi hareketler kontrol edilir. JavaScript komutları sayesinde /move, /speed, ve /distance gibi sunucu adreslerine veri gönderilir. Böylece kullanıcı sadece bir tarayıcı üzerinden aracı hem yönlendirebilir hem de hız ve mesafe gibi parametreleri izleyebilir.

```
void handleMove() {
  if (!server.hasArg("dir")) {
    server.send(400, "text/plain", "Komut eksik");
    return;
  }
  String dir = server.arg("dir");

  if (dir == "forward") forward();
  else if (dir == "backward") backward();
  else if (dir == "left") turnLeft();
  else if (dir == "right") turnRight();
  else if (dir == "pantilt") {
    servoPan.write(120);
    servoTilt.write(60);
    delay(1000);
    servoPan.write(60);
    servoTilt.write(120);
    delay(1000);
    servoPan.write(90);
    servoTilt.write(90);
  }
  else stopMotors();

  server.send(200, "text/plain", "OK");
}

void handleSpeed() {
  if (server.hasArg("val")) {
    speedVal = constrain(server.arg("val").toInt(), 0, 255);
    ledcWrite(PWM_A, speedVal);
    ledcWrite(PWM_B, speedVal);
    Serial.println("Yeni hız: " + String(speedVal));
  }
  server.send(200, "text/plain", "OK");
}
```

Fig. 8.1.4: Donanım kontrolü + Web arayüzü kodu (4. kısım)

Fig. 8.1.4'teki kod bloğunda, sunucudan gelen komutlara nasıl tepki verileceği tanımlanmıştır. handleMove() fonksiyonu, kullanıcıdan gelen dir parametresine göre uygun hareket fonksiyonunu çağırır. Eğer pantilt komutu gelirse, servo motorlar belirli açılar arasında gidip gelerek baş hareketi simüle eder. handleSpeed() fonksiyonu, kullanıcıdan gelen hız bilgisini

alır, sınırlar içinde tutar ve PWM çıkışlarına uygular. handleDistance() fonksiyonu ise, güncel mesafe değerini HTTP yanıtı olarak döner. Bu yapı sayesinde ESP32 gelen HTTP isteklerine anlamlı ve anlık yanıtlar verir.

```
long readDistanceCM() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH, 30000);
  if (duration == 0) return -1;
  return duration * 0.034 / 2;
}

void forward() {
  digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
  ledcWrite(PWM_A, speedVal);
  ledcWrite(PWM_B, speedVal);
}
void backward() {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
  ledcWrite(PWM_A, speedVal);
  ledcWrite(PWM_B, speedVal);
}
void turnLeft() {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
  ledcWrite(PWM_A, speedVal);
  ledcWrite(PWM_B, speedVal);
}
void turnRight() {
  digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
  ledcWrite(PWM_A, speedVal);
  ledcWrite(PWM_B, speedVal);
}
void stopMotors() {
  digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
  ledcWrite(PWM_A, 0);
  ledcWrite(PWM_B, 0);
}
```

Fig. 8.1.5: Donanım kontrolü + Web arayüzü kodu (5. kısım)

Fig. 8.1.5'teki kod bloğunda, temel sürüş komutları ve mesafe ölçüm işlemleri yer alır. readDistanceCM() fonksiyonu, ultrasonik sensörün çalışma prensibine göre yankı süresini ölçer ve bu değeri cm cinsine çevirir. Diğer fonksiyonlar, aracın hareket yönünü kontrol eder: ileri,

geri, sağa ve sola dönüş işlemleri belirli GPIO pinlerine HIGH/LOW sinyali göndererek gerçekleştirilir. PWM sinyali ile motorların hızı speedVal değeri kadar olur. stopMotors() fonksiyonu tüm motorları durdurur. Bu fonksiyonlar doğrudan donanım kontrolü sağlar ve araca fiziksel hareket kazandırır.

8.2 Python (Raspberry Pi): Raspberry Pi üzerinde çalışan Python betikleri, kamera görüntüsünü sürekli olarak alır ve daha önceden eğitilmiş olan CNN modelini kullanarak bitki sınıflandırması yapar. Görüntüden alınan her bir kare TensorFlow/Keras ile yüklenen modele giriş olarak verilir. Modelin çıktısı olan bitki ismi ve yabani olup olmadığı bilgisi daha sonra Flask sunucusu üzerinden istemciye aktarılır.

```
import tensorflow as tf
import numpy as np
import cv2


model_path = "classifier7.h5"
weights_path = "classifier7.weights.h5"

try:
    model = tf.keras.models.load_model(model_path)
    model.load_weights(weights_path)
    print("✅ Model and weights loaded successfully!")
except Exception as e:
    print(f"❌ Error loading model: {e}")
    exit()


categ = ['Black-grass (wild)', 'Charlock (wild)', 'Cleavers (wild)',
         'Common Chickweed (wild)', 'Common wheat (non-wild)',
         'Fat Hen (wild)', 'Loose Silky-bent (wild)', 'Maize (non-wild)',
        'Scentless Mayweed (wild)', 'Shepherds Purse (wild)',
         'Small-flowered Cranesbill (wild)', 'Sugar beet (non-wild)']
```

Fig. 8.2.1: Model-Raspberry entegrasyon kodu (1. kısım)

Fig. 8.2.1'de TensorFlow ile daha önce eğitilmiş olan bir sınıflandırma modelini (classifier7.h5) ve onun ağırlıkları (classifier7.weights.h5) yüklenir. Yükleme işlemi başarılı olursa ekrana başarı mesajı yazdırılır; bir hata oluşursa hata mesajı gösterilir ve program sonlandırılır. Ayrıca, sınıflandırma modelinin tahmin edeceği bitki türleri categ isimli listeyle tanımlanmıştır. Bu liste hem yabani hem de kültür bitkilerini içerir ve sınıflandırma sonucunda hangi bitki türünün tahmin edildiğinin anlaşılmasını sağlar.

```python
def preprocess_image_from_frame(frame):
    try:
        print("📷 Preprocessing captured image...")


        resize_img = cv2.resize(frame, (64, 64))


        Gblur_img = cv2.GaussianBlur(resize_img, (3, 3), 0)


        hsv_img = cv2.cvtColor(Gblur_img, cv2.COLOR_BGR2HSV)


        lower_green = (25, 40, 50)
        upper_green = (75, 255, 255)
        mask = cv2.inRange(hsv_img, lower_green, upper_green)


        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)


        bMask = mask > 0
        clearImg = np.zeros_like(resize_img, np.uint8)
        clearImg[bMask] = resize_img[bMask]


        preprocessed_img = clearImg / 255.0


        img = np.expand_dims(preprocessed_img, axis=0)

        print("✅ Preprocessing complete.")
        return img
    except Exception as e:
        print(f"❌ Preprocessing error: {e}")
        return None
```

Fig. 8.2.2: Model-Raspberry entegrasyon kodu (2. kısım)

Fig. 8.2.2'de verilen bir görüntü karesini model için uygun formata dönüştürmek amacıyla ön işlem adımları uygulanır. Öncelikle görüntü 64x64 piksele yeniden boyutlandırılır ve ardından Gaussian bulanıklaştırma uygulanarak gürültü azaltılır. Görüntü HSV renk uzayına çevrilir ve yeşil renk aralığına karşılık gelen pikseller bir maske ile tespit edilir. Bu maske üzerinde morfolojik işlemler gerçekleştirilerek gürültü ve boşluklar temizlenir. Elde edilen maskeye göre sadece ilgili bölgeler seçilerek temiz bir görüntü oluşturulur ve bu görüntü 0-1 aralığında normalize edilir. Son olarak modelin beklediği giriş şekline getirmek için boyut genişletilir ve işlenmiş görüntü döndürülür. Eğer bir hata oluşursa fonksiyon hata mesajı basar ve None döner.

```python
def predict_and_show(frame):
    print("🧠 Captured image. Running prediction...")
    img = preprocess_image_from_frame(frame)
    if img is None:
        return

    print("🔄 Running model.predict...")
    prediction = model.predict(img, verbose=0)[0]
    predicted_class = np.argmax(prediction)
    confidence = prediction[predicted_class] * 100

    print(f"✅ Prediction complete!")
    print(f"📌 Predicted category: {categ[predicted_class]}")
    print(f"📊 Confidence: {confidence:.2f}%")

    print("\n📈 Prediction Scores:")
    for i, prob in enumerate(prediction):
        print(f"   {categ[i]:<30}: {prob * 100:.2f}%")


    display_img = cv2.resize(frame, (256, 256))
    text = f"{categ[predicted_class]} ({confidence:.2f}%)"
    cv2.putText(display_img, text, (5, 20), cv2.FONT_HERSHEY_SIMPLEX,
                0.45, (0, 255, 0), 1, cv2.LINE_AA)
    cv2.imshow("Prediction", display_img)
```

Fig. 8.2.3: Model-Raspberry entegrasyon kodu (3. kısım)

Fig. 8.2.3'te verilen bir görüntü karesi üzerinde sınıflandırma tahmini yapılır ve sonucu görsel olarak gösterilir. Öncelikle görüntü ön işleme fonksiyonundan geçirilir; ön işleme başarısızsa işlem sonlanır. Ardından model kullanılarak tahmin yapılır ve en yüksek olasılığa sahip sınıf belirlenir. Tahmin edilen sınıf ve buna ait güven skoru yüzdelik olarak ekrana yazdırılır. Ayrıca tüm sınıflara ait olasılık skorları listelenir. Son aşamada, görüntü yeniden boyutlandırılarak üzerine tahmin edilen sınıf ve güven skoru yazılır ve sonuç görseli ekranda gösterilir.

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

print("🎥 Camera started. Press 'c' to capture and classify. Press 'q' to quit.")

while True:
    ret, frame = cap.read()
    if not ret:
        print("❌ Failed to capture from camera.")
        break


    small_frame = cv2.resize(frame, (320, 240))
    cv2.imshow("Live Feed", small_frame)

    key = cv2.waitKey(1) & 0xFF

    if key == ord('c'):
        predict_and_show(frame)
    elif key == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Fig. 8.2.4: Model-Raspberry entegrasyon kodu (4. kısım)

Fig. 8.2.4'te bilgisayar kamerası başlatılır ve canlı görüntü ekrana yansıtılır. Kullanıcı 'c' tuşuna bastığında mevcut kareyi sınıflandırmak için predict_and_show fonksiyonunu çağırır, 'q' tuşuna basıldığında ise kamera kapanır ve tüm pencereler kapatılır.

9. Görüntü İşleme ve Yapay Zekâ Modeli

9.1 Veri Seti: Görüntü işleme algoritmasının eğitimi için kullanılan veri seti, 13 farklı bitki türüne ait toplamda 5500'den fazla bitki fidesi fotoğrafı içermektedir. Bu veri seti, açık kaynaklı bir platform olan Kaggle üzerinden temin edilmiştir. Görseller, her bir sınıf için dengeli bir şekilde dağıtılmıştır. Görüntüler renkli (RGB) formatta ve genellikle 64x64 piksel boyutuna yeniden ölçeklendirilerek modele giriş olarak kullanılmıştır.

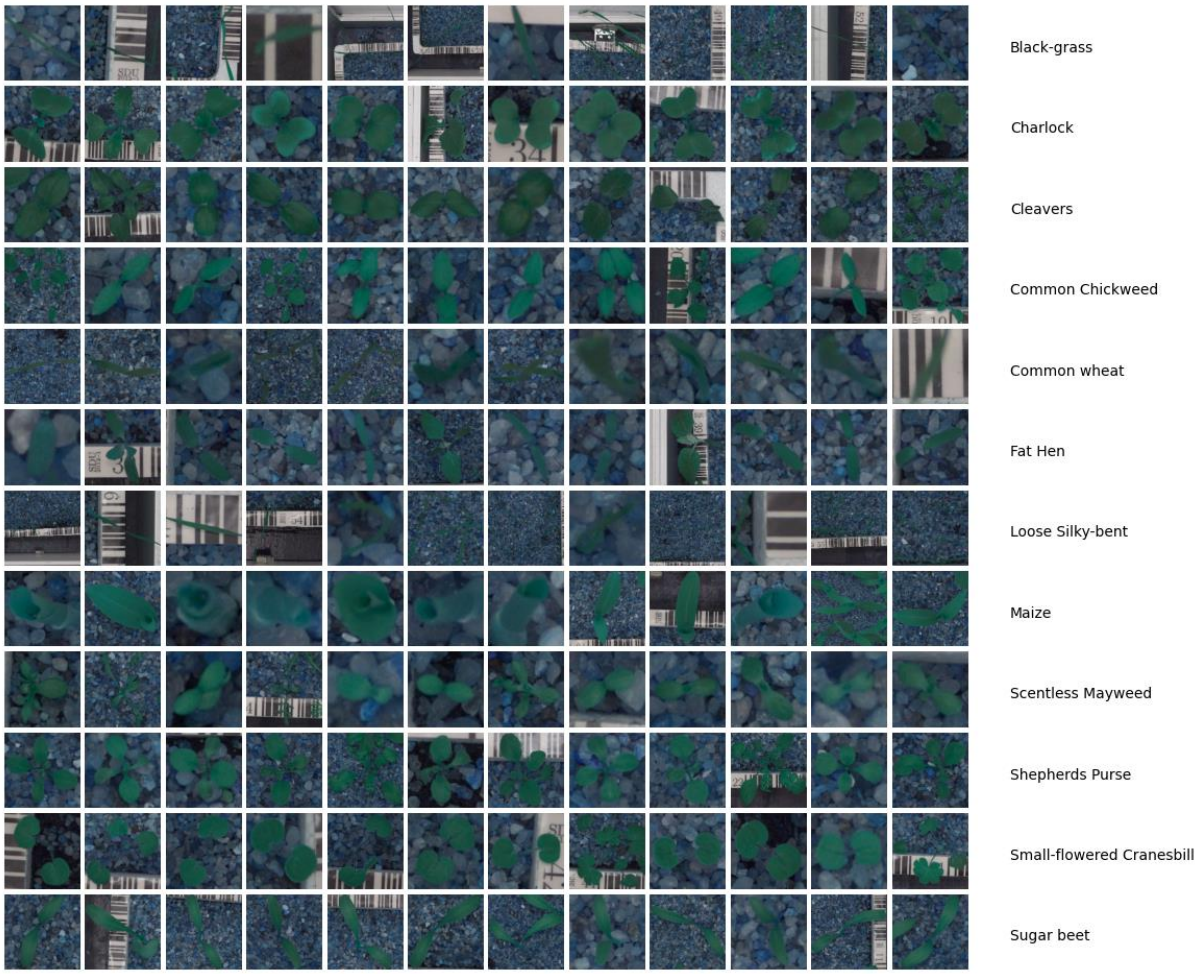

Fig. 9.1.1: Her bir bitki fidesi türüne ait 10 adet resimden oluşan sample'lar

9.2 Ön İşleme (Preprocessing) Aşaması: Öncelikle, belirli bir resim veri kümesinden seçilir ve 64x64 boyutuna yeniden boyutlandırılır. Daha sonra bu resme Gaussian bulanıklaştırma (3x3 kernel) uygulanarak görüntüdeki gürültü azaltılır. Görüntü HSV renk uzayına çevrilerek, yeşil renk aralığına göre maskeleme işlemi yapılır. Oluşan maske ile yalnızca yeşil bölgeler görüntüde korunur ve geri kalanı sıfırlanır.

```
original_img = data[1000]

resized_img = cv2.resize(original_img, (64, 64))

blurred_img = cv2.GaussianBlur(resized_img, (3, 3), 0)

hsv_img = cv2.cvtColor(blurred_img, cv2.COLOR_BGR2HSV)

lower_green = (25, 40, 50)
upper_green = (75, 255, 255)
mask = cv2.inRange(hsv_img, lower_green, upper_green)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

b_mask = mask > 0
clear_img = np.zeros_like(blurred_img, np.uint8)
clear_img[b_mask] = blurred_img[b_mask]

lower_green = (25, 40, 50)
upper_green = (75, 255, 255)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
preprocessed_data_color = []

for img in data:
  resize_img = cv2.resize(img,None,fx=0.50,fy=0.50)
  Gblur_img = cv2.GaussianBlur(resize_img,(3,3),0)
  hsv_img = cv2.cvtColor(Gblur_img, cv2.COLOR_BGR2HSV)
  mask = cv2.inRange(hsv_img, lower_green, upper_green)
  mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
  bMask = mask > 0
  clearImg = np.zeros_like(resize_img, np.uint8)
  clearImg[bMask] = resize_img[bMask]


  preprocessed_data_color.append(clearImg)


preprocessed_data_color = np.asarray(preprocessed_data_color)
```

Fig. 9.2.1: Model implementasyonu (1. kısım)

Bu adımlar daha sonra tüm veri kümesine uygulanarak ön işlenmiş renkli görüntüler preprocessed_data_color listesine kaydedilir. Her bir kategoriden (bitki sınıfından) örnekler ImageGrid yardımıyla görselleştirilir ve böylece sınıflar arası karşılaştırma kolaylaşır. İmplementasyon Fig. 9.2.1'de gösterilmiştir.

9.3 Model Mimarisi: Derin öğrenme modeli, TensorFlow ve Keras kütüphaneleri kullanılarak oluşturulmuş bir Convolutional Neural Network (CNN) mimarisi temelinde geliştirilmiştir. Model mimarisi; girdi katmanı sonrası 3 tane konvolüsyon-kernel katmanı, max-pooling katmanları, dropout (aşırı öğrenmeyi engellemek için), flatten katmanı ve tam bağlantılı (dense) katmanlardan oluşmaktadır. Son katmanda softmax aktivasyon fonksiyonu kullanılarak her bir sınıf için olasılık tahmini yapılmaktadır.

9.4 Model Eğitimi: Veri seti %80 eğitim ve %20 test olmak üzere ikiye ayrılmıştır. Eğitim sürecinde veri artırma (data augmentation) teknikleri kullanılarak modelin genelleme yeteneği artırılmıştır. Model, 25 epoch boyunca Adam optimizasyon algoritması ve categorical cross-entropy kayıp fonksiyonu kullanılarak eğitilmiştir. Eğitim sırasında doğruluk (accuracy), kayıp (loss) ve doğrulama metrikleri sürekli izlenmiştir.

```
from tensorflow.keras.utils import to_categorical

ylabels = to_categorical(ylabels, num_classes=12)

print("Shape of y_train:", ylabels.shape)
print("One value of y_train:", ylabels[0])

from sklearn.model_selection import train_test_split

val_split = 0.25

X_train, X_test1, y_train, y_test1 = train_test_split(preprocessed_data_color, ylabels,
 test_size=0.30, stratify=ylabels,random_state = random_state)

X_train_color, X_test1_color, y_train_color, y_test1_color = train_test_split(data, ylabels,
 test_size=0.30, stratify=ylabels,random_state = random_state)

X_val, X_test, y_val, y_test = train_test_split(X_test1, y_test1, test_size=0.50,
stratify=y_test1,random_state = random_state)

X_val_color, X_test_color, y_val_color, y_test_color = train_test_split
(X_test1_color, y_test1, test_size=0.50, stratify=y_test1,random_state = random_state)

X = np.concatenate((X_train, X_test1))
y = np.concatenate((y_train, y_test1))

print("X_train shape: ", X_train.shape)
print("y_train shape: ", y_train.shape)
print("X_val shape: ", X_val.shape)
print("y_val shape: ", y_val.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y_test.shape)
print("X shape: ", X.shape)
print("y shape: ", y.shape)

X_train = X_train.reshape(X_train.shape[0], 64, 64, 3)
X_val = X_val.reshape(X_val.shape[0], 64, 64, 3)
X_test = X_test.reshape(X_test.shape[0], 64, 64, 3)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_val = X_val.astype('float32')
```

Fig. 9.4.1: Model implementasyonu (2. kısım)

Fig. 9.4.1'deki kod bloğu bir görüntü sınıflandırma modeli için veri önişleme adımlarını ve veri kümesinin eğitim, doğrulama ve test setlerine ayrılmasını gösterir. tensorflow.keras.utils modülünden to_categorical kullanılarak etiketler one-hot kodlamaya dönüştürülür. Ardından, sklearn.model_selection modülünden train_test_split fonksiyonu ile veriler bölünür. val_split 0.25 olarak ayarlanmıştır. Veri setleri (hem ön işlenmiş renkli veriler hem de orijinal renkli veriler) eğitim, test ve doğrulama setlerine ayrılırken stratify parametresi kullanılarak etiket dağılımlarının korunması sağlanır. Son olarak, X_train, X_val ve X_test boyutları yeniden şekillendirilerek (64, 64, 3) formatına getirilir ve veri tipleri float32 olarak ayarlanır.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(shear_range = 0.2,rotation_range=180,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        vertical_flip=True
    )


training_set = train_datagen.flow(X_train,y_train,batch_size=32,seed=random_state,shuffle=True)

import tensorflow as tf
from keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import callbacks
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import optimizers

classifier = Sequential()


classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu', padding = 'same'))

classifier.add(layers.BatchNormalization())

classifier.add(MaxPooling2D(pool_size = (2, 2),strides=2))



classifier.add(Conv2D(64, (3, 3), activation = 'relu', padding = 'same'))
classifier.add(layers.BatchNormalization())
classifier.add(MaxPooling2D(pool_size = (2, 2),strides=2))
```

Fig. 9.4.2: Model implementasyonu (3. kısım)

Fig. 9.4.2'deki kod bloğu görüntü artırma için bir ImageDataGenerator tanımlar ve bir Evrişimsel Sinir Ağı (CNN) modelini oluşturur. ImageDataGenerator shear_range,

rotation_range, zoom_range, width_shift_range, height_shift_range, horizontal_flip ve vertical_flip gibi parametrelerle veri artırma tekniklerini yapılandırır. training_set, bu veri üretecini kullanarak eğitim verilerinden oluşturulur. Model, tf.keras.models.Sequential kullanılarak tanımlanır ve başlangıçta iki adet Conv2D, BatchNormalization ve MaxPooling2D katmanından oluşur. İlk Conv2D katmanı (3,3) filtre boyutu ve relu aktivasyonu ile 32 filtreye sahiptir ve input_shape (64, 64, 3) olarak belirtilmiştir. padding='same' kullanılarak çıktının aynı boyutta kalması sağlanır.

```
classifier.add(Conv2D(64, (3, 3), activation = 'relu', padding = 'valid'))
classifier.add(layers.BatchNormalization())
classifier.add(MaxPooling2D(pool_size = (2, 2),strides=2))



classifier.add(Flatten())


classifier.add(layers.BatchNormalization())
classifier.add(Dense(units = 512, activation = 'relu'))


classifier.add(Dropout(0.2))



classifier.add(layers.BatchNormalization())
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dropout(0.2))
```

Fig. 9.4.3: Model implementasyonu (4. kısım)

Fig. 9.4.3'teki kod bloğu önceki paragrafta bahsedilen CNN modeline ek katmanlar eklemeye devam eder. Bu katmanlar arasında bir başka Conv2D katmanı (64 filtreli), ardından BatchNormalization ve MaxPooling2D katmanı bulunur. Daha sonra, evrişimsel katmanların çıktılarını düzleştirmek için bir Flatten katmanı eklenir. Ardından, iki tam bağlı (Dense) katman eklenir. İlk Dense katmanı 512 birime ve relu aktivasyonuna sahiptir, ardından bir Dropout (0.2 oranında) katmanı gelir. İkinci Dense katmanı 128 birime ve relu aktivasyonuna sahiptir, yine bir Dropout (0.2 oranında) katmanı ile birlikte kullanılır. Her Dense katmanından önce BatchNormalization uygulanmıştır.

```
classifier.add(Dense(units = 12, activation = 'softmax'))

adam_opt = optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
classifier.compile(optimizer = adam_opt, loss = 'categorical_crossentropy', metrics = ['accuracy'])

classifier.summary()

callback_es = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
patience=20, min_delta=0.0001, restore_best_weights=True)

batch_size = 32
epochs = 500

model1 = classifier.fit(training_set,
                        batch_size=batch_size,
                        epochs=epochs,
                        validation_data = (X_val,y_val),
                        shuffle=True,
                        callbacks = [callback_es])

classifier.evaluate(X_test,y_test)

best_model_accuracy = model1.history['accuracy'][np.argmin(model1.history['loss'])]
best_model_accuracy

from sklearn.metrics import confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Greens):

    fig = plt.figure(figsize=(10,10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
```

Fig. 9.4.4: Model implementasyonu (5. kısım)

Fig. 9.4.4'teki kod bloğu modelin son katmanını tanımlar, modeli derler ve eğitir. Son katman, 12 birimli (sınıf sayısı) bir Dense katmanı ve softmax aktivasyon fonksiyonu kullanır. Adam optimize edici (learning_rate=0.001 ile) kullanılarak model derlenir ve categorical_crossentropy kayıp fonksiyonu ile accuracy metriği kullanılır. EarlyStopping geri çağırma (callback_es) val_accuracy'yi izleyerek ve iyileşme 20 ardışık dönem boyunca durduğunda eğitimi durdurarak aşırı uyumu önlemek için yapılandırılmıştır. Model, model1 = classifier.fit() ile eğitilir, batch_size 32, epochs 500 olarak ayarlanır ve X_val, y_val doğrulama verisi olarak kullanılır. Eğitimin ardından, modelin test seti üzerindeki performansı classifier.evaluate(X_test, y_test) ile değerlendirilir. Son olarak, plot_confusion_matrix adlı özel bir fonksiyon tanımlanır ve karmaşıklık matrisini görselleştirmek için matplotlib kullanılır. Bu fonksiyon normalize edilmemiş bir karmaşıklık matrisi çizmek için ayarlanmıştır.

```
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


predY = classifier.predict(X_test)
predYClasses = np.argmax(predY, axis = 1)
trueY = np.argmax(y_test, axis = 1)


confusionMTX = confusion_matrix(trueY, predYClasses)
```

Fig. 9.4.5: Model implementasyonu (6. kısım)

Fig. 9.4.5'teki kod bloğu karmaşıklık matrisinin normalleştirilmesini ve görselleştirilmesini tamamlar. if normalize: koşulu altında, karmaşıklık matrisi (cm) toplam satır değerlerine göre normalize edilir ve float tipine dönüştürülür. Daha sonra, karmaşıklık matrisinin her bir hücresindeki metin, hücre değerine göre siyah veya beyaz renkli olacak şekilde yazdırılır (thresh = cm.max() / 2). plt.tight_layout() kullanılarak düzen optimize edilir. Eksik True label ve Predicted label etiketleri eklenir. Ardından, test setindeki tahminler yapılır: classifier.predict(X_test) kullanılarak predY elde edilir ve np.argmax kullanılarak hem tahmin edilen sınıflar (predYClasses) hem de gerçek sınıflar (trueY) one-hot kodlamadan dönüştürülür. Son olarak, confusionMTX = confusion_matrix(trueY, predYClasses) ile karmaşıklık matrisi hesaplanır.

```
plot_confusion_matrix(confusionMTX, classes = categ)

from sklearn.metrics import f1_score

print(f1_score(trueY, predYClasses, average='macro'))
print(f1_score(trueY, predYClasses, average='micro'))
print(f1_score(trueY, predYClasses, average='weighted'))
print(f1_score(trueY, predYClasses, average=None))

from sklearn.metrics import classification_report

print(classification_report(trueY, predYClasses, target_names=categ))

from sklearn.metrics import multilabel_confusion_matrix

multilabel_confusion_matrix(trueY, predYClasses)

history_df = pd.DataFrame(model1.history)
history_df.head()

plt.rcParams["figure.figsize"] = (7,6)


history_df.loc[:, ['loss', 'val_loss']].plot(title="Cross-entropy")
history_df.loc[:, ['accuracy', 'val_accuracy']].plot(title="Accuracy")

import numpy as np
import matplotlib.pyplot as plt

def display_image(image, title):
    plt.figure(figsize=(4, 4))
    plt.title(title)
    plt.imshow(image)
    plt.axis('off')
    plt.show()
```

Fig. 9.4.6: Model implementasyonu (7. kısım)

Fig. 9.4.6'daki kod bloğu esas olarak sınıflandırma modelinin değerlendirme metriklerini hesaplar ve bir sınıflandırma raporu oluşturur. plot_confusion_matrix fonksiyonu, confusionMTX ve categ (sınıf isimleri olduğu varsayılır) kullanarak bir karmaşıklık matrisi çizer. sklearn.metrics modülünden f1_score kullanılarak makro, mikro, ağırlıklı ve ağırlıksız ortalama F1 skorları hesaplanır ve yazdırılır. Ardından, sklearn.metrics modülünden classification_report kullanılarak trueY, predYClasses ve target_names (sınıf isimleri) ile kapsamlı bir sınıflandırma raporu (hassasiyet, geri çağırma, f1-skoru ve destek içeren) oluşturulur ve yazdırılır. Ayrıca, multilabel_confusion_matrix de hesaplanır. Son olarak, model1.history bir pandas.DataFrame'e dönüştürülür (history_df), başlık (ilk 5 satır) yazdırılır ve matplotlib'in figür boyutu (7,6) olarak ayarlanır. history_df kullanılarak kayıp (loss, val_loss) ve doğruluk (accuracy, val_accuracy) grafikleri çizilir. display_image adında özel bir

fonksiyon da tanımlanır, bir görüntüyü başlık ile birlikte gösterir, boyutu (4,4) olarak ayarlar ve eksenleri kapatır.

```
indices = [7, 11, 23, 37, 58]

for idx in indices:
    pred = np.argmax(classifier.predict(np.expand_dims(X_test[idx], axis=0)), axis=1)
    actual = np.argmax(y_test[idx])

    print(f"Model predicted category for X_test {idx} is: {pred}")
    print(f"Actual Category for X_test {idx} is: {actual}")
    print(f"Actual Category Name for X_test {idx} is: {categ[actual]}")


    display_image(X_test[idx] * 255, f"Processed Image (X_test[{idx}])")
    display_image(X_test_color[idx], f"Original Image (X_test_color[{idx}])")
    print("----------------------------------------------------------------")


classifier.save('classifier7.h5')
classifier.save_weights('classifier7.weights.h5')
```

Fig. 9.4.7: Model implementasyonu (8. kısım)

Fig. 9.4.7'de görüldüğü üzere, bu kod bloğu modelin tek tek örnekler üzerindeki performansını gösterir ve eğitilmiş modeli kaydeder. indices listesi (7, 11, 23, 37, 58) belirli test örneklerini seçmek için kullanılır. Bir döngü içinde, her idx için, modelin X_test[idx] üzerindeki tahmini yapılır (classifier.predict) ve np.argmax kullanılarak tahmin edilen sınıf (pred) ile gerçek sınıf (actual) belirlenir. Bu tahmin edilen ve gerçek sınıflar ile gerçek sınıf adı (categ[actual]) yazdırılır. Ardından, display_image fonksiyonu kullanılarak hem işlenmiş test görüntüsü (X_test[idx]) hem de orijinal renkli görüntü (X_test_color[idx]) gösterilir. Son olarak, eğitilmiş model classifier7.h5 olarak ve modelin ağırlıkları ayrı olarak classifier7.weights.h5 olarak kaydedilir.

9.5 Sonuçlar: Eğitilen model, Raspberry Pi'ye yüklenmiş olup kameradan alınan her yeni görüntü üzerinde sınıflandırma işlemi gerçekleştirmektedir. Elde edilen tahmin sonucu, bitkinin adı ve yabani olup olmadığı bilgisi ile birlikte web arayüzü üzerine yazdırılmaktadır. Test verisi üzerinde %85'in üzerinde doğruluk oranı elde edildiğinden dolayı sahadaki gerçek zamanlı kullanımda güvenilir bir şekilde karar verebileceği varsayılmıştır.

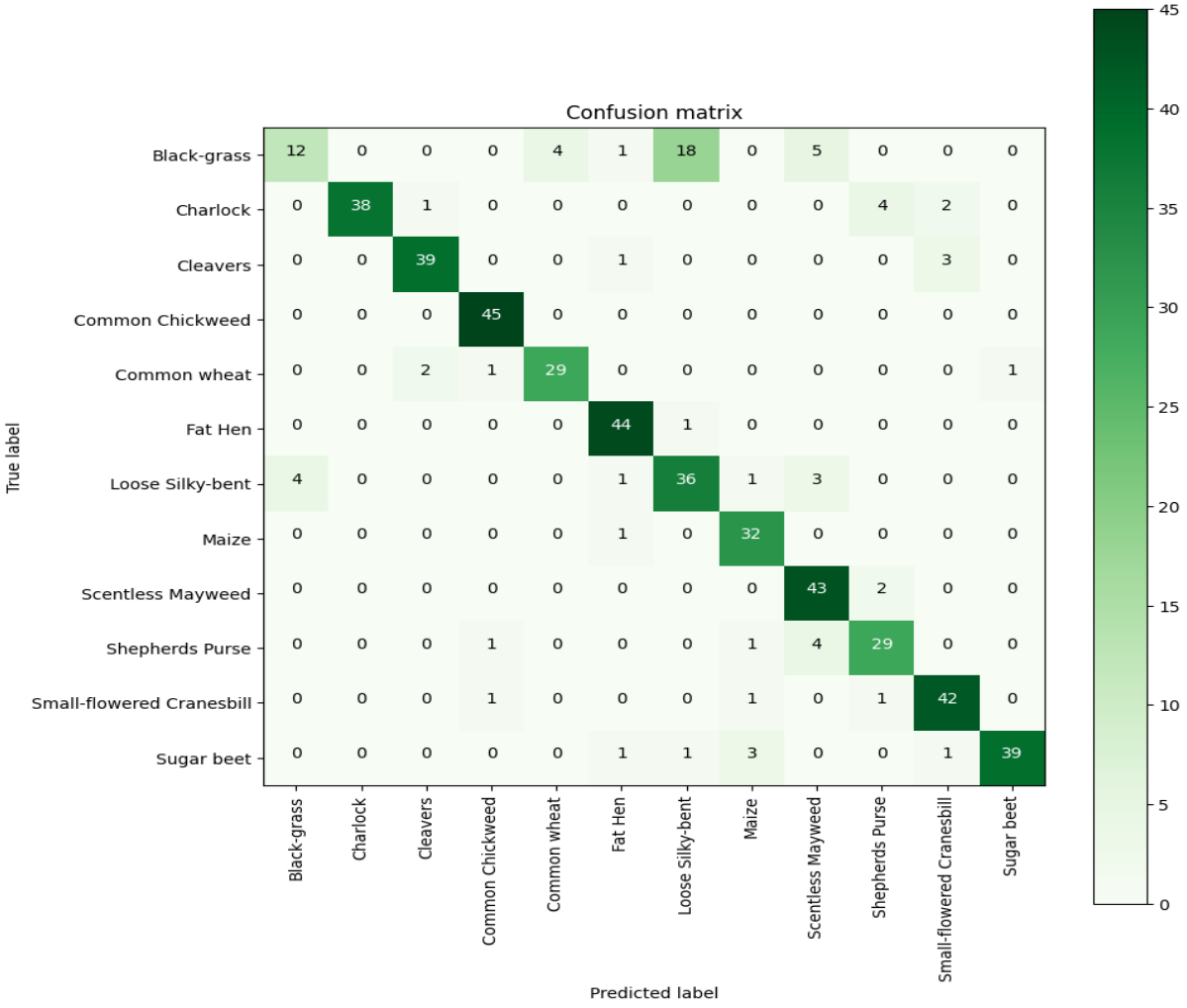

Fig. 9.5.1: Confusion matrix

Fig. 9.5.1'de gösterilen **Confusion matrix (karışıklık matrisi)**, modelin sınıflandırma performansını detaylı olarak gösterir. Gerçek sınıflar ile modelin tahmin ettiği sınıflar karşılaştırılarak hangi sınıfların doğru tahmin edildiği, hangi sınıfların ise karıştırıldığı anlaşılır. Bu sayede modelin güçlü olduğu sınıflar ile zorlandığı sınıflar tespit edilebilir ve sınıflar arasındaki hata kaynakları analiz edilir.

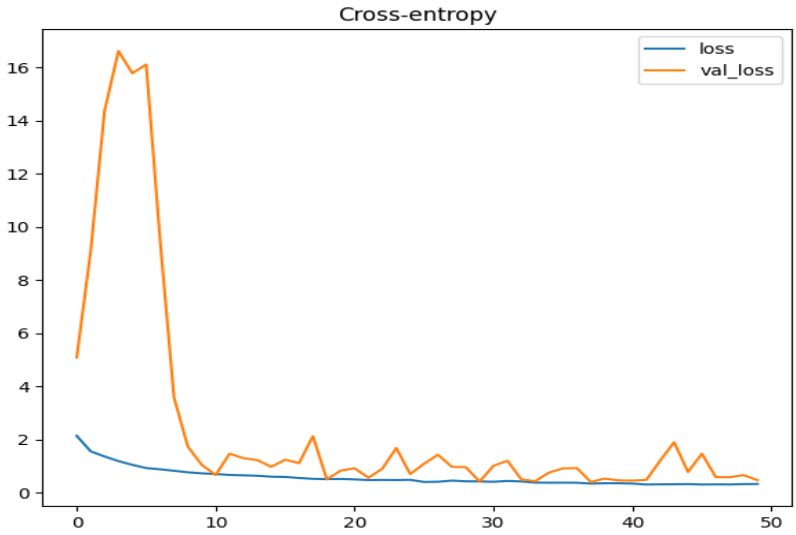

Fig. 9.5.2: Cross-entropy grafiği

Fig. 9.5.2'de gösterilen **Cross-entropy loss grafiği**, modelin eğitim ve doğrulama setlerindeki kayıp değerlerinin epochlar boyunca değişimini gösterir. Eğitim kaybının düzenli bir şekilde azalması, modelin veriyi öğrenebildiğini gösterirken; doğrulama kaybı başlangıçta çok yüksek olup kısa sürede düşmüş ve ardından düşük seviyelerde dalgalanmıştır. Bu durum, modelin öğrenme sürecinde doğrulama verisine hızlı bir adaptasyon sağladığını (10. Epoch) ancak zaman zaman doğrulama kaybının arttığını ve modelin stabil genelleme yapmakta zorlandığını göstermektedir.

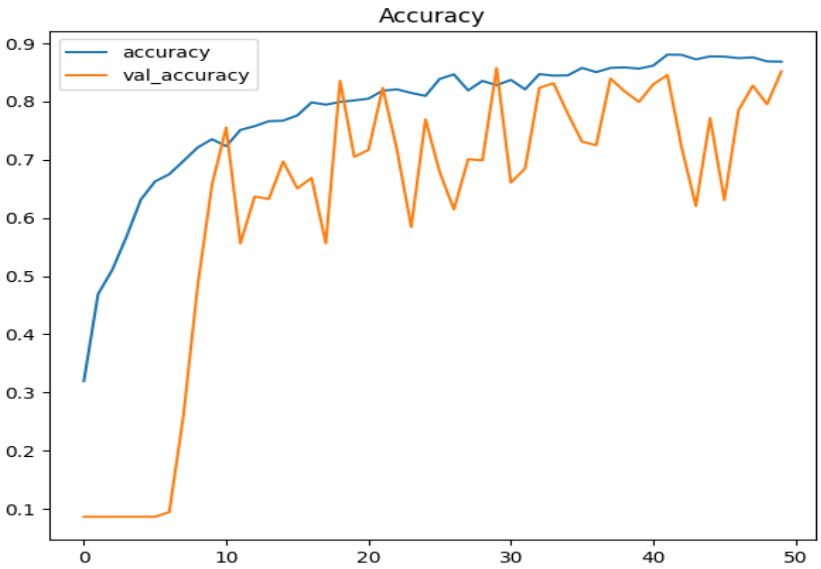

Fig. 9.5.3: Accuracy grafiği

Fig. 9.5.3'de gösterilen **Accuracy grafiği**, eğitim ve doğrulama doğruluklarının epochlar boyunca gelişimini göstermektedir. Eğitim doğruluğu sürekli artarak yaklaşık %85 seviyesine

ulaşmıştır. Ancak validation doğruluğunda belirgin dalgalanmalar görülmekte ve stabil bir doğruluk çizgisi oluşmamıştır. Bu durum modelin bazı epochlarda doğrulama verisine iyi genelleyebildiğini fakat zaman zaman overfitting yaptığı için doğrulama performansının düştüğünü göstermektedir.

10. Kullanıcı Etkileşimi ve Görüntüleme Arayüzü

Web arayüzü, hem hareket kontrolü hem de görüntü işleme sistemine entegre bir yapıdadır. Kamera görüntüsü, Raspberry Pi üzerinde hazırlanan ve görüntü işleme modeliyle bütünleşik halde çalışan bir Python kodunun terminal üzerinden başlatılmasıyla elde edilmektedir. Sisteme entegre edilen keystroke fonksiyonu sayesinde, kullanıcının tuş kombinasyonlarıyla web kamerası aracılığıyla fotoğraf alması sağlanmış; ayrıca bu görüntülerin otomatik olarak Raspberry Pi ekranında gösterilmesini sağlayan bir görsel ön izleme fonksiyonu da yazılıma eklenmiştir.

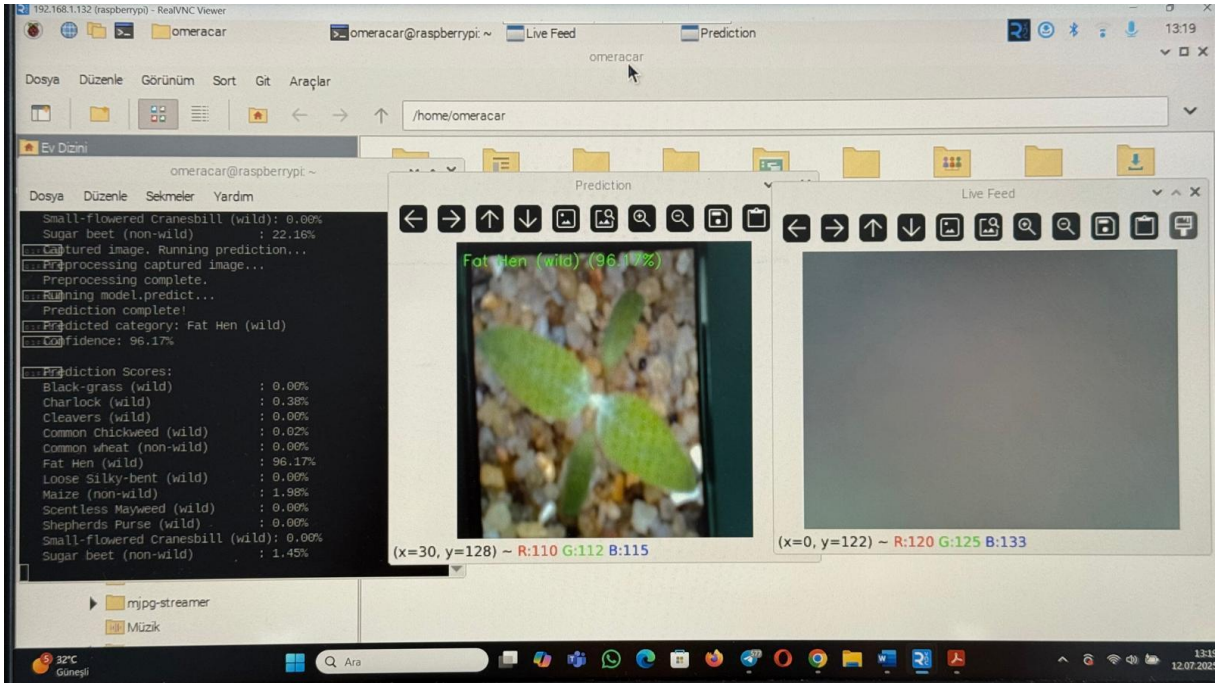

Fig. 10.1: Yabani Bitki Sınıflandırma İşlemi (Live Feed: Kamera Görüntüsü, Prediction: Çıktı Görüntüsü)

Hareket komutları için HTML tabanlı ayrı bir arayüz oluşturulmuştur. Bu arayüz, cep telefonu üzerinden erişilerek ileri, geri, sağa, sola ve dur gibi butonlar ile hem aracın hareketi hem de pan-tilt mekanizmasının yönlendirilmesi sağlanmaktadır. Web arayüzü ile araç arasında bağlantı, ESP32 tarafından oluşturulan Wi-Fi ağı üzerinden sağlanmış, böylece harici bir internet bağlantısına gerek kalmadan sistemin kontrolü mümkün kılınmıştır.

Sınıflandırma sonucunda, model tarafından tanınan bitkinin ismi ile birlikte bu bitkinin yabani mi yoksa tarım ürünü mü olduğu bilgisi tespit edilmekte ve Fig. 10.1'de görüldüğü üzere ekranın üst kısmındaki görüntü üzerine dijital olarak yazdırılmaktadır. Bu bilgi, kullanıcıya anlık olarak sunulmakta ve sistemin doğru karar verip vermediği doğrudan gözlemlenebilmektedir.

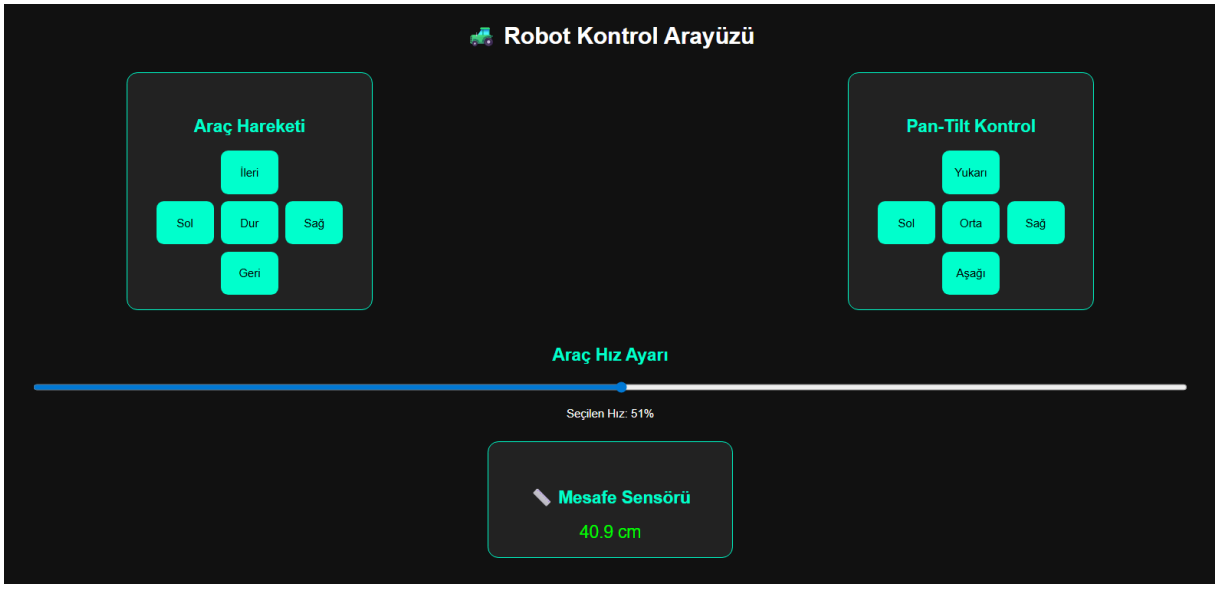

Fig. 10.2: Web tabanlı kontrol paneli

Fig. 10.2'de gösterilen arayüz, geliştirilen tarım robotunun manuel olarak uzaktan kontrol edilmesini sağlayan web tabanlı bir kontrol panelidir. Arayüzde, robotun ileri, geri, sağa ve sola hareket etmesini sağlayan "Araç Hareketi" butonları yer almaktadır. Bu sayede kullanıcı, robotun yönünü doğrudan belirleyebilmektedir. Ayrıca, robot üzerinde bulunan kamera sistemi için "Pan-Tilt Kontrol" bölümü eklenmiştir. Bu bölümdeki butonlar, kameranın yukarı, aşağı, sola, sağa ve merkez pozisyona yönlendirilmesini sağlar; böylece bitkilerin farklı açılardan görüntülenmesi mümkündür. Arayüzün orta kısmında yer alan "Araç Hız Ayarı" bölümü sayesinde kullanıcı, robotun hareket hızını yüzde cinsinden ayarlayabilir ve seçilen hız değeri ekranda anlık olarak görüntülenir. Alt kısımda bulunan "Mesafe Sensörü" kutucuğu ise, robotun önündeki engellere olan uzaklığı gerçek zamanlı olarak santimetre cinsinden göstermektedir. Bu sensör verisi, robotun çarpışmalardan kaçınmasını ve çevresel farkındalığını artırmasını sağlar. Genel olarak bu arayüz, kullanıcıya robot üzerinde tam kontrol imkanı sunarak hem hareket hem de çevresel algılama görevlerini kolaylaştırmaktadır.

11. Bitmiş Proje Görselleri

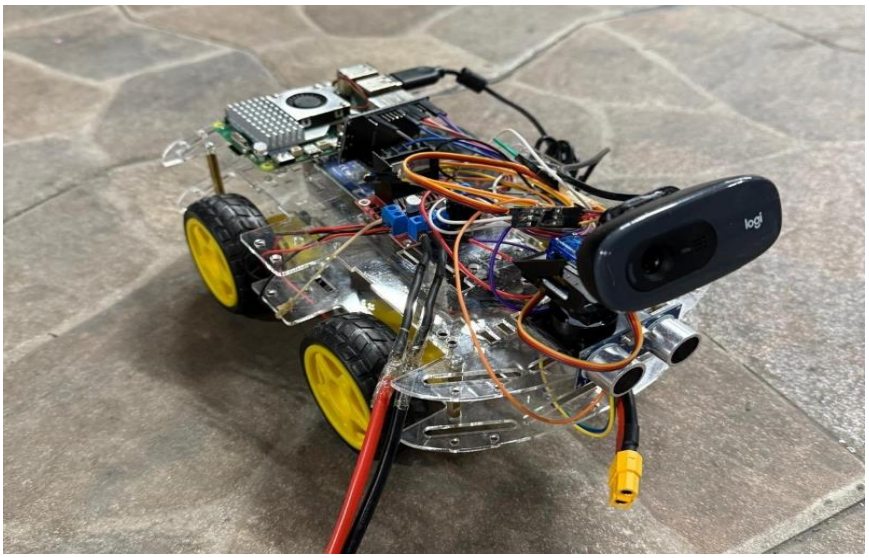

Fig. 11.1: Projenin tamamlanmış hali (Önden görünüm)

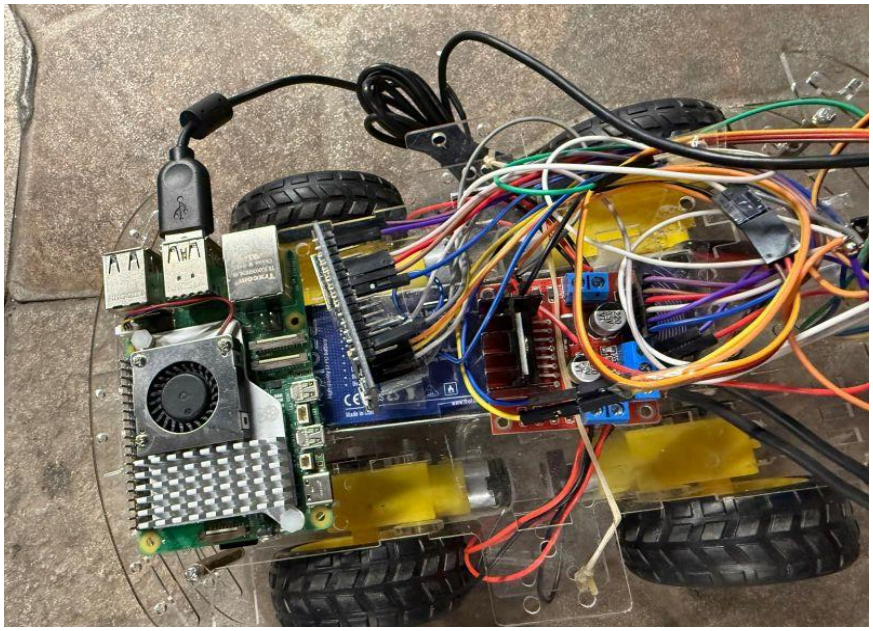

Fig. 11.2: Projenin tamamlanmış hali (Üstten görünüm)

Fig. 11.1 ve Fig. 11.2 görselleri, yapay zeka destekli bitki sınıflandırma robotunun tamamlanmış halini göstermektedir. Robot, dört tekerlekli mobil bir şase üzerine inşa edilmiş olup üzerinde Raspberry Pi, Arduino, motor sürücü, USB kamera ve mesafe sensörü gibi temel donanım birimleri yer almaktadır. Bu donanımlar, görüntü işleme, motor kontrolü ve kablosuz iletişim görevlerini entegre şekilde yerine getirecek şekilde monte edilmiştir.

12. Kaynakça

- Namratha Makanapura *et al* 2022 *J. Phys.: Conf. Ser.* **2161** 012006, Classification of plant seedlings using deep convolutional neural network architectures, doi: 10.1088/1742-6596/2161/1/012006
- Kundur, N.C. and Mallikarjuna, P.B. 2022. Deep Convolutional Neural Network Architecture for Plant Seedling Classification. *Engineering, Technology & Applied Science Research*. 12, 6 (Dec. 2022), 9464–9470. DOI:https://doi.org/10.48084/etasr.5282.

- Ruixiang Li, Jun Pan, Yingdong Pi, Mi Wang, UAV Image Stitching via Global Optimal Seamline Detection and Local Alignment With Seamline Constraint, IEEE Transactions on Geoscience and Remote Sensing, 10.1109/TGRS.2025.3563397, **63**, (1-14), (2025).

- The PLOS ONE Staff (2017) Correction: Plant species classification using flower images—A comparative study of local feature representations. PLOS ONE 12(3): e0175101. https://doi.org/10.1371/journal.pone.0175101

- C. R. Alimboyong, A. A. Hernandez and R. P. Medina, "Classification of Plant Seedling Images Using Deep Learning," *TENCON 2018 - 2018 IEEE Region 10 Conference*, Jeju, Korea (South), 2018, pp. 1839-1844, doi: 10.1109/TENCON.2018.8650178.

- N. R. Rahman, M. A. M. Hasan and J. Shin, "Performance Comparison of Different Convolutional Neural Network Architectures for Plant Seedling Classification," *2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT)*, Dhaka, Bangladesh, 2020, pp. 146-150, doi: 10.1109/ICAICT51780.2020.9333468.

- Nkemelu, D.K., Omeiza, D., Lubalo, N. (2018). Deep Convolutional Neural Network for Plant Seedlings Classification. arXiv preprint arXiv:1811.08404.

- Kaggle Dataset: *Plant Seedlings Classification*. https://www.kaggle.com/competitions/plant-seedlings-classification