Teknoloji Fakültesi

T.C.
MARMARA ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

OTONOM İÇ MEKAN HİZMET ARACI

LİSANS BİTİRME PROJESİ

ESAD ACER

BAYRAM EFE UZUNOĞLU

DR. ÖĞR. ÜYESİ ERSİN TOPTAŞ

İSTANBUL,2025

Teknoloji Fakültesi

T.C.
MARMARA ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ

OTONOM İÇ MEKAN HİZMET ARACI

LİSANS BİTİRME PROJESİ

ESAD ACER
(170221032)
BAYRAM EFE UZUNOĞLU
(170221052)

DR. ÖĞR. ÜYESİ ERSİN TOPTAŞ

İSTANBUL,2025

İÇİNDEKİLER

ŞEKİL LİSTESİ

KISALTMALAR

PD: Proportional-Derivative (Oransal-Türev Kontrol)

PID: Proportional-Integral-Derivative (Oransal-Integral-Türev Kontrol**)**

ESP: Espressif Systems Platform

ESP8266: Wi-Fi modülü/mikrodenetleyici

ESP8266: Gelişmiş Wi-Fi/Bluetooth özellikli Espressif mikrodenetleyici

PWM: Pulse Width Modulation (Darbe Genişlik Modülasyonu)

RX: Receive (Alıcı veri pini)

TX: Transmit (Verici veri pini)

Wi-Fi: Wireless Fidelity

GPIO: General Purpose Input/Output (Genel Amaçlı Giriş/Çıkış)

UWB: Ultra-Wideband (Aşırı Geniş Bant)

BLE: Bluetooth Low Energy (Düşük Güç Tüketimli Bluetooth)

RF: Radio Frequency (Radyo Frekansı)

IMU: Inertial Measurement Unit (Atalet Ölçüm Birimi)

JSON: JavaScript Object Notation (Veri formatı)

HTTP: Hypertext Transfer Protocol (İnternet İletişim Protokolü)

API: Application Programming Interface (Uygulama Programlama Arayüzü)

REST: Representational State Transfer

A*: A-Star Algoritması

LiPo: Lithium Polymer (Lityum Polimer batarya)

3B: Üç Boyutlu

ÖZET

Otonom hareket, aracın nerede olduğunu sürekli kontrol etmeyi ve buna göre hareket etmeyi gerektirdiğinden, konumlandırma bu alanda kritik bir rol oynar. Dış mekanlarda GPS gibi sistemler konumlandırma sağlarken, iç mekanlarda bu sistemler kullanılamaz.

Bu nedenle, iç mekanlarda konumlandırma, genellikle çeşitli sinyal üreten referans noktalarına olan uzaklıkların hesaplanmasıyla gerçekleştirilir. Bu yöntemlerin başlıca dezavantajı, kullanılacak yere önceden bir kurulum gerektirmesi ve ortamın buna uygunluğunun sağlanmasıdır.

Mekan içi konumlandırmada kullanılan bir diğer önemli yöntem ise odometri sistemleridir. Odometri, bir aracın veya robotun katettiği mesafeyi ve yön değişimini, tekerlek dönüşleri veya ivmeölçer/jiroskop gibi dahili sensörlerden gelen verilerle tahmin etme yöntemidir.

Kendi aracımızda ise odometri sistemini tekerlek dönüşlerinden aldığımız veriler ile sağlayacağız. Araçta bu sistemi kullanarak belirlenmiş bir harita üzerinde otonom hareket sağlayacağız. Aracın temel amacı, kendi tasarladığımız web sitesi üzerinden istenilen konuma otonom olarak hareket etmektir. Böylece, aracın kendi sistemlerini kullanarak istenen ortamda otonom hareket ve konumlandırma yeteneği sağlanacaktır.

1. GİRİŞ

1.1 LİTERATÜR ÖZETİ

Mobil robotlar, bir ortamda serbestçe hareket edebilen ve belirli görevleri otonom şekilde gerçekleştirebilen robotlardır.[1] Mobil robotlar, kapalı alanlarda harita üzerinden belirlenen bir hedefe doğru ulaşırken, engellerden kaçınmalı ve doğru yön bulmalıdır.[2] Kapalı mekanda konumlandırma, uydu tabanlı GPS sistemlerinin yetersiz kaldığı iç ortamlarda otonom navigasyon ve hizmet araçları için kritik bir öneme sahiptir. Literatürde bu alanda çeşitli yaklaşımlar bulunmaktadır. Sinyal tabanlı konumlandırma sistemleri, genellikle Wi-Fi, Bluetooth Düşük Enerji (BLE) veya Ultravide Bant (UWB) gibi radyo frekansı (RF) sinyallerini kullanarak konum belirler. Wi-Fi parmak izi, ortamdaki sinyal güçlerinin bir haritasını çıkararak konum tahmini yaparken, BLE işaretçileri sinyal gücü veya zaman farkı prensipleriyle çalışır. UWB ise yüksek zaman çözünürlüğü sayesinde milimetre hassasiyetinde konumlandırma sunabilir. Ancak bu yöntemler genellikle dış altyapı kurulumu gerektirir ve çevresel faktörlerden etkilenebilir.

Alternatif olarak, sensör tabanlı yaklaşımlar bir aracın veya kişinin kendi dahili sensörlerinden gelen verilerle hareketini takip eder. Bu kategoride öne çıkan yöntem odometridir. Odometri, tekerlek dönüşleri veya atalet ölçüm birimleri (IMU- ivmeölçer, jiroskop) gibi sensör verilerini kullanarak kat edilen mesafeyi ve yön değişimini tahmin eder. Tekerlekli odometri robotik sistemlerde yaygınken, inersiyal odometri (Pedestrian Dead Reckoning- PDR) taşınabilir cihazlarda kullanılır. Odometri, zamanla kümülatif hata biriktirme eğiliminde olsa da diğer konumlandırma sistemleriyle (sensör füzyonu) birleştirilerek konum doğruluğunu önemli ölçüde artırabilir ve dış altyapı bağımlılığını azaltır.

1.2 PROJENİN AMACI

Bu projenin temel amacı, kapalı mekanda otonom hareket edebilen bir hizmet aracı tasarlamak ve geliştirmektir. Proje kapsamında belirlenen ana hedefler şunlardır:

- **Kapalı Mekanda Konum Tespiti:** Aracın, dışarıdan herhangi bir harici konumlandırma sinyali olmadan (GPS gibi) kapalı ortam içerisinde kendi konumunu doğru bir şekilde belirleyebilmesi. Bu, özellikle aracın kendi dahili sensörlerinden (odometri) faydalanılarak gerçekleştirilecektir.
- **Otonom Hareket Yeteneği:** Aracın, belirlenen bir kat planının haritası üzerinde tanımlanan çeşitli hedef noktalara otonom ve engellerden kaçınarak gidebilmesi.
- **Kullanıcı Dostu Kontrol Arayüzü:** Kullanıcıların, geliştirdiğimiz bir web sitesi üzerinden aracı kolayca izleyebilmesi ve istenilen konuma otonom olarak gönderebilmesini sağlayan sezgisel bir arayüz sunulması.

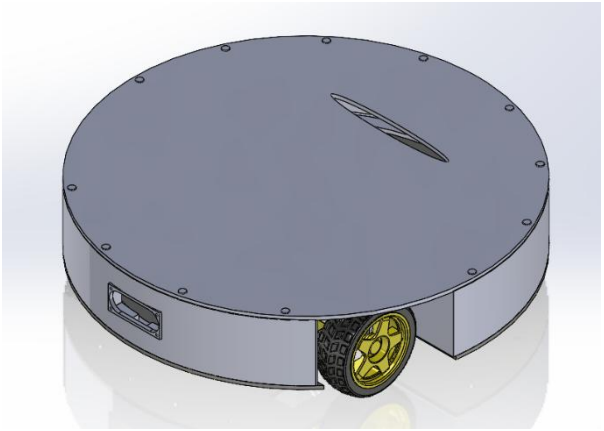
Bu proje, otonom hizmet araçlarının kapalı mekanlardaki kullanımını yaygınlaştırmak, operasyonel verimliliği artırmak ve insan müdahalesini azaltmak için pratik ve esnek bir çözüm sunmayı hedeflemektedir.

2. MATERYAL VE YÖNTEM

Projenin planlanması ve üretimi mekanik, donanım ve yazılım olmak üzere üç ana başlıkta incelendi.

2.1 MEKANİK

Aracın mekanik sistemi tasarlanırken gereklilikleri karşılaması, üretim rahatlığı ve maliyeti göz önüne alınmıştır. Mekanik sistemde ilk aracın tasarımı Solidworks programı üzerinden elde edildi. Daha sonra bu tasarımın üretim planlaması yapıldı ve bu planlamaya göre üretim yapılıp araç imal edildi.



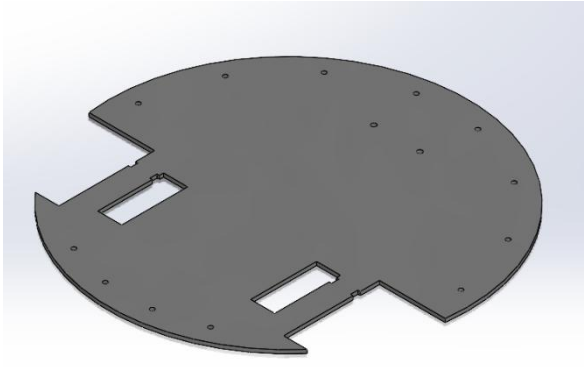
Şekil 1 Araç İzometrik Görüntüsü

2.1.1 Mekanik Tasarım

Aracın mekanik tasarımı Solidworks uygulaması üzerinden yapıldı. Bu tasarım yapılırken ilk olarak aracın ana şekli planlanıp tasarlandı. Aracın rahat hareketi ve görevine uygunluğu düşünülüp ana yapısının yuvarlak şeklinde olmasına dair karar alındı. Bu karardan sonra aracın boyutları hakkında planlama yapıldı. Bu planlama yapılırken aracın içinde bulunacak elektronik sistemin ve batarya boyutları dikkate alındı.

2.1.1.1 Alt Tabla

Aracın boyutları belirlendikten sonra tüm sistemin oturacağı alt tabla tasarımına başlandı. Bu tasarımda belirleyici kısım motor ve motor ekipmanı olacağı için seçilen motorun ölçüleri ve yapısına göre motor tutucu bir sistem tasarlandı. Ayriyeten motorla birlikte tekerlek ve

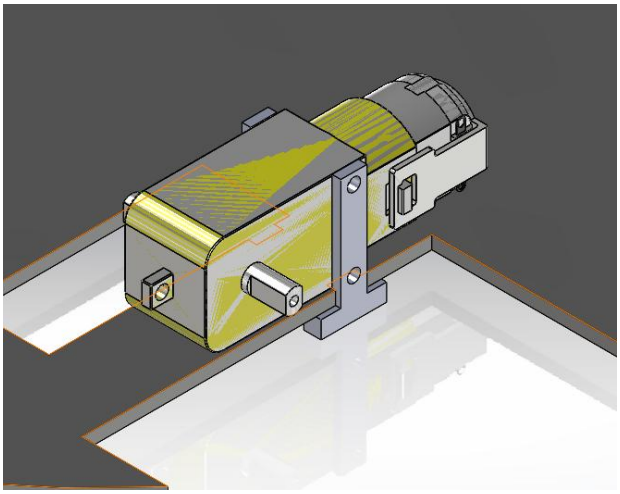


Şekil 2 Araç Alt Taban

enkoderlerinde bulunduğu sistem göz önüne alınarak tasarım yapıldı. Ayrıca alt tabanı yan duvarlarında sabitleneceği deliklere sahip olmalıdır.

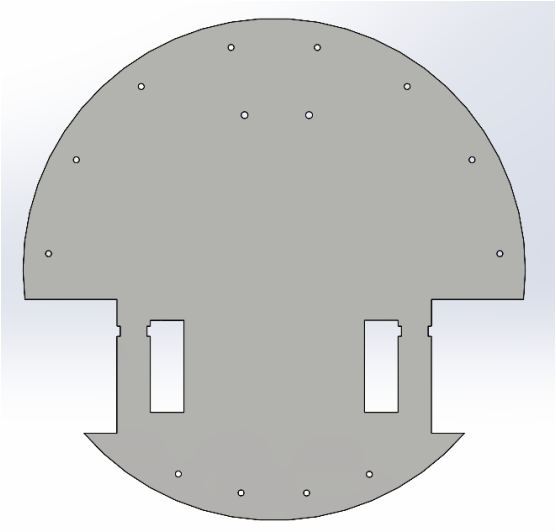
Aracın tekerlek yapısında iki tekerlek önde ve birbirine paralel konumda olup arka kısımda ise sarhoş teker kullanılmıştır. Bu planlamaya göre önde iki motor sabitleme kısmı, tekerlek ve enkoder boşlukları konuldu.

Motorlar yanlarında bulunan delikler kullanılarak t şeklinde bir parçayla araca sabitlendi. Motor iki tarafından sabitlenerek sağlamlık elde edildi.



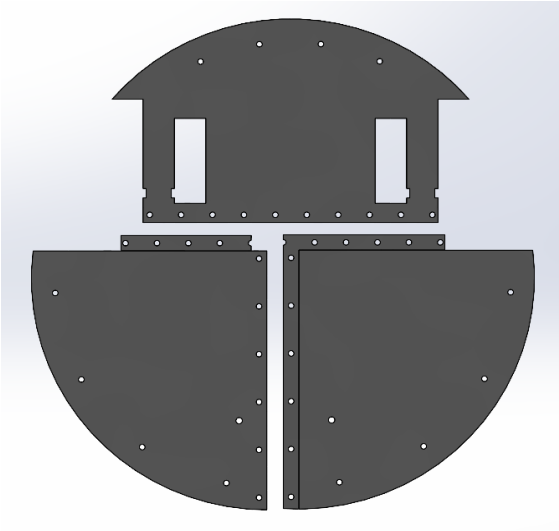
Şekil 3 Motor ve Sabitleme Parçası

Aracın alt tablasında yan duvarları sabitlemek için delikler açıldı bu delikler M3 vidaların boyutlarına göre açıldı.



Şekil 4 Alt Tabla Üstten Görünümü

Alt tablanın üretiminde 3B yazıcı ile üretiminin kararını aldık. Ama 3B yazıcıların basım ölçüleri alt tablanın ölçülerine uyum sağlamamaktaydı. O yüzden alt taban 3 parçaya bölündü. Bu parçalar M3 vida ve yapıştırıcı ile sabitlenmek üzere tasarlandı.

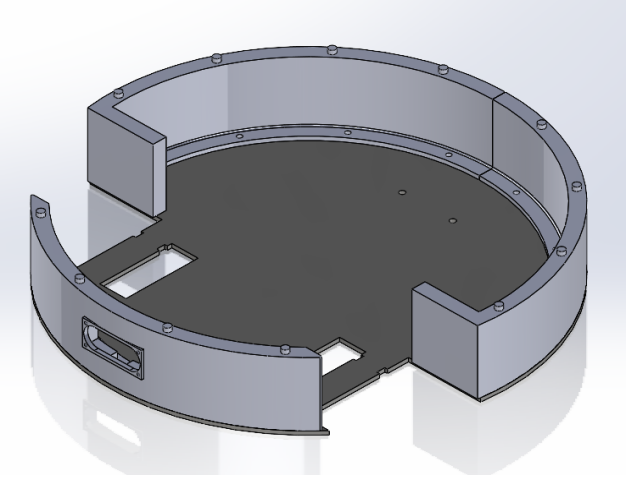


Şekil 5 Alt Tabla Bölünmüş

2.1.1.2 Yan Duvarlar

Yan duvarlar aracın etrafını kapatan ve alt üst tablaya bağlanıyorlar. Bu duvarlarda önemli faktörlerden biri yüksekliktir. Çünkü içerideki donanım sisteminin içeriye oturması gerekmektedir ve aracın taşıyabilir olması gerekmektedir.

Yan duvarlarda alt tabla gibi 3 parça şeklinde tasarlandı. Ve alt ve üst tablaya bağlantısı için belli delikler açıldı. Ön kısımdaki duvarda ise mesafe sensörünün oturması için sensör şeklinde bir boşluk açıldı.

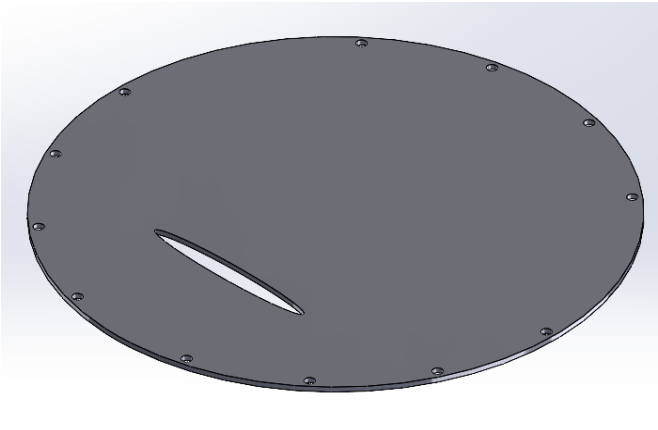


Şekil 6 Yan Duvarlar

Bu duvarlarda alt tabla ile vida yardımıyla sabitlenmektedir. Üst kısmı ise üst tablada bulunan deliklere oturacak şekilde tasarımı yapıldı.

2.1.1.3 Üst Tabla

Aracın üst tablası alt tabla boyutunda tam yuvarlak bir biçimde tasarlandı. Üst tablada bulunan delikler duvarlara tam oturması için tasarlandı.



Şekil 7 Araç Üst Tablası

2.1.2 Mekanik Üretim

Araç üzerindeki her parça iki yöntem kullanılarak üretilmiştir. Bunlar lazer kesim ve 3B yazıcı ile eklemeli imalattır. Bu iki yöntemi kullanmamın sebebi hem rahat erişilebilir olması hem ucuz olmasıdır.

2.1.2.1 Lazer Kesim ile Üretim

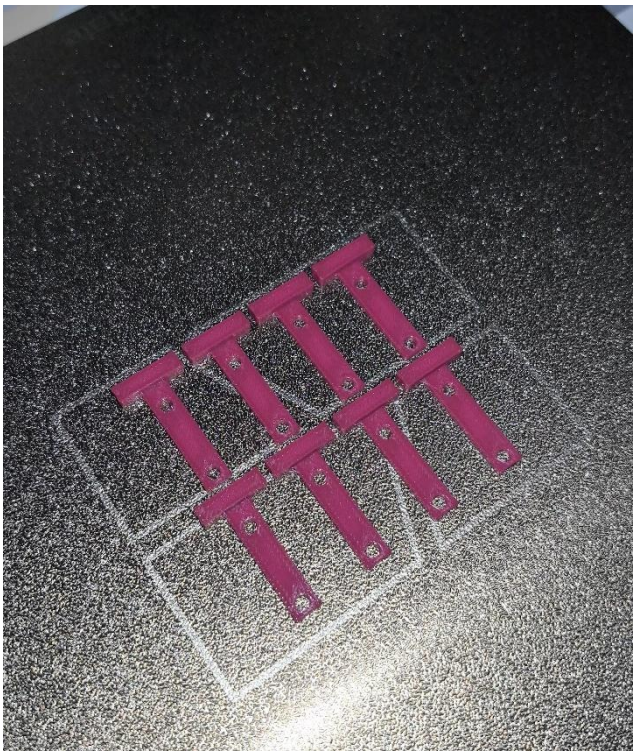
Lazer kesim ile başlarda hem alt hem üst tablanın üretimi gerçekleştirildi. Ama alt tablanın tasarımındaki sık değişiklikler lazer üretimin maliyetini arttırmış oldu. Bu yüzden araç üzerimde sadece üst tabla lazer kesim ile üretilmiş olmuş pleksiglas malzeme kullanılmıştır. Pleksiglas malzeme seçilmesinin sebebi ise hafif olmasına rağmen dayanıklı ve ucuz bir malzeme olmasıdır ve işlemesi kolaydır.

Şekil 8 Lazer Kesim

2.1.2.2 3 Boyutlu Yazıcı ile Üretim

Aracın büyük bir kısmı 3B yazıcı kullanılarak üretilmiştir. Bu üretim yöntemi; ulaşılabilirliği, düşük maliyeti ve kullanım kolaylığı ile öne çıkmaktadır. Bu avantajları nedeniyle tercih edilmiştir. Aracın alt tablası, yan duvarları, motor tutucuları ve enkoderleri 3B yazıcı ile üretilmiştir.

Parçalar yazıcıyla üretilirken kullanılacak filament türü, baskı hızı ve dolgu yoğunluğu gibi faktörler göz önünde bulundurulmuştur. Bu parametreler, parçaların dayanıklılığı ve genel kalite seviyeleri üzerinde belirleyici rol oynamaktadır.



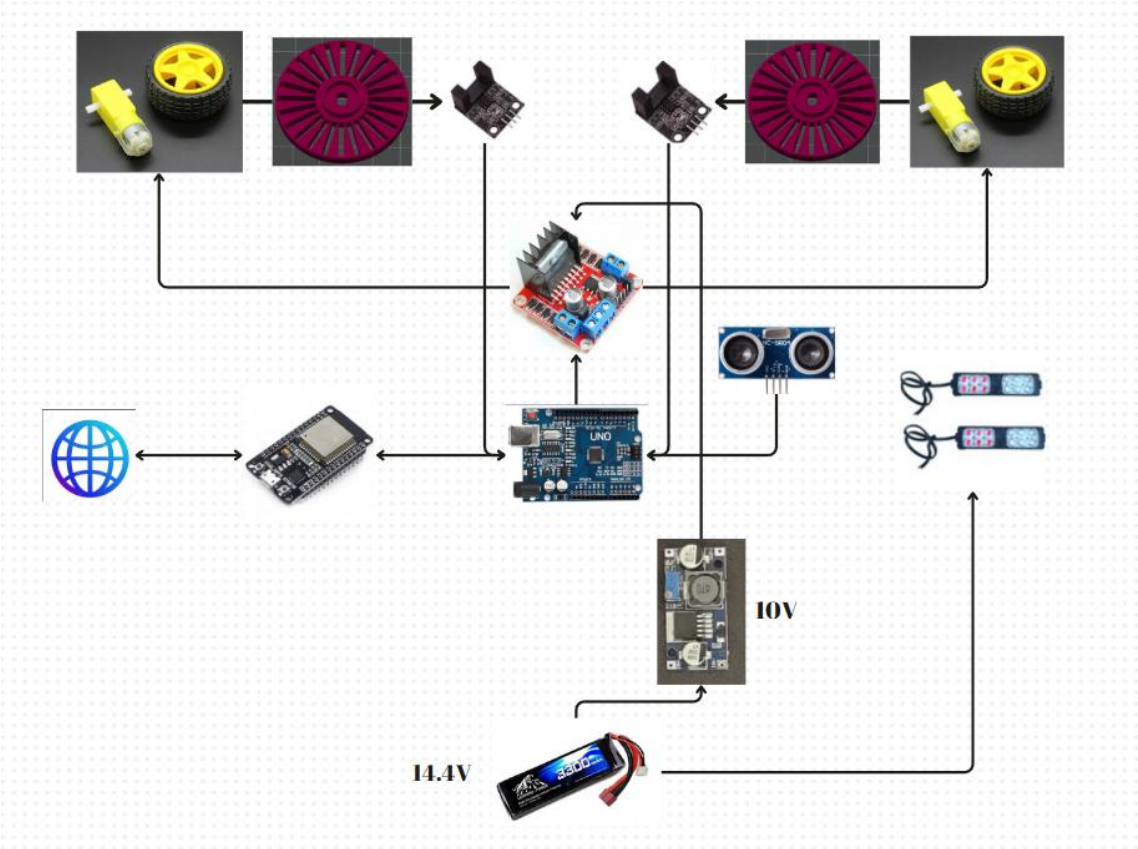
Şekil 9 Motor Tutucu Baskısı

Parçaların üretiminde iki farklı yazıcı kullanılmıştır. Yan duvarlar Ender model yazıcı ile üretilirken, alt tabla, motor tutucu ve enkoder parçalarında BambuLab yazıcı tercih edilmiştir.

Şekil 10 Aracın Üretilmiş Hali

2.2 DONANIM

Aracın donanım yapısında aracın hareket, kontrol ve güç kısmını bulundurmaktadır. Aracın içinde motorlar, motor sürücü, batarya, kontrol kartları, enkoder sayıcılar, voltaj regülatörü bulunmaktadır.



Şekil 11 Araç Şematiği

Bu projede geliştirilen aracın elektronik altyapısı, hareket, algılama, kontrol ve haberleşme fonksiyonlarını bir arada sunacak şekilde yapılandırılmıştır. Sistemin merkezinde yer alan Arduino UNO mikrodenetleyici kartı, tüm bileşenlerin kontrolünü sağlamaktadır.

Araçta yer alan iki adet DC motor, motor sürücü kartı üzerinden beslenmekte ve kontrol edilmektedir. Motorlara entegre edilen enkoderler sayesinde tekerlek dönüş bilgileri alınarak sistemde hız ve yön kontrolü gerçekleştirilmektedir. Enkoder verileri Arduino tarafından işlenerek geri besleme sağlanmaktadır. Ön kısımda konumlandırılan ultrasonik sensör, çevresel engellerin algılanmasını sağlamakta ve çarpışma önleyici sistemin temelini oluşturmaktadır.

ESP8266 modülü, aracı bir web arayüzüne bağlamakta ve bu arayüzden alınan hareket verileriyle araç kontrol edilmektedir. Sistemin enerji ihtiyacı, 14.4V çıkışlı bir LiPo batarya ile sağlanmakta; voltaj dönüştürücü devre aracılığıyla bu enerji 10V seviyesine indirilerek motor sürücü, Arduino ve diğer bileşenlere uygun şekilde dağıtılmaktadır.

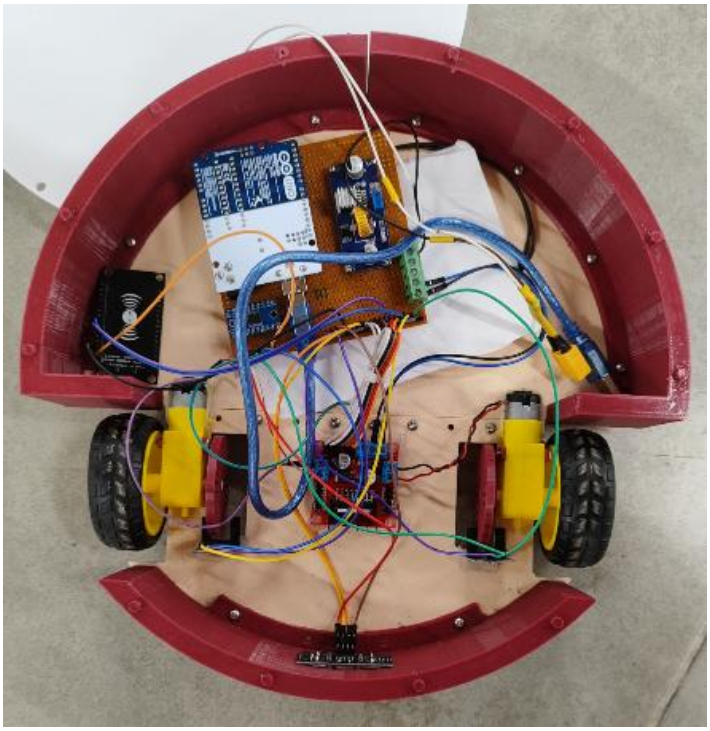
Birbiriyle haberleşen Arduino ve ESP8266 kartı iki farklı voltaj seviyesinde çalıştıkları için direkt bağlantı ile haberleşmesi gerçekleştirilememektedir. Ara bir mantık kanal düşürücü kullanılmıştır.

Araç içerisinde, kendi hazırladığımız bir delikli plaket devresi bulunmaktadır. Bu devre üzerinde Arduino kartı, voltaj regülatörü ve güç kısmına ait bağlantılar yer almaktadır. Bu devreyi yapmamızdaki amaç; kablo karmaşasını azaltmak, bağlantıların daha düzenli ve doğru şekilde yapılmasını sağlamak ve olası kısa devre durumlarını önlemektir.

Devrede, batarya, ışıklar ve motorlar için giriş-çıkış konnektörleri bulunmaktadır. Bataryadan gelen voltaj önce voltaj regülatörüne iletilmekte, burada uygun seviyeye düşürülerek motor sürücüye aktarılmaktadır.

Ayrıca bu devre üzerinde yer alan Arduino kartının dijital pinleri daha düzenli hâle getirilmiş, sıralı şekilde dizilerek bağlantı kolaylığı sağlanmıştır. Devre üzerinden güç dağıtımı yapılarak Arduino kartına ve sensörlere gerekli voltaj sağlanmıştır.

Bu devre sayesinde kablo kalabalığı ve kısa devre problemleri ortadan kaldırılmış; motor sürücü kontrol çıkışları, sensör bağlantıları ve güç hatları doğrudan bu kart üzerinden sağlanarak sistemin kullanımı kolaylaştırılmıştır.



Şekil 12 Araç Elektroniğinin Görüntüsü

Sistemin yapısında kullanılan ana bileşenlerin detaylı açıklaması alt başlıklarda belirtilmiştir.

2.2.1 Güç Sistemi

Aracın güç sisteminde bir batarya, on-off switch, voltaj regülatörü, motor sürücü, motorlar bulunmaktadır.

2.2.1.1 Batarya

Sisteme enerji sağlamak için LiPo batarya tercih edilmiştir. Bu batarya; yüksek depolama kapasitesi, yüksek akımda çalışabilme yeteneği ve hafifliği gibi özellikleriyle diğer batarya türlerinden ayrılmaktadır. Projemizin küçük ve hafif bir enerji kaynağına ihtiyaç duyması nedeniyle bu batarya uygun bulunmuştur.

Şekil 13 Batarya

Batarya, 2250 mAh kapasiteye sahip olup üç hücreden oluşmaktadır. Üç hücreli yapısı sayesinde yaklaşık 14.4V enerji sağlamaktadır.

Lityum Polimer (LiPo) bataryalarda, pilin çalışma süresi genellikle kapasite ile çekilen akım arasındaki ilişki kullanılarak tahmin edilir. Bu ilişki şu formülle ifade edilir:

$t = (Q / I)\ x\ 60$

Burada t (dk) çalışma süresi, Q batarya kapasitesi (mAh), I ise sistemin çektiği akım (mA) olarak tanımlanır. Bu formül, temel elektrik mühendisliği prensiplerine dayanmaktadır ve pilin teorik olarak ne kadar süre çalışabileceğini hesaplamak için yaygın olarak kullanılmaktadır.[3]

Aracımız ortalama olarak 1.5A harcamaktadır. Bu hesaplamaya göre aracımız devamlı çalışma durumunda 1.5 saatlik bir kullanım süresine sahiptir.

Bataryanın pozitif kısmı bir on-off switch ile bağlantısı vardır. Bu switch ile aracın enerjisi rahat bir şekilde kesilmekte veya enerji verilmektedir.

2.2.1.2 Voltaj Regülatörü

Sistemin tek enerji kaynağı bataryadır. Ancak sistemde kullanılan diğer bileşenler, bataryanın doğrudan sağladığı voltajla çalışamamakta; bu voltaj değeri onlar için fazla gelmektedir. Bu nedenle sistemde bir voltaj düşürücü regülatör kullanılarak gerilim seviyesi uygun hâle getirilmiştir ve her bileşen ihtiyaç duyduğu voltajla çalışacak şekilde beslenmektedir.

Şekil 14 LM2596 Voltaj Regülatörü

Regülatör kartı olarak LM2596 3A modeli tercih edilmiştir. Bu modelin tercih edilme nedeni, sağladığı voltaj dönüşümleri ve taşıyabildiği akım kapasitesinin sistemimiz için yeterli olmasıdır. Güç sisteminde kullanılan LM2596 kartı, kendi hazırladığımız delikli plaket devresinin içerisinde yer almaktadır.

2.2.1.3 Motorlar

Aracın hareketini sağlayan, ön tarafta konumlandırılmış iki adet motor ve bu motorlara bağlı tekerlekler bulunmaktadır. Dönüş hareketlerinde daha iyi manevra kabiliyeti sağlamak amacıyla, motorlar ön kısımda birbirine paralel şekilde yerleştirilmiştir. Motor olarak 9–12V

aralığında çalışan DC motorlar tercih edilmiştir. Bu motorlar, düşük güçlerle kolayca sürülebilmekte ve kontrol açısından pratik bir kullanım sunmaktadır.

Motorların her iki tarafında bağlantı yapılabilmesi, tercih edilme nedenlerinden biridir. Bir tarafa tekerlek bağlanırken, diğer tarafa enkoder yerleştirilmiştir. Bu sayede, ek bir dişli sistemine ihtiyaç duymadan tekerleğin dönüşü doğrudan ölçülebilmektedir.

Motorlar, bataryadan gelen voltajı düşüren bir regülatör aracılığıyla beslenmektedir. Motorlar doğrudan bataryadan beslendiğinde aşırı ısınma ve gereğinden fazla hız meydana geldiğinden, voltaj düşürülerek daha stabil bir çalışma sağlanmaktadır.

2.2.1.4 Motor Sürücü

Motor sürücü, motorların çalıştırılmasını sağlayan temel bileşendir. Bu projede, L298N model motor sürücü kullanılmıştır. Bu sürücü, aynı anda iki motorun sürülmesine olanak tanımakta ve üzerinde bulunan 5V regülatör sayesinde Arduino kartına ve sensörlere doğrudan enerji sağlamaktadır.

L298N'nin tercih edilme sebepleri; iki motoru ayrı yönlerde ve farklı hızlarda sürebilmesi, böylece aracın hem düz hareketlerinde hem de dönüşlerinde kontrol kolaylığı sağlamasıdır. Sürücü, doğrudan Arduino kartına bağlıdır ve motorlar Arduino üzerinden kontrol edilmektedir.

2.2.2 Sensörler

2.2.2.1 Enkoder Sayıcı Sensörler

Bu sensörler, kızılötesi ışık kullanarak önlerinde bir cisim olup olmadığını algılamaktadır. Artımlı enkoderlerle birlikte kullanıldıklarında, enkoderin kaç adım ilerlediği bu sensörler aracılığıyla ölçülebilir hâle gelmektedir. Böylece, enkoderlerden elde edilen verilerle aracın hareketi kontrol edilebilmektedir.

Söz konusu sensörler, sistemde aracın kontrolünü gerçekleştirmemizi sağlayan temel sensörlerdir. Voltaj beslemesi ve sinyal bağlantıları, delikli plaket üzerinde hazırlanan devre aracılığıyla Arduino kartına bağlanmıştır.

2.2.2.2 Mesafe Sensörü

Bu sensör aracın önünde olup aracın çarpmasını engellemek için bulunmaktadır. Bu sensör sürekli mesafe ölçerek belli bir mesafe altına düşüldüğünde aracın durmasını sağlamaktadır. Sensör ultrasonik sensördür ve ses dalgalarını kullanarak mesafe ölçümü yapmaktadır. Bir yerden ultrasonik bir dalga göndermekte ve diğer taraftan bu sesi dinlemektedir ve gidiş geliş süresini ölçerek mesafeyi hesaplamaktadır. Bu sensörde Arduino kartına bağlanarak veri ölçümü sağlanmaktadır.

2.2.3 Kontrol Kartları

Aracın kontrolü haberleşmesi gibi işlemler kontrol kartları ile sağlanmaktadır araçta iki adet kontrol kartı kullanılmaktadır Arduino ve ESP8266. Sistemde iki ayrı kart kullanmamızın sebebi ise kartlara düşen işlem yoğunluğunu bölüşmek ve her kartın ana görevinin başka olmasıdır. Arduino aracı kontrol ederken ESP8266 kartı aracın internete bağlanmasını ve uzaktan kontrolünü sağlamaktadır. Böylelikle iki yoğun işlemi ikiye bölerek sistemin daha doğru ve rahat şekilde çalışması sağlandı.

2.2.3.1 Arduino

Arduino kartının temel görevi, aracın hareketini ve genel kontrolünü sağlamaktır. Kart üzerinde bulunan gömülü yazılım sayesinde araç yönlendirilmekte, kontrol edilmekte ve sensörlerden veri okunmaktadır. Araç, hareket komutlarını ESP8266 kartından aldığı verilere göre

gerçekleştirmektedir. Arduino ile ESP8266 arasında haberleşme, RX-TX pinleri üzerinden seri iletişim yoluyla sağlanmaktadır.

ESP8266 kartından gelen yön ve mesafe bilgileri Arduino tarafından işlenmekte; bu verilere göre motorlara, motor sürücü aracılığıyla gerekli hareket komutları iletilmektedir. Arduino kartında yer alan yazılım, aracın düz bir şekilde ilerlemesini ve istenilen açılarda dönüş yapmasını mümkün kılmaktadır. Ayrıca, enkoder verilerini işleyerek mesafe hesaplaması yapmakta ve bu veriyi anlık olarak ESP8266 kartına ileterek canlı konum takibini gerçekleştirmektedir.

Arduino kartı delikli plaket üzerine konnektörler yardımı ile direkt oturarak sistemde sabit yere ve doğru bağlantılara sahip olmaktadır. Arduino ayriyeten ESP8266 kartının çalışması için gerekli enerjiyi sağlamaktadır.

2.2.3.2 ESP8266 Kartı

ESP8266, kablosuz ağlara bağlanabilen düşük maliyetli bir Wi-Fi modülüdür.[4] ESP8266 kartı, sistemde haberleşme birimi olarak görev yapmaktadır. Kartın temel işlevi, önceden oluşturduğumuz internet tabanlı arayüz (web sitesi) ile bağlantı kurarak gerekli verileri çekmek ve bu verileri Arduino kartına iletmektir.

ESP8266, Wi-Fi bağlantısı aracılığıyla internete erişir ve kontrol arayüzünden (örneğin yön, hız veya hedef mesafe gibi bilgiler) alınan komutları seri iletişim yoluyla Arduino'ya aktarır. Böylece aracın uzaktan kontrolü mümkün hale gelir. ESP8266, bu görevinde sensör verisi işleme veya doğrudan donanım kontrolü yapmamakta; yalnızca arayüz ile araç arasında köprü görevi üstlenmektedir.

Arduino kartı, ESP8266'dan aldığı bu komutlara göre fiziksel hareketleri ve motor sürücülerini kontrol eder. ESP8266, delikli plaket üzerinde sabitlenmiş ve sistemle uyumlu şekilde bağlantıları yapılmıştır. Güç ihtiyacı Arduino üzerinden sağlanarak ek bir enerji kaynağına gerek kalmadan çalışması sağlanmaktadır.

2.2.3.3 Mantık Kanal Düşürücü

Mantık seviye düşürücü kart, farklı voltaj seviyelerinde çalışan iki cihazın birbiriyle sorunsuz şekilde haberleşmesini sağlar. Biz de bu kartı, Arduino ve ESP8266 arasında haberleşme kurmak için kullandık. Her iki karttan da besleme alan bu devrede, bir tarafa ESP8266'den gelen 3V, diğer tarafa ise Arduino'dan gelen 5V bağlanmıştır. Rx ve Tx pinleri, karşılıklı olarak A ve B pinlerine bağlanarak veri iletişimi sağlanmıştır. Bu kart sayesinde, iki kart arasında oluşabilecek haberleşme hataları engellenmiş ve stabil bir iletişim ortamı oluşturulmuştur. Bu kart kendi yaptığımız delikli plaket devresinde bulunmaktadır. Böyle daha düzenli ve doğru bir biçimde bağlantılarının yapılması sağlanmıştır.

2.3 YAZILIM

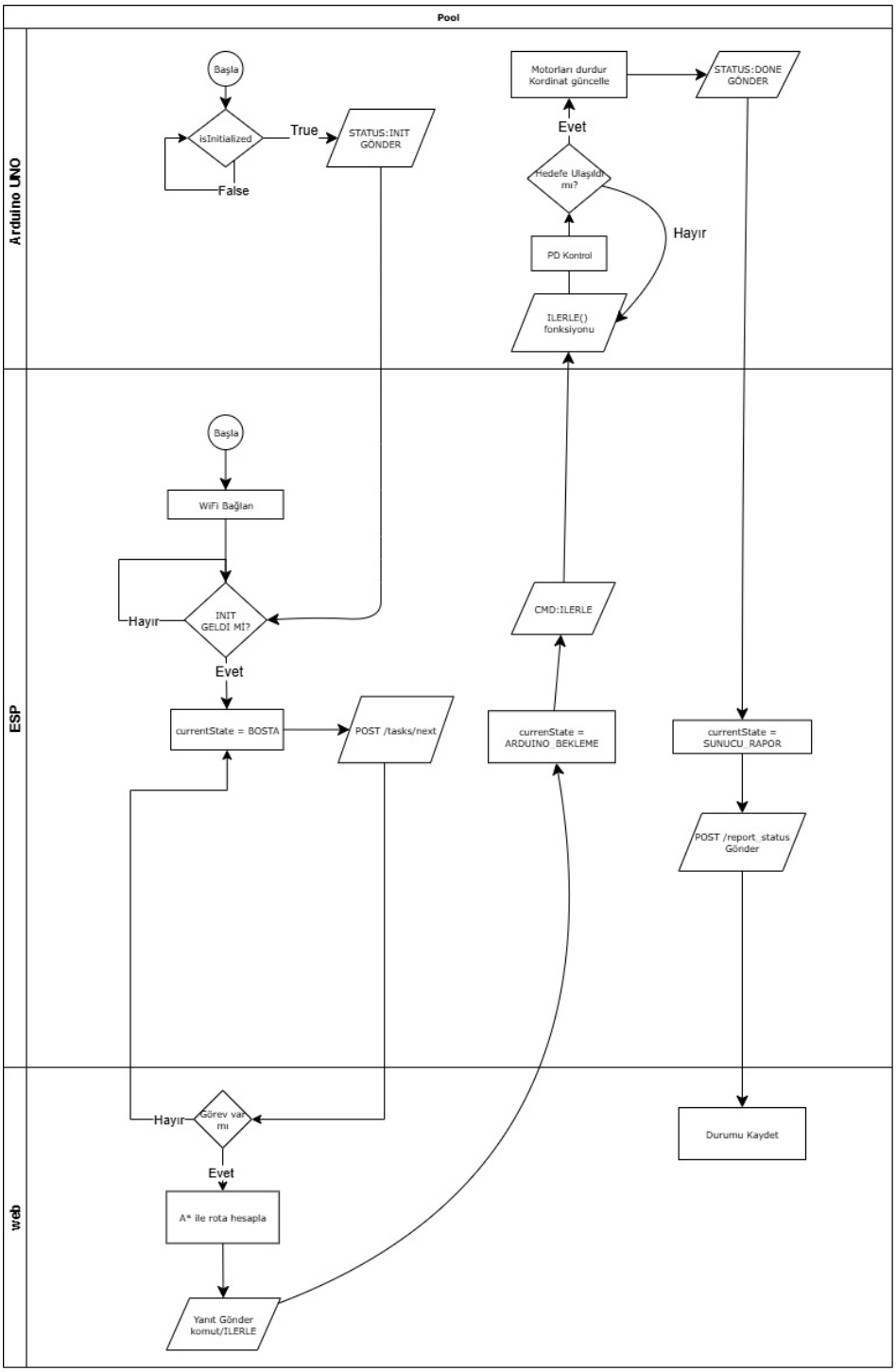
Aracın yazılım altyapısı, sistemin modüler yapısına uygun olarak farklı alanlara ayrılmıştır. Bu ayrım, hem görevlerin daha verimli bir şekilde yerine getirilmesini hem de sistemin daha kolay yönetilebilmesini sağlamaktadır. Araç içerisinde birden fazla mikrodenetleyici kartı bulunduğundan, her bir kartın üstlendiği göreve özel yazılımlar geliştirilmiştir.

Genel olarak yazılım yapısı üç temel başlığa ayrılmaktadır:

- Kontrol Yazılımı: Arduino kartı üzerinde çalışmakta olup, motorlar, kontrol algoritması ve fiziksel donanımların yönetilmesini sağlamaktadır.

- Haberleşme Yazılımı: ESP8266 kartı ile yazılım arayüzü arasında veri aktarımını sağlamakta ve alınan bilgileri Arduino'ya iletmektedir.
- Arayüz ve Web Entegrasyonu: Kullanıcı tarafından oluşturulan web tabanlı kontrol paneli ile aracın uzaktan yönlendirilmesini sağlamaktadır.

Her bir yazılım bileşeni, kendi görev alanında çalışmakta ve bu parçalar bir bütün olarak aracın otonom ya da uzaktan kontrollü şekilde çalışmasını mümkün kılmaktadır. Aşağıda bu bileşenler detaylı şekilde açıklanmıştır.



Şekil 15 Yazılım Algoritması

2.3.1 Arduino UNO

3 katmanlı mimarimizin en alt katmanında yer alan donanım kontrolcüsü olarak görev yapar. Arduino UNO kartı, ESP8266'dan gelen komutları alır, bu komutları fiziksel hareketlere dönüştürür ve robotun durumunu sensörler aracılığıyla takip ederek geri bildirim verir.

Arduino kartının ana sorumlukları:

- ESP8266 ile güvenilir bir iletişim kurmak.
- ESP8266'dan gelen komutları işleyerek gerekli cevapları oluşturmak.
- Her komut sonrası ESP8266'ya bulunduğu konumu bildirmek.
- PD kontrol algoritmasını kullanarak robotun düz bir çizgide hareketini sağlamak.
- Ultrasonik sensör kullanarak engel tespit etmek.

2.3.1.1 Ana Kontrol Akışı ve Haberleşme

Yazılımın kararlı çalışması, ESP8266 ile kurulan güvenilir iletişime bağlıdır. Bu amaçla, basit bir durum yönetimi ve el sıkışma protokolü geliştirilmiştir.

- **El Sıkışma Protokolü**: Arduino ilk çalıştığında, ESP8266'nın WiFi ağına bağlanması gibi işlemler zaman alabilir. Bu senkronizasyon sorununu çözmek için, Arduino isInitialized bayrağı false durumundayken, her iki saniyede bir ESP8266'ya STATUS:INIT mesajı gönderir. Bu mesaj, "Ben hazırım ve komut bekliyorum" anlamına gelir. ESP8266'dan ilk geçerli komut geldiğinde bu bayrak "true" konumuna getirilir ve INIT mesajlarının gönderimi durdurulur. Bu mekanizma, sistemin her açılışta başlatır.

```
if (!isInitialized) {
  if (millis() - lastInitSendTime > 2000) {
    Serial.println("Handshake: Sending INIT to ESP...");
    send_status_report(STATUS_INIT);
    lastInitSendTime = millis();
  }
}
```

Şekil 16 ESP Kartından Haber Bekleme Yazılımı

- **Komut İşleme Döngüsü**: Ana loop() fonksiyonu, sürekli olarak "readFromEsp()" fonksiyonu aracılığıyla seri portu dinler. Gelen veriler bir buffer'da biriktirilir ve yeni satır karakteri ile karşılaşıldığında tam bir komutun alındığı kabul edilir. Bu komutHazır bayrağını tetikler ve "processCommand()" fonksiyonu çağrılarak komutun işlenmesi sağlanır. İşlenen komut gelen komuta göre belli fonksiyonlara götürerek aracın hareketi sağlanmaktadır.

```
void readFromEsp() {
  while (ESP_SERIAL.available() > 0) {
    char c = ESP_SERIAL.read();
    if (c == '\n') { commandReady = true; break; }
    else { espSerialBuffer += c; }
  }
}
```

Şekil 17 ESP Kartından Gelen Veriyi Okuyan Fonksiyon

```
void processCommand(String command) {
  isInitialized = true;
  command.trim();
  Serial.print("Received from ESP: "); Serial.println(command);

  char cmd_type[20];
  int value;
  int parsed = sscanf(command.c_str(), "CMD:%[^(](%d)", cmd_type, &value);

  if (parsed >= 1) {
    String cmdStr = String(cmd_type);
    const char* resultStatus;

    if (cmdStr.equalsIgnoreCase("FORWARD")) {
      resultStatus = go_straight(value > 0 ? value : 1);
    } else if (cmdStr.equalsIgnoreCase("TURN_LEFT")) {
      resultStatus = turn_left();
    } else if (cmdStr.equalsIgnoreCase("TURN_RIGHT")) {
      resultStatus = turn_right();
    } else {
      Serial.print("Unknown command type: "); Serial.println(cmdStr);
      isInitialized = false; // Re-enable handshake if command is bad
      return;
    }

    send_status_report(resultStatus);
  } else {
    Serial.print("Failed to parse command: "); Serial.println(command);
    isInitialized = false; // Re-enable handshake if command is bad
  }
}
```

Şekil 18 Gelen Komutu İşleyen Fonksiyon

2.3.1.2 Enkoder Tabanlı Hareket ve PD Kontrol Algoritması

Bu projede, motorlar PD (Oransal-Türev) kontrol yöntemi ile sürülmüştür.[5] ESP kartından gelen komutlar işlendikten sonra komutlara göre belli hareket fonksiyonları çalışmaktadır. Bu hareketler ileri, sağa dönüş ve sola dönüş olarak sıralanmıştır. Araç bu hareketleri gerçekleştirirken belirli algoritma ve yöntemler kullanılmaktadır.

Robot hareket ederken ilerlediği mesafe yaptığı dönüşlerin derecelerini motorlara sabit şekilde bulunan enkoderler sayesinde hesaplamaktadır. Robotun hassas ve tekrarlanabilir hareketler yapabilmesi için enkoder geri beslemesine dayalı kapalı döngülü bir kontrol sistemi tercih edilmiştir.

Enkoder Verilerinin Kesmeler ile Okunması: Tekerlek dönüşlerini sayan enkoderler, Arduino'nun 2. ve 3. pinlerine bağlıdır. Bu pinler kesme yapısına sahiptir. "attachInterrupt()" fonksiyonu kullanılarak, tekerleğin her bir adımı ana program akışını durdurmaksızın, arka

planda çalışan “leftISR()” ve “rightISR()” fonksiyonları tarafından anında sayılır. Bu yöntem, “loop()” döngüsünün gecikmelerinden etkilenmeyen, kayıpsız ve hassas bir mesafe ölçümü sağlar. Böylelikle araç hareket halindeyken ana döngüde herhangi bir bozulmadan hesaplamalar yapılmaktır.

“attachInterrupt()” fonksiyonu kullanılırken kesme yapısındaki girişinde tetiklemeyi “CHANGE” komutu ile tetiklemesini gerçekleştirdik. Böylelikle enkoderden daha hassas okuma gerçekleştirmiş olduk. ”RISING” veya “FALLING” komutları kullanılsaydı enkoderin her iki adımında bir okuma gerçekleşecekti ve hassasiyet azalacaktı.

Düz Çizgide Hareket için PD Kontrol: “go_straight()” fonksiyonu, robotun düz ilerlemesini gerçekleştirmesini gerçekleştiren fonksiyondur. Araç bu hareketi gerçekleştirirken düz bir çizgide kalmasını sağlamak için bir PD kontrolörü uygular.

- P (Oransal) Terimi: Her döngüde sol ve sağ enkoder sayaçları arasındaki fark hesaplanır. Bu fark bizim hata değerimizi vermektedir. Bu hata, Kp sabiti ile çarpılarak anlık bir düzeltme değeri oluşturur. Böylelikle hata oranı azaltılması hedeflenmektedir.
- D (Türevsel) Terimi: Hatanın zaman içindeki değişim hızı hesaplanır ve Kd sabiti ile çarpılır. Bu terim, robotun anlık hataya aşırı tepki vererek salınım yapmasını engeller ve harekete bir sönümleme etkisi katarak daha kararlı bir sürüş sağlar. Bu iki terimin toplamı olan correction değeri, motorlara gönderilen PWM sinyallerini anlık olarak ayarlayarak robotun belirlenen rotada kalmasını sağlar. Türevsel terim hesabı ise sağ motorun bir önceki sağ motor değerinden çıkarılarak zamandaki değişime bölümü elde edilir.

Bu bulunan oransal ve türevsel değerler katsayılar ile çarpılıp bir düzeltme değerine eşitlenmektedir. Bu düzeltme değeri sağ ve sol motorunun hızına etki ederek aracın düz bir şekilde ilerlenmesi sağlanmıştır.

```
int cur_mil = millis();
int error = (left_s_counter - right_s_counter) / 2;
int derivative = (right_s_counter - previous_count_R) / (cur_mil  - prev_mil);
int correction = Kp * error + Kd * derivative;
prev_mil = cur_mil;

int pwm_L = base_pwm - correction;
int pwm_R = (base_pwm + correction);
pwm_L = constrain(pwm_L, 0, 255);
pwm_R = constrain(pwm_R, 0, 255);
analogWrite(pwm_left, constrain(pwm_L, 0, 255));
analogWrite(pwm_right, constrain(pwm_R, 0, 255));
```

Şekil 19 PD Kontrol Kodu

Robot düz ilerleme yaparken kat ettiği mesafe hesaplanmalıdır. Bu yüzden enkoderlerin bir adımının kaç mm ilerlemeye karşılık geldiğinin hesabı yapılmalıdır. Deneme yanılma yoluyla bulunan değerle birlikte hesap yapılıp araç belli mesafede hareketi sağlanmıştır.

```
  float current_dist_mm = ((left_counter + right_counter) / 2.0) * ENCODER_TICK_TO_MM;
  if (current_dist_mm >= target_dist_mm) {
    break; // Exit the loop when distance is reached
  }
  delay(10); // Small delay to prevent starving other processes
}
```

Şekil 20 Robot Mesafe Kodu

Robot hareket sisteminde dönüş fonksiyonları da bulunmaktadır. Bu dönüş fonksiyonları robotun sağa ve sola dönüşünü sağlamaktadır. İki yöne 90 derece dönüş sağlamaktadır. Bu dönüşte enkoder kullanarak dönüş mesafesi hesaplanmıştır. Araç yapımızda iki motorda aynı hizada önde olduğu için iki motoru aynı anda ters yöne çevirdiğimizde kendi etrafında dönüş gerçekleştirmektedir. Eğer iki teker arası mesafe bilinirse dönüş yapacağı çemberin çapı bilinir. Bu çap bilindiğinde dönüş çemberinin çevresi hesaplanmaktadır. Eğer her iki tekerlekte bu çemberin çeyreği kadar mesafe dönüş sağlarsa 90 derece dönüş sağlanmaktadır. Bu hesaplama ile aracın doğru açıda dönmesi sağlanmıştır.

```
const char* turn_right() {
  reset_encoders();
  // Left motor forward, right motor backward
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW);
  digitalWrite(in3, LOW); digitalWrite(in4, HIGH);

  analogWrite(pwm_left, base_pwm);
  analogWrite(pwm_right, base_pwm);

  while( (left_counter * ENCODER_TICK_TO_MM < DEGREES_TO_MM_90_TURN) &&
         (right_counter * ENCODER_TICK_TO_MM < DEGREES_TO_MM_90_TURN) ) {
    delay(1);
  }
  stop_motors();
  robotState.theta = (robotState.theta + 1) % 4;
  return STATUS_DONE;
}
```

Şekil 21 Dönüş Fonksiyonu

2.3.1.3 Engel Tanıma

Robot hareket ederken önüne belirlenemeyen bir engel çıkabilmektedir. Bu engeli algılayıp çarpmasını engellemek için aracın önünde bulunan ultrasonik mesafe sensörü ile sürekli br mesafe ölçümü yapılmaktadır. Önüne bir engel çıktığında sistemi uyarmakta ve robotu durdurmaktadır ve yapacağı işlemleri bekletmektedir. Böylelikle robotun herhangi bir yere çarpıp zarar vermesini veya zarar görmesini engellemiş olduk.

```
bool check_obstacle() {
  digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH, 25000); // 25ms timeout
  float distance_cm = (duration / 2.0) * 0.0343;
  return (distance_cm > 0 && distance_cm < OBSTACLE_DISTANCE_CM);
}
```

Şekil 22 Engel Tanıma Kodu

2.3.1.4 Durum Raporlama ve Koordinat Takibi

Her hareket komutu başarıyla tamamlandıktan veya bir engelle karşılaşıldıktan sonra, Arduino anında ESP8266'ya bir durum raporu gönderir.

Bu rapor, STATUS:DURUM:X:yeni_x:Y:yeni_y:T:yeni_yon formatındadır. Arduino, kendi içinde tuttuğu robotDurum yapısındaki X, Y koordinatlarını ve yön (Theta) bilgisini, yaptığı harekete göre günceller ve bu yeni durumu rapora ekler. Bu kapalı döngü raporlama, sistemin

WEB katmanının her zaman robotun haritadaki konumu hakkında doğru bilgiye sahip olmasını sağlar.

```
void send_status_report(const char* status) {
  String report = "STATUS:" + String(status) +
                  ":X:" + String(robotState.x) + ":Y:" + String(robotState.y) +
                  ":T:" + String(robotState.theta) + "\n";
  Serial.print("Sending to ESP: "); Serial.print(report);
  ESP_SERIAL.print(report);
}
```

Şekil 23 Durum Raporlama Kodu

2.3.2 ESP8266

3 katmanlı yazılım mimarimizin 2. katmanıdır. Operasyon yöneticisi olarak çalışır. Web sunucumuzla arduino üzerinde çalışan kod arasında çalışır. Kodumuzun temel amacı Web'den gelen verileri alıp arduinoya'ya ileterek hareket etmesine yardımcı olur.

ESP kartının ana sorumlukları:

- HTTP protokolünü kullanarak sunucumuzla JSON formatında veri alışverişi yapmak.
- Arduino UNO ile güvenilir bir seri iletişim kurmak.
- Durum Makinası mimarisini yönetmek

2.3.2.1 Durum Makinesi Mimarisi

ESP8266 yazılımının en önemli bileşeni, RobotDurum ile tanımlanan olay tabanlı durum makinesidir. Bu mimari, ağ gecikmeleri, fiziksel hareketlerin zaman alması gibi asenkron olayları yönetmek için idealdir. Robot, her an yalnızca bir durumda bulunur ve durumlar arası geçişler, sunucudan veya Arduino'dan gelen bir mesaj ya da bir zaman aşımı gibi belirli bir tarafından tetiklenir.

Ana Çalışma Durumları ve Akışı:

- **DURUM_ACILIS**: Sistem ilk başladığında, Arduino'dan STATUS:INIT mesajının gelmesini bekler. Bu el sıkışma süreci tamamlanana kadar başka hiçbir işlem yapılmaz. Bu, sistemin sadece tüm bileşenler hazır olduğunda harekete başlamasını sağlar.

- **DURUM_BOSTA**: El sıkışma tamamlandıktan sonra girilen varsayılan durumdur. Robot bu durumdayken, periyodik olarak sunucunun /tasks/next API'sini sorgulayarak yeni bir görev olup olmadığını kontrol eder.

- **DURUM_GOREV_BASLATMA**: Sunucudan geçerli bir görev atandığında bu duruma geçilir. Robot, görevi başlatmak için mevcut konumunu /report_status API'si üzerinden sunucuya raporlar

- **DURUM_KOMUT_ALMA**: Sunucudan, hesaplanan rota üzerindeki bir sonraki hareket komutunu almak için /next_command API'sine istek gönderir.

- **DURUM_ARDUINO_YA_GONDERME**: Alınan komutu Arduino'ya seri port üzerinden iletir.

- **DURUM_ARDUINO_BEKLEME**: Komut gönderildikten sonra girilen bekleme durumudur. Arduino'nun hareketi tamamlayıp bir sonuç raporlaması beklenir.

- **DURUM_SUNUCU_RAPOR**: Arduino'dan gelen sonuç, sunucuya raporlanır. Bu adımdan sonra sistem tekrar DURUM_KOMUT_ALMA'ya dönerek bir sonraki hareketi ister.

- **DURUM_GOREV_SONLANDIRMA**: Sunucu KomutTamamla mesajı gönderdiğinde girilir ve robotu tekrar DURUM_BOSTA moduna alır.

2.3.2.2 Haberleşme ve Veri İşleme

- **Sunucu ile İletişim**: Tüm sunucu iletişimi, httpIstekYap() fonksiyonu içinde HTTP GET ve POST istekleri ile gerçekleştirilir. Sunucudan gelen ve sunucuya gönderilen veriler, JSON formatındadır. Gelen JSON verileri, ArduinoJson kütüphanesi kullanılarak ayrıştırılır ve işlenir.

- **Arduino ile İletişim**: ESP8266 ile Arduino arasındaki veri akışını sağlamak amacıyla SoftwareSerial kütüphanesi kullanılmıştır. Haberleşme için, belirlediğimiz metin tabanlı bir protokol kullanıldı. Bu protokole göre, ESP tarafından gönderilen tüm komutlar CMD: önekini, Arduino'dan gelen durum ve koordinat raporları ise STATUS: önekini taşımaktadır. Tüm mesajlar yeni satır karakteri ile sonlandırılarak mesajların başlangıç ve bitiş noktaları net bir şekilde belirlenmiştir. arduinoYanitiniOkuVeAyristir() fonksiyonu, Arduino'dan gelen bu mesajları ayrıştırarak içindeki durumu ve koordinat verilerini alır ve durum makinesinin bir sonraki adıma geçmesini tetikler

2.3.3 WEB Kodu

Üç katmanlı sistem mimarimizin en üst katmanında, projenin beyni olarak görev yapan Go Sunucusu yer almaktadır. Bu katman, fiziksel robotun operasyonel detaylarından tamamen soyutlanmıştır. Temel görevi, "nereye gidileceğini" ve "hangi görevin yapılacağını" planlamak ve bu kararları yönetmektir.

Sunucu, çift katmanlı bir yapıya sahiptir:

- Arayüz Katmanı (Ön Yüz - Frontend): Kullanıcı ile sistem arasındaki etkileşim noktasıdır.
- API ve Mantık Katmanı (Arka Yüz - Backend): Sistemin ESP8266 bileşeni ile haberleşen ve tüm iş mantığını yürüten makine odaklı kısımdır.

2.3.3.1 Arayüz Katmanı

Bu katman, kullanıcının sistemi yönetmesini ve izlemesini sağlayan görsel bir web arayüzüdür. Temel yetenekleri şunlardır:

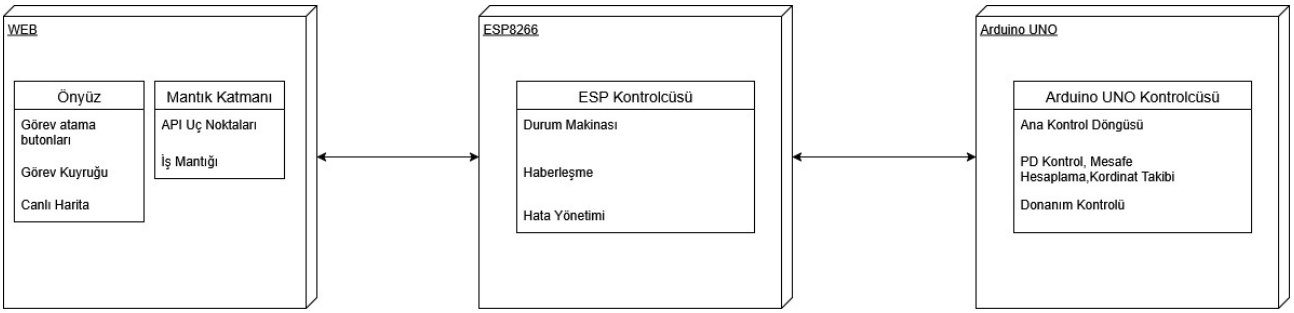
- Görev Atama: Kullanıcının, robotun gitmesi gereken hedef istasyonları seçerek sisteme yeni görevler eklemesine olanak tanır.
- Gerçek Zamanlı Harita Takibi: Robotun anlık konumu (X, Y koordinatları) ve yönü (Theta), sunucuya gelen raporlar doğrultusunda güncellenerek statik bir harita üzerinde görselleştirilir. Bu, robotun operasyonel durumunun canlı olarak izlenmesini sağlar.
- Görev Kuyruğu Görüntüleme: Veri tabanında bekleyen, atanmış ve tamamlanmış tüm görevlerin listesini sunarak sistemin yoğunluğu ve geçmiş operasyonlar hakkında bilgi verir.

2.3.3.2 API ve Mantık Katmanı

Bu katman, sistemin görünmeyen ancak en kritik işlemlerini gerçekleştirir. ESP8266 mikrodenetleyicisi ile standart bir RESTful API üzerinden haberleşir.

- Görev Kuyruklama: Ön yüzden gelen yeni görev taleplerini alır ve bunları bir SQLite veri tabanına beklemede durumuyla kaydeder. Bu sayede görevler kalıcı hale gelir ve sunucu yeniden başlatılsa bile kaybolmaz.
- Stratejik Rota Planlama: ESP8266'dan /tasks/next API'sine bir istek geldiğinde, veri tabanından en eski pending görevi alır. Ardından, önceden yüklenmiş olan harita verisi üzerinde, robotun mevcut konumu ile hedefin konumu arasında en verimli rotayı bulmak için A* (A-Star) algoritmasını çalıştırır. A* algoritması, hedefe giden en kısa yolu bulmak için kullanılan yaygın bir arama algoritmasıdır.[6] Bu algoritma, engelleri dikkate alarak en kısa ve en güvenli yolu hesaplar.
- Komut Üretimi: A* algoritmasının ürettiği koordinat listesinden oluşan rotayı, ESP8266 ve Arduino'nun anlayabileceği basit, sıralı komutlara dönüştürür.
- Durum Raporlarını İşleme: ESP8266'dan gelen durum raporlarını işleyerek robotun güncel konumunu hafızada tutar. Eğer bir STATUS:OBSTACLE raporu gelirse, mevcut rotayı iptal eder ve A* algoritmasını tekrar çalıştırarak yeni bir rota hesaplar.

Şekil 24 WEB Arayüzü



Şekil 25 Yazılım Şablonu

3. BULGULAR VE TARTIŞMA

Bu çalışmada geliştirilen mobil robot sistemi, ESP8266 ile Arduino UNO arasında kurulan yazılımsal seri haberleşme sayesinde temel hareket komutlarını başarıyla almış ve uygulamıştır. Yapılan testlerde "FORWARD", "TURN_LEFT" ve "TURN_RIGHT" komutları ile robot sabit ızgara üzerinde yönlenerek hedef konumlara doğru ilerleyebilmiştir.

PD kontrol algoritması, düz ilerleme sırasında tekerlekler arası hız farklarını dengelemekte yeterli derecede başarılı olmuştur. PD kontrol uygulanırken araçtaki salınım hareket en az noktaya indirilse dahi çalışmayı bozmadan ufak salınımlar devam etmektedir. PD kontrolde bizim gibi direkt referans değeri olmadan karşılaştırma yaparak işlemlerin yapılması zorluğu arttırmaktadır. Çünkü araçta birçok değişken bulunmaktadır. Ve bu değişkenler anlık olarak farklı sonuçlar çıkarmaktadır.

Dönüş hareketlerinde ±10° sapma ile 90° dönüşlerin tamamlandığı görülmüş; bu sapmanın, yüzey tutuşu, tekerlek boyutu ve motor tepkisinden kaynaklandığı düşünülmektedir. Yazılım tarafında ise komut işleme ve geri bildirim mekanizması (STATUS mesajları) düzgün şekilde çalışmış, bağlantı kopması gibi bir hata yaşanmamıştır.

Bu bulgular, sistemin temel seviye otonom hareket görevlerini başarıyla gerçekleştirdiğini, ancak bazı yönlerden geliştirilmeye açık olduğunu göstermektedir.

4. SONUÇLAR

Bu proje kapsamında, ESP8266 ve Arduino UNO arasında yazılım tabanlı seri iletişim sağlanarak çevresel faktörlere tepki verebilen, temel komutlara yanıt verebilen ve belirli görevleri gerçekleştirebilen bir mobil robot sistemi başarıyla geliştirilmiştir. Sistem, kablosuz olarak aldığı komutları işleyip, bu komutlara uygun motor hareketlerini gerçekleştirerek bir ızgara sisteminde konum takibi yapabilmektedir.

Hareketin doğruluğunu sağlamak ve tekerlekler arası senkron kaymaların önüne geçmek amacıyla, sisteme bir PD (Oransal-Türev) kontrol algoritması entegre edilmiştir. Bu kontrol yapısı, robotun hedeflenen mesafe boyunca doğrusal ve dengeli bir şekilde ilerlemesini sağlarken, sistem karmaşıklığını artırmadan yeterli hassasiyet sağlamıştır. Projede özellikle basitlik ve gerçeklenebilirlik göz önüne alınarak PID yerine PD kontrol tercih edilmiştir; bu da robota yeterli tepki süresi ve stabil hareket performansı kazandırmıştır.

Yapılan testlerde, robotun ileri hareket, sağa-sola dönüş ve engel algılama gibi temel görevleri başarıyla yerine getirdiği gözlemlenmiştir. Ultrasonik sensör ile engel algılama işlemi sayesinde, potansiyel çarpışmalar önceden tespit edilmiş ve robot güvenli şekilde durdurulmuştur. Komutların doğru şekilde alınması ve karşılıkların (durum mesajları) ESP8266'ya iletilmesi ile çift yönlü haberleşmenin kararlı çalıştığı doğrulanmıştır.

Sistemin modüler mimarisi, ileride yapılabilecek güncellemeler ve genişletmeler için uygun bir altyapı sunmaktadır. Gelecek çalışmalarda sisteme kamera modülleri, yönsel görüntü işleme, gelişmiş sensör sistemleri ya da tam PID kontrol algoritması eklenerek hem kontrol hassasiyeti artırılabilir hem de robotun otonom davranışları geliştirilebilir. Ayrıca Wi-Fi üzerinden canlı komut gönderimi ve konum takibi gibi üst düzey özelliklerle sistemin uzaktan erişilebilirliği de sağlanabilir.

KAYNAKÇA

[1] Siciliano, B., Khatib, O. (2016). Springer Handbook of Robotics. Springer.

[2] Siegwart, R., Nourbakhsh, I. R., Scaramuzza, D. (2011). Introduction to Autonomous Mobile Robots. MIT Press.

[3] Hambley, A. R. (2013). Electrical engineering: Principles and applications (6th ed.). Pearson

[4] Espressif Systems, "ESP8266 Technical Reference Manual", 2023.

[5] Dorf, R. C., Bishop, R. H. (2016). Modern Control Systems. Pearson.

[6] Hart, P. E., Nilsson, N. J., Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics.

EKLER

Ek 1 – Arduino kartı üzerinde çalışan yazılım

```
#include <SoftwareSerial.h>


const byte ESP_RX_PIN = 4;
const byte ESP_TX_PIN = 7;
SoftwareSerial espSerial(ESP_RX_PIN, ESP_TX_PIN);


#define ESP_SERIAL espSerial
const unsigned long ESP_BAUD_RATE = 9600;
String espSerialBuffer = "";
bool commandReady = false;
bool isInitialized = false;
unsigned long lastInitSendTime = 0;


struct RobotState {
  int x;
  int y;
  int theta; // 0:North, 1:East, 2:South, 3:West
};
RobotState robotState = {0, 0, 0};


const float GRID_CELL_DISTANCE_MM = 300.0;
const float DEGREES_TO_MM_90_TURN = 262.5;
const float ENCODER_TICK_TO_MM = 6.2;


const int encoder_left = 2;
const int encoder_right = 3;
const int in1 = 8;
const int in2 = 6;
const int in3 = 10;
const int in4 = 9;
```

```
const int pwm_left = 5;
const int pwm_right = 11;
#define TRIG_PIN 12
#define ECHO_PIN 13
const float OBSTACLE_DISTANCE_CM = 15.0;


volatile long left_counter = 0;
volatile long right_counter = 0;
float Kp = 8.82;
float Kd = 0.5;
int base_pwm = 150;

long  left_s_counter = 0;
long  right_s_counter = 0;



const char* STATUS_INIT = "INIT";
const char* STATUS_DONE = "DONE";
const char* STATUS_OBSTACLE = "OBSTACLE";



void setup() {
  Serial.begin(115200);
  while (!Serial);
  Serial.println("\n\n-- Integrated Arduino UNO Robot Controller --");


  ESP_SERIAL.begin(ESP_BAUD_RATE);
  Serial.print("Listening to ESP8266 on SoftwareSerial at ");
  Serial.print(ESP_BAUD_RATE);
  Serial.println(" baud.");
```

```

  // Pin Configurations
  pinMode(encoder_left, INPUT_PULLUP);
  pinMode(encoder_right, INPUT_PULLUP);
  pinMode(in1, OUTPUT); pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT); pinMode(in4, OUTPUT);
  pinMode(pwm_left, OUTPUT); pinMode(pwm_right, OUTPUT);
  pinMode(TRIG_PIN, OUTPUT); pinMode(ECHO_PIN, INPUT);

  // Attach Interrupts (Pins 2 and 3 are the only interrupt pins on the Uno)
  attachInterrupt(digitalPinToInterrupt(encoder_left), leftISR, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoder_right), rightISR, CHANGE);
}


void loop() {
  readFromEsp();

  if (!isInitialized) {
    if (millis() - lastInitSendTime > 2000) {
      Serial.println("Handshake: Sending INIT to ESP...");
      send_status_report(STATUS_INIT);
      lastInitSendTime = millis();
    }
  }

  if (commandReady) {
    commandReady = false;
    processCommand(espSerialBuffer);
    espSerialBuffer = "";
  }
}

```

```
void readFromEsp() {
  while (ESP_SERIAL.available() > 0) {
    char c = ESP_SERIAL.read();
    if (c == '\n') { commandReady = true; break; }
    else { espSerialBuffer += c; }
  }
}

void processCommand(String command) {
  isInitialized = true;
  command.trim();
  Serial.print("Received from ESP: "); Serial.println(command);

  char cmd_type[20];
  int value;
  int parsed = sscanf(command.c_str(), "CMD:%[^(](%d)", cmd_type, &value);

  if (parsed >= 1) {
    String cmdStr = String(cmd_type);
    const char* resultStatus;

    if (cmdStr.equalsIgnoreCase("FORWARD")) {
      resultStatus = go_straight(value > 0 ? value : 1);
    } else if (cmdStr.equalsIgnoreCase("TURN_LEFT")) {
      resultStatus = turn_left();
    } else if (cmdStr.equalsIgnoreCase("TURN_RIGHT")) {
      resultStatus = turn_right();
    } else {
      Serial.print("Unknown command type: "); Serial.println(cmdStr);
      isInitialized = false; // Re-enable handshake if command is bad
      return;
    }

```

```
    send_status_report(resultStatus);
  } else {
    Serial.print("Failed to parse command: "); Serial.println(command);
    isInitialized = false; // Re-enable handshake if command is bad
  }
}

void send_status_report(const char* status) {
  String report = "STATUS:" + String(status) +
                  ":X:" + String(robotState.x) + ":Y:" + String(robotState.y) +
                  ":T:" + String(robotState.theta) + "\n";
  Serial.print("Sending to ESP: "); Serial.print(report);
  ESP_SERIAL.print(report);
}



const char* go_straight(int grid_cells) {
  float target_dist_mm = grid_cells * GRID_CELL_DISTANCE_MM;
  reset_encoders();

  // Set motor direction to FORWARD
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW);
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW);

  long previous_count_L = 0;
  long previous_count_R = 0;

  unsigned long prev_mil = 0;
  unsigned long cur_mil = 0;

  while (true) {
    if (check_obstacle()) {
      stop_motors();
      Serial.println("!!! OBSTACLE DETECTED !!!");
      return STATUS_OBSTACLE;
    }

    int cur_mil = millis();
    int error = (left_s_counter - right_s_counter) / 2;
    int derivative = (right_s_counter - previous_count_R) / (cur_mil  - prev_mil);
    int correction = Kp * error + Kd * derivative;
    prev_mil = cur_mil;

    int pwm_L = base_pwm;
    int pwm_R = (base_pwm + correction);
    pwm_L = constrain(pwm_L, 0, 255);
    pwm_R = constrain(pwm_R, 0, 255);
    analogWrite(pwm_left, constrain(pwm_L, 0, 255));
    analogWrite(pwm_right, constrain(pwm_R, 0, 255));


    previous_count_L = left_s_counter;
    previous_count_R = right_s_counter;

    float current_dist_mm = ((left_counter + right_counter) / 2.0) * ENCODER_TICK_TO_MM;
    if (current_dist_mm >= target_dist_mm) {
      break; // Exit the loop when distance is reached
    }
    delay(10); // Small delay to prevent starving other processes
  }

  stop_motors();

  switch(robotState.theta) {
    case 0: robotState.y += grid_cells; break; // North
    case 1: robotState.x += grid_cells; break; // East
    case 2: robotState.y -= grid_cells; break; // South
    case 3: robotState.x -= grid_cells; break; // West
```

```
  }
  return STATUS_DONE;
}

const char* turn_right() {
  reset_encoders();
  // Left motor forward, right motor backward
  digitalWrite(in1, HIGH); digitalWrite(in2, LOW);
  digitalWrite(in3, LOW); digitalWrite(in4, HIGH);

  analogWrite(pwm_left, base_pwm);
  analogWrite(pwm_right, base_pwm);

  while( (left_counter * ENCODER_TICK_TO_MM < DEGREES_TO_MM_90_TURN) &&
         (right_counter * ENCODER_TICK_TO_MM < DEGREES_TO_MM_90_TURN) ) {
    delay(1);
  }
  stop_motors();
  robotState.theta = (robotState.theta + 1) % 4;
  return STATUS_DONE;
}

const char* turn_left() {
  reset_encoders();
  // Left motor backward, right motor forward
  digitalWrite(in1, LOW); digitalWrite(in2, HIGH);
  digitalWrite(in3, HIGH); digitalWrite(in4, LOW);

  analogWrite(pwm_left, base_pwm);
  analogWrite(pwm_right, base_pwm);

  while( (left_counter * ENCODER_TICK_TO_MM < DEGREES_TO_MM_90_TURN) &&
         (right_counter * ENCODER_TICK_TO_MM < DEGREES_TO_MM_90_TURN) ) {
    delay(1);
```

```
void leftISR() {
  left_counter++;
  left_s_counter++;
  }
void rightISR() {
  right_counter++;
  right_s_counter++;
  }

void reset_encoders() {
  stop_motors();
  delay(50);
  noInterrupts();
  left_counter = 0;
  right_counter = 0;
  interrupts();
}

void stop_motors() {
  digitalWrite(pwm_left, LOW);
  digitalWrite(pwm_right, LOW);
  digitalWrite(in1, LOW); digitalWrite(in2, LOW);
  digitalWrite(in3, LOW); digitalWrite(in4, LOW);
}

bool check_obstacle() {
  digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH, 25000); // 25ms timeout
  float distance_cm = (duration / 2.0) * 0.0343;
  return (distance_cm > 0 && distance_cm < OBSTACLE_DISTANCE_CM);
}
```

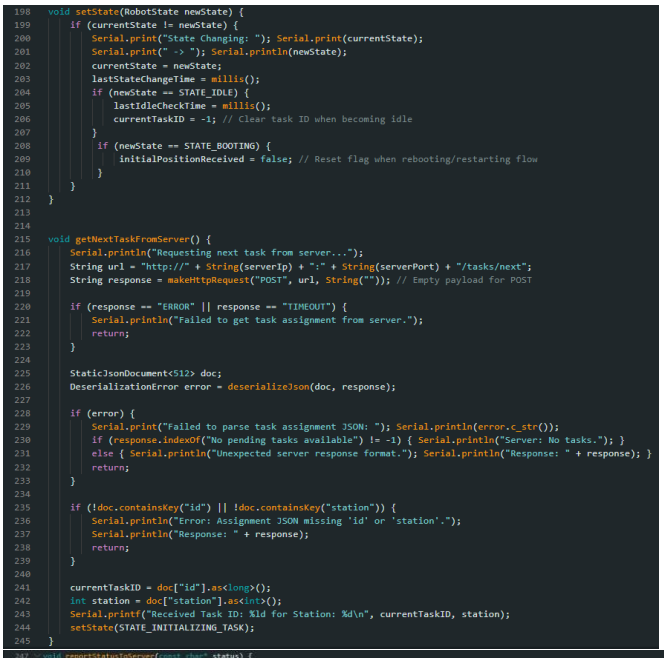
Ek 2 – ESP kartı üzerinde çalışan kod

```ino
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
#include <ArduinoJson.h> // Use v6+

const char* ssid = "*****";
const char* password = "****";


const char* serverIp = "192.168.170.239";
const uint16_t serverPort = 8080;

const unsigned long WIFI_CONNECT_TIMEOUT = 30000;
const unsigned long HTTP_TIMEOUT_MS = 10000;
const unsigned long IDLE_CHECK_INTERVAL_MS = 5000;
const unsigned long ARDUINO_RESPONSE_TIMEOUT_MS = 7000;
const unsigned long GET_CMD_RETRY_DELAY_MS = 2000;
const unsigned long REPORT_RETRY_DELAY_MS = 3000;

#include <SoftwareSerial.h>
const byte ARDUINO_RX_PIN = D5;
const byte ARDUINO_TX_PIN = D6;
SoftwareSerial ArduinoSerial(ARDUINO_RX_PIN, ARDUINO_TX_PIN);
#define ARDUINO_SERIAL ArduinoSerial

const unsigned long ARDUINO_BAUD_RATE = 9600;

const char* CmdForward     = "FORWARD(1)";
const char* CmdTurnLeft    = "TURN_LEFT(90)";
const char* CmdTurnRight   = "TURN_RIGHT(90)";
const char* CmdNone        = "NO_COMMAND";
const char* CmdComplete    = "PATH_COMPLETE";
const char* CmdError       = "ERROR";

const char* StatusInit     = "INIT";
const char* StatusDone     = "DONE";
const char* StatusObstacle = "OBSTACLE";
const char* StatusTimeout  = "TIMEOUT";

const int OrientationNorth = 0;
const int OrientationEast  = 1;
const int OrientationSouth = 2;
const int OrientationWest  = 3;
```

```ino
enum RobotState {
    STATE_BOOTING,             // Initial state, waiting for Arduino INIT_POS
    STATE_IDLE,                // Connected, waiting for task assignment
    STATE_INITIALIZING_TASK,   // Task assigned, reporting initial pos to server
    STATE_GETTING_COMMAND,     // Ready for next navigation command from server
    STATE_SENDING_TO_ARDUINO,  // Sending received command to Arduino
    STATE_WAITING_FOR_ARDUINO, // Waiting for DONE/OBSTACLE status from Arduino
    STATE_REPORTING_TO_SERVER, // Reporting Arduino status (DONE/OBSTACLE) + pos to server
    STATE_TASK_FINALIZING,     // Calling /complete or /fail endpoint (Optional, server handles internally)
    STATE_ERROR                // Unrecoverable error state
};

RobotState currentState = STATE_BOOTING;
long currentTaskID = -1;
String lastServerCommand = "";
String lastArduinoStatus = "";

int lastReportedX = 0;
int lastReportedY = 0;
int lastReportedTheta = 0;
bool initialPositionReceived = false;

unsigned long lastStateChangeTime = 0;
unsigned long lastIdleCheckTime = 0;
unsigned long arduinoCmdSentTime = 0;

String arduinoSerialBuffer = "";

void connectWiFi();
void handleStateMachine();
void setState(RobotState newState);
String makeHttpRequest(String method, String url, String payload);
bool sendCommandToArduino(String command);
bool readAndParseArduinoResponse();
void reportStatusToServer(const char* status);
void getNextCommandFromServer();
void getNextTaskFromServer();

void setup() {
    Serial.begin(115200);
    while (!Serial && millis() < 2000) { ; }
    Serial.println("\n\nESP8266 Robot Coordinator Booting...");

    ARDUINO_SERIAL.begin(ARDUINO_BAUD_RATE);
    Serial.println("Arduino Serial (using SoftwareSerial on D5/D6) initialized.");
    Serial.print("Baud rate set to: ");
    Serial.println(ARDUINO_BAUD_RATE);
```

```ino
    connectWiFi();
    setState(STATE_BOOTING);
}

void loop() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("WiFi Disconnected. Attempting reconnect...");
        connectWiFi();
        delay(1000);
        return;
    }

    // Non-blocking Arduino Serial Read
    while (ARDUINO_SERIAL.available() > 0) {
        char c = ARDUINO_SERIAL.read();
        if (c == '\n') {
            if (currentState == STATE_BOOTING || currentState == STATE_WAITING_FOR_ARDUINO) {
                if (!readAndParseArduinoResponse()) {
                    Serial.println("Failed to parse buffered Arduino data.");
                }
            } else {
                Serial.print("Ignoring unexpected Arduino data in state ");
                Serial.print(currentState); Serial.print(": "); Serial.println(arduinoSerialBuffer);
            }
            arduinoSerialBuffer = "";
        } else if (c >= 0) {
            arduinoSerialBuffer += c;
            if (arduinoSerialBuffer.length() > 200) {
                Serial.println("Warning: Arduino Serial Buffer overflow. Clearing.");
                arduinoSerialBuffer = "";
            }
        }
    }

    handleStateMachine();
    delay(50);
}

void handleStateMachine() {

    if (currentState == STATE_BOOTING) {
        if (millis() - lastStateChangeTime > 15000 && !initialPositionReceived) {
            Serial.println("Timeout: Still waiting for initial INIT from Arduino...");
            lastStateChangeTime = millis(); // Reset timer to try again
        }
```

```ino
    switch (currentState) {
        case STATE_IDLE:
            if (millis() - lastIdleCheckTime >= IDLE_CHECK_INTERVAL_MS) {
                lastIdleCheckTime = millis();
                getNextTaskFromServer();
            }
            break;

        case STATE_INITIALIZING_TASK:
            reportStatusToServer(StatusInit);
            break;

        case STATE_GETTING_COMMAND:
            getNextCommandFromServer();
            break;

        case STATE_SENDING_TO_ARDUINO:
            if (sendCommandToArduino(lastServerCommand)) {
                arduinoCmdSentTime = millis();
                setState(STATE_WAITING_FOR_ARDUINO);
            } else {
                Serial.println("Failed to send cmd to Arduino. Retrying...");
                delay(500);
            }
            break;

        case STATE_WAITING_FOR_ARDUINO:
            if (millis() - arduinoCmdSentTime > ARDUINO_RESPONSE_TIMEOUT_MS) {
                Serial.println("Timeout waiting for Arduino response!");
                lastArduinoStatus = StatusTimeout;
                // Report timeout as obstacle to server to force check/replan
                reportStatusToServer(StatusObstacle);
            }
            // State transition happens in readAndParseArduinoResponse
            break;

        case STATE_REPORTING_TO_SERVER:
            reportStatusToServer(lastArduinoStatus.c_str());
            break;

        case STATE_TASK_FINALIZING:
            Serial.println("Server indicated task complete/error. Returning to Idle.");
            currentTaskID = -1;
            setState(STATE_IDLE);
            break;

        case STATE_ERROR:
            Serial.println("Entered ERROR state. Halting operations.");
```



```
void setState(RobotState newState) {
    if (currentState != newState) {
        Serial.print("State Changing: "); Serial.print(currentState);
        Serial.print(" -> "); Serial.println(newState);
        currentState = newState;
        lastStateChangeTime = millis();
        if (newState == STATE_IDLE) {
            lastIdleCheckTime = millis();
            currentTaskID = -1; // Clear task ID when becoming idle
        }
         if (newState == STATE_BOOTING) {
             initialPositionReceived = false; // Reset flag when rebooting/restarting flow
         }
    }
}


void getNextTaskFromServer() {
    Serial.println("Requesting next task from server...");
    String url = "http://" + String(serverIp) + ":" + String(serverPort) + "/tasks/next";
    String response = makeHttpRequest("POST", url, String("")); // Empty payload for POST

    if (response == "ERROR" || response == "TIMEOUT") {
        Serial.println("Failed to get task assignment from server.");
        return;
    }

    StaticJsonDocument<512> doc;
    DeserializationError error = deserializeJson(doc, response);

    if (error) {
        Serial.print("Failed to parse task assignment JSON: "); Serial.println(error.c_str());
        if (response.indexOf("No pending tasks available") != -1) { Serial.println("Server: No tasks."); }
        else { Serial.println("Unexpected server response format."); Serial.println("Response: " + response); }
        return;
    }

    if (!doc.containsKey("id") || !doc.containsKey("station")) {
        Serial.println("Error: Assignment JSON missing 'id' or 'station'.");
        Serial.println("Response: " + response);
        return;
    }

    currentTaskID = doc["id"].as<long>();
    int station = doc["station"].as<int>();
    Serial.printf("Received Task ID: %ld for Station: %d\n", currentTaskID, station);
    setState(STATE_INITIALIZING_TASK);
}
```

```
void reportStatusToServer(const char* status) {
    if (currentTaskID < 0) {
        Serial.println("Error: Cannot report status, no active Task ID.");
        setState(STATE_IDLE); return;
    }
    // Ensure we have a position before reporting, especially for INIT
    if (!initialPositionReceived && strcmp(status, StatusInit) == 0) {
        Serial.println("Error: Trying to report INIT status before receiving position from Arduino.");
        setState(STATE_BOOTING); // Go back to waiting for Arduino
        return;
    }


    Serial.printf("Reporting status '%s' for Task %ld (Pos: %d,%d Theta: %d)...\n",
        status, currentTaskID, lastReportedX, lastReportedY, lastReportedTheta);

    String url = "http://" + String(serverIp) + ":" + String(serverPort) + "/tasks/" + String(currentTaskID) + "/report_status";

    StaticJsonDocument<200> payloadDoc;
    payloadDoc["x"] = lastReportedX; payloadDoc["y"] = lastReportedY; payloadDoc["theta"] = lastReportedTheta;
    payloadDoc["last_cmd_status"] = status;
    String payload; serializeJson(payloadDoc, payload);

    String response = makeHttpRequest("POST", url, payload);

    if (response == "ERROR" || response == "TIMEOUT") {
        Serial.println("Failed to report status to server. Will retry...");
        delay(REPORT_RETRY_DELAY_MS);
        // Stay in REPORTING_TO_SERVER to retry
        return;
    }

    StaticJsonDocument<128> doc; // Check server response
    DeserializationError error = deserializeJson(doc, response);
    if (error || !doc.containsKey("message")) { Serial.print("Warning: Could not parse OK response after reporting status. Assuming OK. Resp: " + response); Serial.println();}
    else { Serial.print("Server response: "); Serial.println(doc["message"].as<String>()); }

    setState(STATE_GETTING_COMMAND);
}

void getNextCommandFromServer() {
    if (currentTaskID < 0) { Serial.println("Error: Cannot get command, no Task ID."); setState(STATE_IDLE); return; }

    Serial.printf("Requesting next command for Task %ld...\n", currentTaskID);
    String url = "http://" + String(serverIp) + ":" + String(serverPort) + "/tasks/" + String(currentTaskID) + "/next_command";
    String response = makeHttpRequest("GET", url, String("")); // Empty payload for GET
```

```
    if (response == "ERROR" || response == "TIMEOUT") { Serial.println("Failed to get cmd. Retrying..."); delay(GET_CMD_RETRY_DELAY_MS); return; } // Stay in state

    StaticJsonDocument<256> doc;
    DeserializationError error = deserializeJson(doc, response);

    if (error || !doc.containsKey("command")) {
        Serial.print("Failed parse command JSON: "); if(error) Serial.println(error.c_str()); else Serial.println("Key missing");
        Serial.println("Response: " + response); Serial.println("Retrying..."); delay(GET_CMD_RETRY_DELAY_MS); return; // Stay in state
    }

    lastServerCommand = doc["command"].as<String>();
    Serial.print("Received command: "); Serial.println(lastServerCommand);

    if (lastServerCommand == CmdComplete) { Serial.println("Server: Path Complete!"); setState(STATE_TASK_FINALIZING); }
    else if (lastServerCommand == CmdError) { Serial.println("Server reported ERROR."); setState(STATE_TASK_FINALIZING); }
    else if (lastServerCommand == CmdNone) { Serial.println("Server: NO_COMMAND (Waiting?). Retrying..."); delay(GET_CMD_RETRY_DELAY_MS); /* Stay in state */ }
    else if (lastServerCommand == CmdForward || lastServerCommand == CmdTurnLeft || lastServerCommand == CmdTurnRight) { setState(STATE_SENDING_TO_ARDUINO); }
    else { Serial.print("Unknown command: "); Serial.println(lastServerCommand); setState(STATE_TASK_FINALIZING); /* Treat as error */ }
}

bool sendCommandToArduino(String command) {
    if (command == "") { Serial.println("Err: Empty cmd to Arduino."); return false; }
    String formattedCmd = "CMD:" + command + "\n";
    Serial.print("Sending Arduino: "); Serial.print(formattedCmd);
    size_t bytesSent = ARDUINO_SERIAL.print(formattedCmd); ARDUINO_SERIAL.flush();
    if (bytesSent != formattedCmd.length()) { Serial.println("Warn: Bytes mismatch sending?"); }
    return bytesSent > 0;
}


bool readAndParseArduinoResponse() {
    if ((currentState == STATE_BOOTING || currentState == STATE_WAITING_FOR_ARDUINO) && arduinoSerialBuffer.startsWith("STATUS:")) {

        Serial.print("Parsing valid STATUS message from Arduino: "); Serial.println(arduinoSerialBuffer);

        char statusStr[20] = {0};
        int x = -1, y = -1, t = -1;

        // Parse the incoming string
        int parsedCount = sscanf(arduinoSerialBuffer.c_str(), "STATUS:%19[^:]:X:%d:Y:%d:T:%d", statusStr, &x, &y, &t);

        if (parsedCount == 4) {
            String tempStatus = String(statusStr);
            tempStatus.toUpperCase();
            lastArduinoStatus = tempStatus;

            lastReportedX = x;
```

```
String makeHttpRequest(String method, String url, String payload = "") {
    WiFiClient client; HTTPClient http; String response = "ERROR";
    Serial.print("HTTP "); Serial.print(method); Serial.print(": "); Serial.println(url);
    if (payload != "") { Serial.print("Payload: "); Serial.println(payload); }
    http.setTimeout(HTTP_TIMEOUT_MS);
    if (http.begin(client, url)) {
        int httpCode = 0;
        if (method == "GET") { httpCode = http.GET(); }
        else if (method == "POST") { http.addHeader("Content-Type", "application/json"); httpCode = http.POST(payload); }
        else { Serial.println("Unsupported HTTP method"); http.end(); return "ERROR"; }
        if (httpCode > 0) {
            response = http.getString();
            Serial.print("HTTP Resp Code: "); Serial.println(httpCode);
            if (response.length() < 200) { Serial.print("HTTP Resp Body: "); Serial.println(response); }
            else { Serial.println("HTTP Resp Body (truncated)..."); }
            if (httpCode < 200 || httpCode >= 300) { Serial.print("HTTP req potential fail code: "); Serial.println(httpCode); }
        } else {
            Serial.printf("[HTTP] %s fail, err: %s\n", method.c_str(), http.errorToString(httpCode).c_str());
            if (httpCode == HTTPC_ERROR_CONNECTION_REFUSED || httpCode == HTTPC_ERROR_SEND_HEADER_FAILED || httpCode == HTTPC_ERROR_CONNECTION_FAILED) { response = "ERROR"; }
            else if (httpCode == HTTPC_ERROR_READ_TIMEOUT) { response = "TIMEOUT"; }
            else { response = "ERROR"; }
        }
        http.end();
    } else { Serial.printf("[HTTP] Unable to connect %s\n", url.c_str()); response = "ERROR"; }
    return response;
}

void connectWiFi() {
    if (WiFi.status() == WL_CONNECTED) return;
    Serial.print("Connecting WiFi: "); Serial.print(ssid);
    WiFi.mode(WIFI_STA); WiFi.begin(ssid, password);
    unsigned long startAttemptTime = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime < WIFI_CONNECT_TIMEOUT) { Serial.print("."); delay(500); }
    if (WiFi.status() != WL_CONNECTED) { Serial.println("\nWiFi Fail!"); setState(STATE_ERROR); }
    else {
        Serial.println("\nWiFi connected!"); Serial.print("IP: "); Serial.println(WiFi.localIP());
        if (currentState == STATE_BOOTING || currentState == STATE_ERROR) {
            Serial.println("WiFi ok, -> BOOTING (wait Arduino Init)...");
            setState(STATE_BOOTING);
        }
    }
}
```