T.C.

AKDENİZ ÜNİVERSİTESİ

TARIM ROBOTLARI İÇİN IMU VE GPS DESTEKLİ YERSEL KOORDİNAT SİSTEMİ PLATFORMUNUN GELİŞTİRİLMESİ

Evren BAŞER

FEN BİLİMLERİ ENSTİTÜSÜ

TARIM MAKİNALARI VE TEKNOLOJİLERİ MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS

MAYIS 2022

ANTALYA

T.C.

AKDENİZ ÜNİVERSİTESİ

TARIM ROBOTLARI İÇİN IMU VE GPS DESTEKLİ YERSEL KOORDİNAT SİSTEMİ PLATFORMUNUN GELİŞTİRİLMESİ

Evren BAŞER

FEN BİLİMLERİ ENSTİTÜSÜ

TARIM MAKİNALARI VE TEKNOLOJİLERİ MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

MAYIS 2022

ANTALYA

T.C.
AKDENİZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TARIM ROBOTLARI İÇİN IMU VE GPS DESTEKLİ YERSEL KOORDİNAT SİSTEMİ PLATFORMUNUN GELİŞTİRİLMESİ

Evren BAŞER

TARIM MAKİNALARI VE TEKNOLOJİLERİ MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

Bu tez T.C. Akdeniz Üniversitesi
Bilimsel Araştırma Projeleri Koordinasyon Birimi
Tarafından FYL-2020-5264 nolu proje ile desteklenmiştir.

MAYIS 2022

T.C.

AKDENİZ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

TARIM ROBOTLARI İÇİN IMU ve GPS DESTEKLİ YERSEL KOORDİNAT SİSTEMİ PLATFORMUNUN GELİŞTİRİLMESİ

Evren BAŞER

TARIM MAKİNALARI VE TEKNOLOJİLERİ MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

Bu tez 09.05.2022 tarihinde jüri tarafından Oybirliği / Oyçokluğu ile kabul edilmiştir.

Prof. Dr. Mehmet TOPAKCI	……
Prof. Dr. Deniz YILMAZ	……
Dr. Öğr. Üyesi İlker ÜNAL	……

ÖZET

TARIM ROBOTLARI İÇİN IMU ve GPS DESTEKLİ YERSEL KOORDİNAT SİSTEMİ PLATFORMUNUN GELİŞTİRİLMESİ

Evren BAŞER

Yüksek Lisans Tezi, Tarım Makinaları ve Teknolojileri Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. MEHMET TOPAKCI

Mayıs 2022; 93 sayfa

Hassas tarım faaliyetlerinde gerçekleştirilen çıktılarda istenilen verim konumlandırma işlemleri ile doğrudan ilişkilidir. Ekim, dikim, gübreleme, ilaçlama ve hasat işlemlerinde tarım alanının verimli kullanılması, iş gücü, zaman ve maddi maliyetlerin en aza indirgenmesi amaçlanarak tarım alanı üzerinde koordinat sistemi oluşturularak otonom tarım makinalarının tarımsal faaliyetlerdeki temelini kapsamaktadır.

Yöntem olarak tarımsal faaliyetlerde otonom sistemlerin kullanımı artmaktadır. Traktörlere entegre edilebilen otomatik dümenleme sistemi en önemli örneğidir. Otomatik dümenleme ve otonom sistemlerin temelinde konumlandırma işlemleri bulunmakta ve elektronik ve yazılım tabanında çalışmaktadır. Disiplinler arası ihtiyacı doğuran bu yaklaşım ile ülkemizde tarımsal faaliyetlerde elektronik ve yazılım tabanlı yersel koordinat sistemi platformu çalışması ilk olma niteliği taşıyacaktır.

Çalışmanın temelinde konumlandırma amaçlı kullanılan GPS tabanlı sistemler oldukça yaygın olmakla beraber tarımsal faaliyetlerde de kullanılmaktadır. Ancak GPS tabanlı sistemlerde gerek maliyet gerekse konumlandırma hatalarından dolayı tekil kullanımı tercih edilememektedir. Planlanan süreçte yersel koordinat sistemi platformu, GPS destekli hata düzeltme işlemleri ile kapalı devre koordinat sistemi oluşturma ve hareketli sistem koordinat sistemi karşılaştırma ölçümleri ile beraber gerçekleştirilerek konum hesaplama süreci gerçekleştirilecektir. Konum hesaplamalarında ortaya çıkan hassasiyet doğrultusunda ilaçlama ve gübreleme öncelikli olarak kullanılacaktır. Sistemi modüler yapıda tasarlayarak elektrikle çalışan hareketli bütün tarım makinalarında kullanılabilir özgünlükte olması hedeflenmiştir.

Mobil robot üzerinde geliştirilen Sensor Fusion sisteminden ve RTK GPS'ten elde edilen konum bilgileri değerlendirildiğinde ortalama olarak 0,11 m sapma değeri belirlenmiştir. Her iki sistemden elde edilen konum bilgileri sapma değerlerine göre RMSE (Ortalama Karekök Sapması) 0,13 m olarak belirlenmiştir.

ANAHTAR KELİMELER: GPS, IMU, Konumlandırma, Rota Planlama, Sensör Füzyon, Tarım robotları

JÜRİ: Prof. Dr. Mehmet TOPAKCI

Prof. Dr. Deniz YILMAZ

Dr. Öğr. Üyesi İlker ÜNAL

ABSTRACT

DEVELOPMENT OF IMU AND GPS APPLIED GROUND COORDINATE SYSTEM PLATFORM FOR AGRICULTURAL ROBOTS

Evren BAŞER

MSc Thesis in Agricultural Machinery and Technologies Engineering

Supervisor: Prof Dr. MEHMET TOPAKCI

May 2022; 93 pages

The desired yield in the outputs realized in precision agriculture activities is directly related to the positioning processes. It covers the basis of autonomous agricultural machinery in agricultural activities by creating a coordinate system on the agricultural area with the aim of efficient use of the agricultural area in sowing, planting, fertilizing, spraying and harvesting processes, minimizing labor, time and material costs.

As a method, the use of autonomous systems in agricultural activities is increasing. The most important example is the automatic steering system that can be integrated into tractors. Automatic steering and autonomous systems are based on positioning processes and work on an electronic and software basis. With this approach, which creates an interdisciplinary need, the study of an electronic and software-based terrestrial coordinate system platform in agricultural activities in our country will be the first.

GPS-based systems used for positioning on the basis of the study are quite common, but are also used in agricultural activities. However, in GPS-based systems, singular use cannot be preferred due to both cost and positioning errors. In the planned process, the location calculation process will be carried out by performing the terrestrial coordinate system platform, GPS assisted error correction processes, creating a closed circuit coordinate system and moving system coordinate system comparison measurements. Spraying and fertilizing will be used primarily in line with the sensitivity in location calculations. If the measurements obtained are considered sufficient, it can also be used in sowing, planting and harvesting processes. By designing the system in a modular structure, it is aimed to make it usable in all electrically mobile agricultural machines.

When the location information obtained from the Sensor Fusion system developed on the mobile robot and the RTK GPS were evaluated, an average deviation of 0.11 m was determined. According to the deviation values of the position information obtained from both systems, the RMSE (Mean Square Root Deviation) was determined as 0.13 m.

KEYWORDS: Agricultural Robots, GPS, IMU, Navigation, Positioning, Sensor Fusion

COMMITTEE: Prof.Dr. Mehmet TOPAKCI
Prof.Dr. Deniz YILMAZ
Asst. Prof. Dr. İlker ÜNAL

ÖNSÖZ

Günümüz teknolojisinin her geçen günde ilerlemesi, birçok alanda yeni ihtiyaçların ve yeni metodolojilerin ortaya çıkmasıyla tecrübe edilmesi gereken, farklı açılardan değerlendirilmesi gereken birçok konu ortaya çıkmıştır. Transistörün icadı ile endüstri 3 devriminin hayatımıza etkilerinden bahsedilirse, aylarlarca yapılan hesaplama işlemlerinin bir kaç saniyeye indirgenmesi, posta yolu yerine elektronik ortamda saniyeler içinde kıtalar arası bilgi transferinin gerçekleştirilmesi yeni ihtiyaçlar ve yeni sorunlar ortaya çıkarmıştır.

Tarımsal üretim faaliyetleri kapsamında hassas tarım yaklaşımlarının ortaya çıkması birçok ülke tarafından uygulanarak elde edilen verim ile beraber bu alanın önemini arttırmış ve karşılaşılan engeller üzerinde çözüm getirme ihtiyacını da arttırmıştır. Hassas tarım faaliyetlerinin en önemli parçalarından birisi de otonom robotlardır. Bu çerçevede açık alan otonom sistemlerinin kullandığı GPS sensörlerinin rolü büyüktür. Ancak kendine has bazı problemlerin ortaya çıkmasında, hassas tarım süreçlerinde ihtiyaç duyulan hassas konumlandırma gibi süreçlerinde bazı ek ihtiyaçlar ortaya çıkarmıştır. Bu çalışmada hedeflenen, GPS kullanımı otonom hareketlerde IMU sensörü ile elde edilen verilerin GPS hatalarının düzeltilmesinde kullanılmasını, otonom hareketlerin gerçekleştirilmesi için gerekli mobil robotun geliştirilmesi ve navigasyon işlemlerinin gerçekleştirilmesini sağlamaktır. Bu işlemlerin gerçekleştirilmesinde, maaliyeti düşük sensörler ve elektronik donanım modüleri kullanılarak bakım onarım süreçlerinde de kolaylık sağlanması hedeflenmiştir.

Yüksek lisans eğitimimin her sürecinde bilgisini, yardımını ve emeğini esirgemeyen danışman hocam Sayın Prof. Dr. Mehmet TOPAKCI'ya, bilgi ve tecrübelerinden yararlandığım kıymetli hocam Sayın Dr. İlker ÜNAL'a, tez izleme komitesi üyelerine, doktora eğitimime başladığım günden bu zamana her türlü konuda bilgilerini, yardımlarını ve tecrübelerini esirgemeyen Sayın hocalarım Prof. Dr. Davut KARAYEL ve Prof. Dr. Murad ÇANAKCI'ya, araştırmaya maddi destek sağlayan Akdeniz Üniversitesi Bilimsel Araştırma Projeleri Koordinasyon Birimine ve Tarım Makinaları ve Teknolojileri Mühendisliği bölümü öğretim üyesi ve araştırma görevlisi hocalarım ve arkadaşlarıma, işyerinde beraber çalıştığım yöneticim Nihat Erim İNCEOĞLU'na , eşim Hatice BAŞER ve babam Musa BAŞER'e teşekkür ederim.

İÇİNDEKİLER

AKADEMİK BEYAN

Yüksek Lisans Tezi olarak sunduğum “Tarım Robotları İçin Imu ve Gps Destekli Yersel Koordinat Sistemi Platformunun Geliştirilmesi” adlı bu çalışmanın, akademik kurallar ve etik değerlere uygun olarak yazıldığını belirtir, bu tez çalışmasında bana ait olmayan tüm bilgilerin kaynağını gösterdiğimi beyan ederim.

09.05.2022

Evren BAŞER

SİMGELER VE KISALTMALAR

Simgeler

dB : Desibel

s : Saniye

V : Gerilim

A : Akım

W : Watt

Hz : Hertz

g : Yer Çekimi Kuvveti

Kısaltmalar

IMU : Ineartial Measure Unit

I2C : Haberleşme Yöntemi

GPS : Küresel Konumlandırma Sistemi

DOF : Serbestlik Derecesi

RMSE : Ortalama Karekök Sapması

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Yaşadığımız gezegende yaklaşık 7,6 milyar insan vardır. 2050 yılında toplam nüfusun 9,8 milyar insana ulaşması tahmin edilmektedir (U. Nations, 2020). Giderek daha azalan oranla gıda üretimini artırma ihtiyacı gibi çeşitli sorunlar günümüzün küresel problemlerindendir. Çünkü 2050 yılına kadar dünya nüfusunun yaklaşık %68'i kentsel ortamlarda yaşaması ve mevcut gıda üretim kapasitesinin iki katına ihtiyaç duyulması olasıdır. 1991'de tarıma elverişli alanların yaklaşık %39,47'sini temsil ettiğinden, ekilebilir arazi önemli ölçüde azalmıştır. (U. Nations Economic & Social Affairs, 2018). Dünyada tarım faaliyetlerinde kullanılan arazi alanı 2013 yılında %37,7'ye düştüğü görülmüştür (X. Zhang vd. 2018). İnsanların, yaşam koşullarını iyileştirme önerisi ile koşullarda küresel kentleşme sürecindeki artış, bireyin insanları daha sağlıklı bir yaşam ve beslenme tarzı aramaya iten finansal gelir ile (L. Zhang vd. 2018) artan yüksek kaliteli gıda talebini çözmek için gıda üreticileri, dünya çapında rekabetçi tarımda kalmak için maliyet tasarrufu sağlayan yöntemler aramaktadırlar. Endüstri pazarı, kırsal üreticiler, verimlerini en üst düzeye çıkarmak için tarımsal faaliyetlerine yeni yöntem ve teknikler ilave etmektedirler. Çünkü dünya genelinde tarımsal faaliyetlerin maliyetlerini yaklaşık %39'u işgücü neden olduğu maliyetler oluşturmaktadır. Bununla beraber tarımsal faaliyetlerde çalışacak kalifiye eleman bulunamaması tarımsal faaliyetlerin çıktılarında verim düşüklülüğüne neden olmaktadır (Castellanos vd. 2019). Bu yaşanan süreçlerle beraber hassas tarım uygulamaları önem kazanmıştır. Tarımsal faaliyetlerin ihtiyaçlarını giderebilmek için otonom ve robotik sistemler geliştikçe denenmektedir. Gelecekteki ihtiyaçların karşılanabilmesi için teknolojinin getirdiği imkanların beraberinde oluşan yeni sorunlarla beraber incelenmesi gerekmektedir.

Hassas tarım faaliyetlerinde GPS sensörünün kullanımında bazı sınırlılıklar söz konusudur. GPS sensörünün engeller üzerinden sinyal yansımaları ya da sinyalin elde edilememesi, gezegen yörüngesinde bulunan uydulara erişilememesi ve bağlı olduğu gömülü sistem dahilinde verilerin doğru zamanada ele alınamaması yani gecikmelerin yaşanması sonucunda konumlandırma hataları meydana gelmektedir. Zaman döngüsü dahilinde oldukça hassas değerlendirilmesi gereken verilerin sonuç olarak istenilen birim çalışma zamanında doğru kararın alınarak otonom sistemlere doğru hareket bilgisinin verilmesi gerek otonom sistem hareket dahilinde farklı oluşan durum sonucunda otonom sistemin farklı yöne doğru hareket etmesi ya da durması için kritik öneme sahiptir. Bu anlamda çalışmanın içerisinde GPS sensörü ile beraber dahil edilmiş IMU sensörünün yardımıyla elde edilen verilerin matematiksel modellerle GPS sensöründen gelen konum hatalarını düzeltme süreçlerine, IMU sensöründen elde edilen doğrulama işlemleri dahil edilmiştir. IMU sensörü üzerinden ivme, yön, yer çekimi, manyetik pusula bilgileri ele alınarak sapma açısı düzeltmeleri, kalman filtreleme, doğrusal hızlanma parametreleri üretilmiş hareket halinde iken GPS konum verileri üzerinde düzeltme işlemleri gerçekleştirilmiştir.

Farklı IMU sensörleri üzerinde kontrollü deneyler gerçekleştirilmiştir. Sensör ve sinyal düzeltme fonksiyon çıktıları üzerinde karşılaştırmalar yapılarak gerçekleştirilen hareket veya manevra üzerinde oluşan ivme, yön değişimleri gözlemlenerek en uygun IMU sensörü belirlenmiştir. Kalman filtreleyicisi ile IMU sensöründe titreşimlerden kaynaklanan gürültü sinyallerinin sönümlenmesi işlemleri gerçekleştirilerek, elde edilen verilerin doğrulanması sağlanmıştır. GPS konum verilerinin daha hızlı anlamlandırılabilmesi için 10 DOF IMU sensörü üzerinden manyetik alan değerleri

kullanılmıştır. Mobil robot sabit hızla ilerlerken doğrusal hızlanma tespiti yaparak IMU sensörlerinden gelen verilerin aynı süreçte GPS sensöründen elde edilen verilerin doğrulama işlemlerine devam eden ilerleyiş esas alınmıştır.

Otonom hareketlerin gerçekleştirilmesi, sensörlerden verilerin alınması, alınan veriler üzerinde işlemlerin gerçekleştirilmesi, kullanıcı tarafından hedef koordinat bilgisinin verilmesi, motor hareketlerinin gerçekleştirilmesi için çeşitli karar verme fonksiyonlarının işlemlenmesi süreçleri tek sıra halinde algoritmik olarak ele alındığında zaman kayıpları gerçekleşmesi muhtemeldir. Bu doğrultuda işlem süreçlerinin girdi ve çıktıları tespit edilmiştir. Tespit edilen girdi ve çıktılar, hangi fonksiyonlarda girdi olarak kullanıldığı sınıflandırılmıştır. Çoklu görev yetisine sahip mikro denetleyici kart kullanılarak yazılım sınıflandırılarak her bir işlem sınıfı için ayrı işlem çekirdeğine atanmıştır. Bu yaklaşımın kullanılması eş zamanlı çalışma sürecini devreye sokarak yazılım üzerinde verilerin işlenmesi, çıktı birimlerine gönderilmesi süreçlerinde bekleme yapmadan motor hareketlerinin sağlanması ve hareket sonucunda oluşan yeni durumun değerlendirilmesine olanak sağlamıştır.

Düşük maliyetli elektronik birimlerle çalıştırılması hedeflenen mobil robotun elektronik donanımlarında arıza durumu olmasıyla ilgili parça kolayca değiştirilebilmektedir. Mobil robotun üzerinde bulunan elektronik bileşenler farklı GPS ve IMU sensörleri ile kullanılabilir. Sistem, deneysel karşılaştırma işlemlerine uygun tasarlanmıştır. Elde edilen konum verileri, yaygın kullanılan harita servislerine dönüştürülebilmekte ve farklı gözlemleme platformlarına bu veri sunulabilmektedir. Bunu dışında yazılım üzerinde bulunan haberleşme protokolü sayesinde internete bağlanabilen mobil robota uzaktan komut gönderebilme alt yapısı kurulmuştur. Karşılıklı mesajlaşma yaklaşımı ile kurgulanan haberleşme hizmeti üzerinden kullanıcı ile mobil robot arasında iletişim alt yapısı geliştirilmiştir.

Hassas tarım faaliyetlerinde önde gelen ihtiyaçlar arasında, yer alan otomatik dümenleme özelliğine sahip düşük maliyetli otonom robot prototip çalışmaları yer almaktadır. Çalışmada, GPS ve IMU sensörlerinin beraberinde kullanıldığı planlanan rota üzerinde hareket edebilen, hedef koordinatların verilmesiyle görevlendirilerek tarım arazilerinde ilgili hedef noktaya erişimi gerçekleştirilerek sürücüye ihtiyaç duymadan tarımsal faaliyetleri gerçekleştirmede hareket kabiliyetine sahip otonom araç geliştirilmesi ve düşük maliyetli elektronik donanımlardan oluşan modüler bir yapı kurgulanması hedeflenmiştir.

2. KAYNAK TARAMASI

2.1. Gömülü Sistem Teknolojisi

Gömülü sistem teknolojileri, kendine özgü giriş ve çıkış birimleri tasarlanmış veya karmaşık bir sistemin parçası olarak özel bir işlevi yerine getirmek için geliştirilmiş yazılıma sahip mikroişlemci tabanlı özel bilgisayar donanım sistemidir. İşlemci çekirdeklerinde, gerçek zamanlı işlemler için hesaplama yapmak üzere üretilir ya da modifiye edilmektedirler (Furht vd., 2012).

Gömülü sistem yapılarında, basit bir kullanıcı ara yüzünden karmaşık grafik kullanıcı ara yüzlerine kadar, tek bir mikro denetleyiciden bağlı çevre birimleri ve ağları olan bir işlemci takımına kadar değişir. Gömülü bir sistemin kompleks yapısı, tasarlandığı göreve bağlı olarak büyük ölçekte değişmektedir. Gömülü sistem uygulamaları, hesap makinesi gibi basit çaplı sistemlerden, mega fabrikalar veya uzaya gönderilen roketlere kadar çeşitlilik gösterir. Üretilen tüm mikroişlemcilerin neredeyse tamamı gömülü sistemlerde kullanılmaktadır (Furht vd., 1991).

Statik yazılım olarak isimlendirilen gömülü sistem programlama yöntemleri, belli bilgisayar donanımı kaynaklarıyla çalışan statik okunur bellekte veya geçici bellek yongalarında saklanır. Dış dünya ile çevre birimleri aracılığıyla, giriş ve çıkış cihazlarını birbirine bağlar.

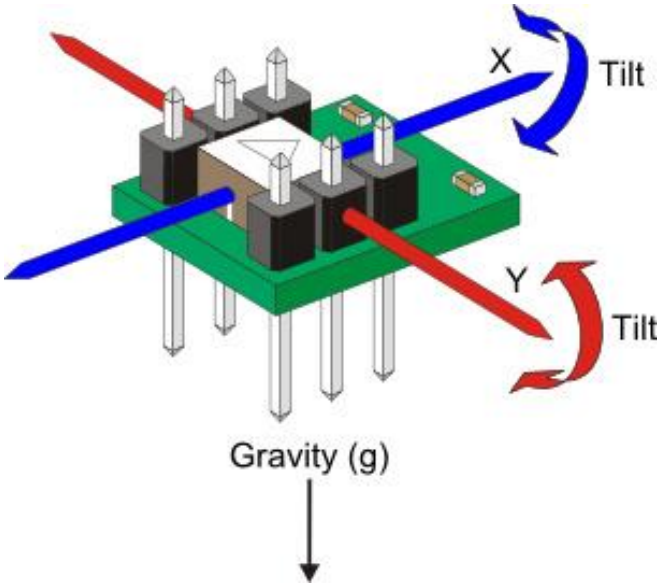
Sensör, fiziksel miktarı ölçer ve elektrik sinyaline dönüştürür, bu daha sonra gömülü bir sistem mühendisi veya herhangi bir elektronik cihaz tarafından okunabilir. Bir sensör, ölçülen miktarı belleğe kaydeder (Guha vd., 1995).

Gömülü sistem, herhangi bir sistem içerisinde yer alarak, o sistemi akıllı hale getiren elektronik donanım ve yazılım ile oluşmuş entegre sistemdir. Gömülü sistemler adından da anlaşılacağı üzere genellikle daha büyük bir sistemin içinde gömülü olarak bulunurlar ve belirli bir amacı gerçekleştirmeye yönelik çalışmaktadırlar.

2.2. IMU

Uzayda hareket halindeki bir cismin üstünde oluşan 3 eksen ivme ve 3 eksen dönme kuvvetini ölçmek için iki tip sensörden oluşan “Inertial Measurement Unit” (IMU) adı verilen cihazlar, günümüzde uzay araçları, insansız hava araçları gibi birçok hareketli platformun hareketinin stabilize edilmesinde kullanılmaktadır (Zhao ve Wang, 2012).

IMU, ana işlemciye gönderilen açısal hız ve doğrusal ivme verisini tek bir modülde toplayan elektronik bir birimdir. IMU temelde iki farklı sensör içerir. Bunlar ivmeölçer, jiroskoptur. İvmeölçer üç farklı eksende üç ayrı sinyal üretir. İtici birimden ve fiziksel etkilerden dolayı, ivme ölçen bu sensörler yer çekiminden etkilenmektedir. Sensörler daimi olarak yer çekiminin etkisinde kalmaktadırlar. Ölçü skalası olarak bir, iki veya üç eksende ölçüm yapabilen türevleri vardır. Bunlar $\pm 1g, \pm 2g, \pm 4g$ vb. gibi değerler ile ifade edilmektedir. Şekil 2.1’de sensör hareket jiroskop ivme ölçer yapısı verilmiştir (Tanenhaus vd., 2012).



Şekil 2.1. IMU Sensörü jiroskop ivmeölçer yapısı

Jiroskop ve ivmeölçer daha stabil veri elde edinimine gereksinim duymaktadır. Bu yüzden birbirlerini referans alarak iki sensör birleştirilir ve hız, pozisyon gibi bilgiler tek bir birimden yani IMU'dan alınır. Degrees of Freedom (DOF) terimi IMU'nun serbestlik derecesini niteler. 3 eksen gyro ve 3 eksen ivmeölçerli bir IMU 6 DOF olarak ifade edilir.

Jiroskop ve ivmeölçer tek başlarına kullanıldığında ürettikleri değerlerde kaymalar meydana gelmektedir. Örneğin 3 saniye sonra 1,5 derece kayabilmektedir. Bu nedenle hassas ölçümlerde kullanılamazlar. En ufak titreşimlerde çok yüksek gürültü oluşturan ivmeölçerler ile birlikte jiroskop kullanıldığında bu gürültüler filtrelenmektedir. IMU'da ise jiroskoplar bu kuvvetlerden etkilenmediklerinden referans olarak ivmeölçerler ile birlikte kullanılmaktadırlar (Zhu ve Zhou, 2004).

2.3. GPS

GPS, Global Positioning System; küresel konumlama sistemi anlamına gelen sözcüklerin baş harflerinden oluşan kısaltmadır. Dünya üzerindeki konumunu GPS uydularından en az 3 ya da yükseklik bilgiside istenirse 4 adedinden gelen oldukça hassas atomik saat sinyalleri sayesinde konumunu bulabilen cihazlardır. GPS, Amerika Birleşik Devletleri'nin geliştirdiği bir küresel konumlama hizmetidir. GPS'in dışında Rusya'nın geliştirmiş olduğu GLONASS, Hindistan'ın geliştirmiş olduğu IRNSS, Çin'in geliştirmiş olduğu BeiDou (BDS) ve Avrupa ülkelerinin geliştirdiği Galileo gibi küresel konumlama sistemleri de kullanılmaktadır. Tüm bu sistemlerin birleşimine de Global Navigation Satellite System (küresel uydu seyrüsefer sistemi, GNSS) ismi verilmektedir. Ancak günlük hayatta GPS ismi küresel konumlandırma hizmetlerinin önüne geçmektedir (Kaplan ve Christopher, 2017).

GPS'in çalışma prensibi, dünya yörüngesinde bulunan GPS uydularında oldukça hassas özel olarak tasarlanmış atomik saatler bulunmaktadır. Sistem içerisinde kullanılan

atomik saatler, diğer uydular ve yeryüzünde bulunan karşılık gelen atomik saatler ile senkronize olarak çalışmaktadır. Herhangi bir sapma günlük olarak düzeltilmektedir. Uydular, yörüngede bulundukları konumu ve bu hassas saat bilgisini sürekli olarak Dünya'ya göndermektedirler. Dünya üzerindeki herhangi bir GPS alıcısı çalıştırıldığında, kapsama alanında bulunan GPS uydularından en az 3 tanesinden bu sinyallere ihtiyaç duyar ve gelen saat bilgilerinin mutlak zamandan ne kadar sapma yaptığını bularak her bir uydudan uzaklığını öğrenebilen GPS alıcısı, bu bilgiler ile dünya üzerindeki konumunu hesaplayabilmektedir (Kaplan ve Christopher, 2017).

Navigasyon cihazları ve cep telefonlarımız ise konum bilgisini kullanarak haritada o an bulunulan konumu gösterebilir ve seçilen hedefe ulaşılması için gerekli rota bilgilerini hesaplayarak en kısa yolu, en ekonomik yolu seçebilmetedir. Bu işlem için cihaz içerisinde harita bilgisinin yüklü olması gereklidir. Cep telefonlarında ise, harita bilgisi önceden yüklenebileceği gibi, internet bağlantısı aracılığıyla en güncel harita bilgisi elde edilebilmesinin yanı sıra anlık birçok farklı konumla ilişkilendirilebilir bilgiye de erişilebilmektedir (Kaplan ve Christopher, 2017).

Bu sistem ilk olarak tamamen askeri amaçlar için kurulmuştur. Bu sistemden faydalanarak askeri çıkartmalarda kolayca yön bulmak ve roket atışlarında nokta atışı yapmak hedeflenmiştir. Ancak, 1980'lerde GPS sistemi sivil kullanıma da açılmıştır. Fakat GPS alıcılarının doğada kullanılabilecek kadar küçük ve ucuz olmasından sonra yaygın şekilde kullanılmaya başlanmıştır. Şu an GPS alıcılarının arabada, gemide veya doğada kullanılmak üzere üretilmiş birçok modeli mevcuttur. Dünyada ünlü GPS alıcısı üreticileri olarak Garmin, TomTom, Magellan gibi markalar sayılabilir (Hong vd., 2006).

Amerika savunma bakanlığı, gezegen yörüngesine 24 GPS uydusu fırlatmıştır. Bu sayı 2008 yılında 32'ye ulaşmıştır. Bunlardan bazıları sadece Amerikan ordusunun kullanımındadır. Belli yükseklikte bulunan uydular yeterli görüş alanına sahiptirler ve yerküre üzerinde bir GPS alıcı istasyonunun en az 4 adet uyduyu görebileceği şekilde yörüngede gezinmektedirler. Uydular, dünyanın her tarafına düşük zaman peryotlarında yer ve zaman bilgisi yayınlar. Küçük, elde kullanılan portatif GPS sensörü bu uydulardan sinyalleri alır ve sinyal içinde bulunan metinsel ifadeleri çözerek, konum bilgilerini elde eder. Yatayda iki boyutlu yer bilgisini elde etmek için (enlem ve boylam), alıcı üç uydudan sinyal almak zorundadır. Dördüncü sinyal alınabilirse, yükseklik de belirtilir (Kaplan ve Christopher, 2017).

3 boyutlu uzayda düşündüğümüzde ise, bu sefer çemberlerin kesişimleri değil kürelerin kesişimleri söz konusu oluyor. İlk örneğe benzer bir örnek vermek gerekirse; A uydusundan aldığınız sinyaller size, A uydusundan 10 km uzakta olduğunuzu söylüyor. Bu da demek oluyor ki A uydusu merkez olmak üzere, 10 km yarıçaplı bir kürenin üzerinde herhangi bir noktadasınız (Arabboevna vd., 2021)

GPS alıcılarının birkaç kör noktası vardır. Uyduların gönderdiği sinyaller "Görüş Hattında" Line of Sight ilerler. Yani bulutlardan, camdan ve plastikten geçebilir, ancak ağaç ve tepe gibi katı cisimlerden geçemez. En büyük sorun, kalın bir orman örtüsü altında veya derin vadi ve kulvarlar içinde cihazın yeterli uydu sinyali alamaması ihtimalidir. Ayrıca GPS sinyalleri katı yansıdığı için engebeli araziye ve bulutlu havalarda hassasiyet azalır. Bu durumda ya açık alanda ilerlemeli ya da bilinen ek sensör destekleriyle kullanılması gerekmektedir (Abdukadirova vd., 2021).

2.4. Yazılım Geliştirme

Raspberry Pi üzerinde Python ile çalışmanın en iyi yanlarından biri, Python'un platformda uyumlu olmasıdır. Raspberry Pi Vakfı, gücü, çok yönlülüğü ve kullanım kolaylığı nedeniyle özellikle Python'u ana dil olarak seçmiştir. Python, Raspbian'a önceden yüklenmiş olarak gelmektedir.

Raspberry Pi üzerinde Python yazmak için birçok farklı seçenek vardır. En uygun seçenekler, Mu düzenleyicisi ve SSH üzerinden uzaktan düzenleme seçeneğidir.

Raspbian işletim sistemi, programları yazmak için kullanabilecek önceden yüklenmiş birkaç Python IDE'si ile birlikte gelmektedir. IDE'lerden birisi de ana menüde bulunabilen Mu düzenleyicisidir. “Raspberry Pi Icon → Programming → Mu” yolunu takip edilerek erişilmektedir. Mu düzenleyicisi ilk kez açıldığında, editör için Python modunu seçme seçeneği sunulur. Bu öğreticideki kod için Python 3'ü seçilir. Python 3 versiyonu birçok sensör ve diğer çevre birimlerinin erişim kütüphanelerini barındırmasıyla avantaj sağlamaktadır.

Mu düzenleyicisinin Raspbian sürümüne önceden yüklenmemiş olma ihtimali vardır. Mu yüklü değilse, her zaman aşağıdaki dosya konumuna gidilerek yüklenebilir. Raspberry Pi için önerilen yazılımı içeren bir diyalog açılır. Mu düzenleyicisi yüklenebilir ve aktif edilebilir. Mu düzenleyicisi, Raspberry Pi'de Python'a başlamak için kullanıcı dostu editör sağlamaktadır. Raspberry Pi'ya uzaktan erişim ile Python yazılım dosyaları yüklenebilmektedir. Raspbian, Raspberry Pi'ye SSH üzerinden uzaktan bağlanılmasına izin vermektedir.

Raspberry Pi'ye SSH üzerinden bağlanmadan önce, Raspberry Pi Tercihleri alanında SSH aktif hale getirilmektedir. Yapılandırma göründüğünde, arayüzler sekmesi seçilir ve ardından SSH bağlantısı gerçekleştirilmektedir. Raspberry Pi'nin IP adresi atanması gerekmektedir ve bu işlemlerin sonucunda başka bir ortamdan Raspberry Pi yönetilebilmektedir. IP adresini belirlemek için Terminal uygulamasına erişilmesi gerekmektedir. Terminale “Aksesuarlar” menüsünden erişilebilir. Terminale bağlandığında “pi@raspberrypi:~ $ hostname -I” kod bütünü ile host bilgisine erişilir. Raspberry Pi için geçerli IP adresine ulaşılır. IP adresi ile artık Raspberry Pi'e uzaktan bağlanabilmeye uygundur. Raspberry Pi'nin IP adresini kullanarak artık başka bir bilgisayardan SSH'ye geçilebilir. Raspbian kurulumu sırasında kurulum sihirbazını çalıştırırken kullanıcı tarafından tanımlanmış şifrenin girilmesi gerekir. Belirtilen işlemlerle Raspberry Pi üzerinde Python programlamaya hazır konuma gelinmiştir.

Raspberry Pi üzerinde Python ile projeler oluşturmaya başlanmadan önce, kod için özel bir dizin oluşturmak önemlidir. Raspberry Pi, birçok farklı dizine sahip eksiksiz bir dosya sistemine sahiptir. Hazırlanan Python projeleri için ayrılmış bir yere sahip olmak, her şeyin düzenli ve kolay bulunmasına yardımcı olmaktadır.

Raspberry Pi'a erişmek için SSH tercih edildiğinde, python-projects dizini oluşturmak için komut satırı kullanılır. Raspberry Pi komut satırına erişileceği için proje dosyalarını düzenlemek için bir komut satırı metin düzenleyicisi kullanılması gerekir. Farklı ortamlara ait yazılım geliştirme platformlarında ve yazılım geliştirilebilen farklı cihazlarda da geçerli yazılım geliştirme ortamlarından elde edilmiş yazılım dosyalarının

da Raspberry Pi ortamına aktarılması mümkündür. Raspberry Pi'deki dosyaları uzaktan düzenlemek için VS Code'u da kullanabilir. Fiziksel bileşenlerle etkileşim için, Raspberry Pi'a bağlı çeşitli GPIO cihazlarıyla etkileşim kurmak için kullanımı kolay bir arayüz sağlar.

Breadboard devreleri, bileşenleri birlikte lehimlemek zorunda kalmadan devrenin hızlı bir şekilde prototiplenmesini sağlamaktadır. Breadboard'lar genel bir düzeni takip eder. Sağ ve sol taraflarda, iki ray devre tahtası boyunca uzanır. Bu raylardaki her delik birbirine bağlıdır. Genellikle bunlar pozitif (voltaj veya VCC) ve negatif (toprak veya GND) olarak adlandırılır. Çoğu devre tahtasında, pozitif ray pozitif bir işaretle (+) işaretlenmiştir ve yanında kırmızı bir çizgi olacaktır. Negatif ray negatif işareti ile (işaretlenmiş -) ve yanında çalışan mavi çizgi vardır. Kartın iç kısmında, bileşen rayları, devre tahtasının yanlarındaki pozitif ve negatif raylara dik olarak uzanır. Bu rayların her biri, bileşenleri yerleştirmek için delikler içerir. Tek bir raydaki tüm delikler bağlanır. Ortada, breadboard'un iki tarafını ayıran bir oluk vardır. Oluğun karşı taraflarındaki raylar birbirlerine iletken değildir.

Jumper Telleri, atlama telleri, GPIO pinleri ve bileşenleri arasındaki yolları lehimlemek zorunda kalmadan elektronik devrenin bağlantılarını prototiplenmesine olanak tanımaktadır. Üç farklı tipte gelirler, erkek erkeğe, dişi erkeğe, dişiden dişiye şeklindedir.

GPIO Pinleri, Raspberry Pi, kartın üst kenarı boyunca kırk GPIO pinine sahiptir. Raspberry Pi'yi harici bileşenlere bağlamak için bu GPIO pinleri kullanılabilir. Raspberry Pi'de beş farklı tipte pin bulunmaktadır. GPIO pini giriş veya çıkış için kullanılabilen genel amaçlı pinlerdir. 3V3 pinleri, bileşenler için 3,3 V'luk bir güç kaynağı sağlar. 3,3 V ayrıca tüm GPIO pinlerinin sağladığı dahili voltajdır. 5 V pinleri, Raspberry Pi'ye güç sağlayan USB güç girişi ile aynı olan 5 V'luk bir güç kaynağı sağlar. Pasif kızılötesi hareket sensörü gibi bazı bileşenler 5 V gerektirir. GND pinleri devreler için toprak bağlantısı sağlamaktadır.

2.5. İşletim Sistemi

Raspberry Pi'nin işletim sistemi bir microSD kartta saklanır. Raspberry Pi işletim sistemini kurmanın birden çok yöntemi vardır. SD kartı üzerinden ilk kurulum için Raspberry Pi Imager'ı kullanılması gerekmektedir. Raspberry Pi Imager'ı indirdikten sonra kurulum uygulaması başlatılır. Biçimlendirmek istenen SD kartla birlikte yüklemek istenilen işletim sisteminin seçilmesine olanak tanıyan bir ekran ile kuruluma devam edilir.

Raspbian, ilk açılışta parola yapılandırmasına, yerel ayarların yapılmasına, bir Wi-Fi ağı seçilmesine ve işletim sisteminin güncellenmesine yardımcı olacak bir kurulum sihirbazı sağlamaktadır. Gelen yönlendirme talimatlarına göre kurulum tamamlanmaktadır.

3. MATERYAL VE METOT

3.1. Materyal

3.1.1.Mobil robot

Mobil robot; yazılım, elektronik birimleri, güç kaynakları, DC motor, mekanik aksam ve şase bölümlerinden oluşmaktadır. Robot üzerinde, yazılımların yüklü olduğu iki adet mikro denetleyici, GPS sensörü, IMU sensörü, güç kaynakları için motorları besleyen bir akü, mikro denetleyicilerin bulunduğu iki adet akü, hareket sistemi için iki adet motor ve motor sürücü kartları, 4 tekerlek, alüminyum şase, elle kumanda modülü mevcuttur.

3.1.1.1. Motor sürücü kartı

Sisteme hareket veren iki adet motor bulunmaktadır. Her motora, 20 A akıma kadar kontrol edebilen motor sürücü kartı bağlanmıştır. Motor sürücü kartı, 28V'a kadar olan besleme gerilimi altında çalışabilir özelliktedir.

Sürücü kartları, Arduino mikrodenetleyeci platformu ile rahatlıkla kontrol edilebilmektedir. Ürün üzerindeki pasif soğutucu yeterli soğutmayı yaptığından başka soğutucuya gerek duyulmamıştır. Ölçüleri 50x50x43 mm'dir. Çizelge3.1-3.2'de motor besleme bağlantıları ve kontrol pin tanımları verilmiştir. Şekil 3.1'de motor sürücü kartının görseli verilmiştir.

Çizelge 3.1. Motor sürücü kartı besleme bağlantıları

Motor ve besleme bağlantıları	
- B+	Besleme gerilimi (28v'a kadar)
- B-	Besleme toprak bağlantısı
- M+	Motor+ ucu
- M-	Motor- ucu

Çizelge 3.2. Motor sürücü kartı kontrol pinleri

Kontrol Pinleri	
VCC	+5 VDC
GND	Toprak Bağlantısı
IS1, IS2	Akım Test Pinleri
EN1, EN2	PWM girişleri
IN1, IN2	Motor yön kontrol pinleri

Şekil 3.1. 20 A motor sürücü kartı

3.1.1.2. DC motor

Mobil robot çatısında kullanılan iki motor doğru akımla çalışmaktadır. 24 V ve 12 A akü ile beslenmektedir. Her bir motor 250 W gücündedir. Motor üzerinde yük kesildiğinde elektromanyetik fren sistemi devreye girmektedir. İlerleme hızı 6 km/h'e kadar ulaşabilmektedir. DC Motor, 61 cm çapındaki arka 2 tekerleğe hareket vermektedir. DC motora ilişkin görünüm Şekil 3.2'de verilmiştir.

Şekil 3.2. DC motor görünümü

Her bir motor, bir motor sürücüsüne ve elle kumanda sistemine bağlıdır. İki farklı yöntemle aynı anda DC motorun kumanda edilmesi, doğrudan yük alarak ilerletilmesi

tehlikelidir. Bu sorunun çözümü kutuplu anahtarlama tekniği kullanılarak çözülebilmektedir.

3.1.1.3. Mekanik sistem

Mobil robot genel ölçülerinde 115x66x91 cm'dir. Sistemde 250 W güç üreten 2 DC motor, 24 V 12 A jel akü, 6-8 saatlik tam şarj kapasiteli adaptör, alüminyum şase, 25 cm çapında 2 sarhoş tekerlek, 61 cm çapında doğrudan motorlara bağlı 2 adet tekerlek bulunmaktadır.

3.1.1.4. El kontrol sistemi

El kontrol sistemi kendi içerisinde dahili hız ayarı, sinyal ve güç aç/kapa fonksiyon butonlarına sahiptir. Bununla beraber, joystick şeklinde her yöne hareketi kumanda edebilecek özelliklere sahiptir. Şekil 3.3'te el kontrol sisteminin görseli verilmiştir.

Şekil 3.3. El kontrol sistemi

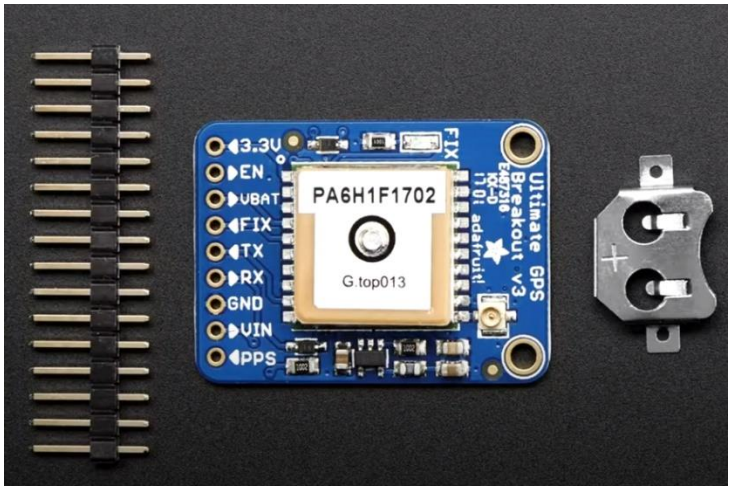
3.1.2. Robot GPS modül

Adafruit marka GPS modülü, kendi sınıfında ve bulunduğu fiyat aralığında en çok tercih edilen GPS modüllerinden biridir. GPS modülü, 165 dBm hassasiyet, 10 Hz güncelleme, 66 kanal veri alış veriş özelliği, 5V besleme tasarımı ve sadece 20 mA akım çekimi, montaj kolaylığı, RTC pil uyumlu, dahili veri kaydı, düzeltmede PPS çıktısı, durum izleme LED'i özeliklerine sahiptir. 66 kanalda 22 uyduyu takip edebilen modül, yüksek hassasiyetli bir alıcıya ve dahili bir antene sahiptir. Saniyede 10 adete kadar konum bilgisi alabilir. GPS modülünde, düşük çıkışlı 3.3 V regülatör ile ENABLE pinini kullanarak 5 V'luk güç sağlanabilmektedir. İsteğe bağlı CR1220 bağlantısı için bir ayak izi olan herhangi bir mikro denetleyici pini kullanarak GPS modülü kapatılmalıdır.

RTC'yi çalışır durumda tutmak ve doğru veri başlangıçlarına izin vermek için GPS pili kullanılmalıdır. GPS sensörü ledi, uydu ararken yaklaşık 1 Hz'de yanıp sönmekte ve uydu bulduktan sonra her 15 seferde bir yanıp sönmektedir (Technology Inc, 2011).

Geliştirilen sistemde kullanılan Adafruit Ultimate GPS aktif edildiğinde hemen seri veri göndermeye başlamaktadır. GPS, paket olarak büyük veri mesajları gönderir ve ardından beklemekte ve daha sonra tekrar veri mesajları göndermektedir. Gönderilen verinin sadece doğrudan okunması yetmez, aynı zamanda anlamlı hale getirmek için verinin sınıflandırılması üzerinde işlemler yapılması gerekmektedir. Bu nedenle verileri sınıflandırmak için karmaşık işlevler yapılması gerekir. Aynı zamanda gelen yeni verilerin hiçbirinin kaçırılmadığından emin olunması gereklidir.

10 Hz'lik veri gönderim işlemini gerçekleştiren GPS modülü saniyede 10 adet veri paketi hazırlar anlamına gelmektedir. Her saniyelik süreçte gelen veri paketleri alınmakta ve algoritma içerisinde kuyrukta bekletilmektedir.



Şekil 3.4. Adafruit Ultimate GPS breakout V3

3.1.3. Yazılım çözümleri

Yazılım çözümleri, donanım birimlerinin kontrolü, farklı donanım birimlerinin haberleşmesi, sensörlerden veri elde edilmesi, elde edilen verinin sınıflandırılması ve işlenmesi, işlenmiş verinin ise motorları doğru tetiklenmesi ile ilgili hareket kararlarının alınmasını kapsamaktadır. Geliştirilen sistemde yazılım dilleri olarak C++ ve Python kullanılmıştır. C++ yazılım dili, motor sürücülerinin tetiklenmesinde rol oynayan Arduino mikrodenetleyici kartında, Python programlama dili ise Raspberry Pi mikrodenetleyici kartında derlenerek kullanılmıştır.

Arduino kartının motor sürücülerini kontrol etme konusunda daha uygun olması, yüksek akım ile çalışan motor sürücü kartları ile herhangi bir arıza durumunda maliyeti düşük şekilde değişiminin gerçekleştirilebiliyor olması ile avantaj sağlamıştır. Raspberry Pi kartı donanımsal olarak daha yüksek özellikte olması ile birlikte ihtiyaç duyulan gerilim ve akım değerleri dışında yük altında kalması durumunda çalışamayacak ya da bozulma durumu ile karşılaşabileceğinden motor sürücü yönetiminden ayrı tutulmuştur.

Raspberry Pi kartı dahilinde sensörlerden veri alınması işlemi, çoklu işlem görevi önem arz ettiğinden tercih edilmesinde etkendir. İki farklı sensörden aynı zaman diliminde veri toplanabiliyor olması, yine aynı birim zarfında elde edilen değerlerin işlenmesi ve motor sürücü kartları için çıktı üretebiliyor olması kritik zaman kaybının engellenmesinde etkendir. Python yazılım dilinin birçok sensör ve donanım birimi ile haberleşmesinde kullanılan ek yazılım kütüphanelerinin bulunması, açık kaynak kodlu olması ek avantajlar sağlamaktadırlar. Kullanılan ücretsiz kod geliştirme araçları çeşitliliği ve platform bağımsızlığı yazılım geliştirme sürecinde kaliteyi arttırmaktadır.

3.1.3.1. Raspberry Pi Os işletim sistemi

Kendi görüntü makinasına sahip olan Raspberry Pi işletim sistemi, Linux çekirdeğine sahiptir. Birçok işletim sistemi Raspberry Pi ile çalıştırılabilir. Raspberry Pi işletim sistemi, donanımla daha uyumlu olması ve performans açısından birçok avantajı olması dolayısıyla tercih edilmiştir. Sensörler ile doğrudan beraber çalışacak olan işletim sistemi ilgili sensör kütüphaneleri ile doğrudan çalışır ve veri elde etme süreçlerinde etkin rolü bulunmaktadır.

Raspberry Pi Os işletim sistemi dahilinde bulunan Mu düzenleyicisi python kodlama dilinin yazılmasında kullanılmıştır. Python 3'te hazırlanmış algoritma, sensörlerden veri elde etme, konum bilgilerinin hesaplanması ve motor yön hesaplamalarını barındırmaktadır. Bu algoritma, Arduino mikro denetleyicisine ilgili haberleşme işlemlerini gerçekleştirerek gerekli bilgi transferini sağlamaktadır.

3.1.3.2. Arduino kod derleyicisi

Motor kontrol kartının tetiklenmesi ile ilgili komutların derlenmesi için ücretsiz bir kod editörü ve derleyicisi olan Arduino kod derleyicisi kullanılmıştır. Derleyici, C++ kodlama dilini desteklemektedir. Derleyici üzerinde Arduino kartına ilgili kodu yükleme özelliği mevcuttur. Derleyici, motor kontrol kartının tetiklenmesi için gerekli yazılımın Arduino kartına yüklenmesi için kullanılmaktadır. Arduino mikrodenetleyici kartına doğrudan erişim ya da arayüz destekli yönetim durumu söz konusu değildir.

3.1.4. Gömülü sistem teknolojisi

Akıllı tarım yaklaşımlarının önem kazanmasında gömülü sistemlerin rolü oldukça büyüktür. Hassas konum sistemleri, otomatik dümenleme gibi sayılabilecek birçok örnekten bahsedilebilir. Akıllı karar verme mekanizmasının sağlanması günümüz teknolojilerinde bilgisayarlar ve bilgisayarların temelinde mikrodenetleyiciler gerçekleştirmektedir. Geliştirilen mobil robotun sensörlerden verileri alması, ilgili hesaplama katmanlarından geçmesi ve hareket yön kararını vermesi, Raspberry Pi kartına yüklenmiş yazılım ile gerçekleştirilmiştir. Raspberry Pi üzerinde bulunan yazılımın çıktısı motorlara belirlenmiş görevlerden oluşmaktadır. Arduino mikrodenetleyici kartı ise motor sürücülerine ilgili tetiklemeleri gerçekleştirmektedir.

3.1.4.1. Raspberry Pi

Raspberry Pi 3 Model B, boyutları küçük, birçok uygulama için kullanılabilen üçüncü nesil Raspberry Pi'dir. Kablosuz LAN ve Bluetooth bağlantısı ekleme özelliği mevcuttur. Navigasyon işlemlerinin merkezi görevindedir. Sensörlere doğrudan

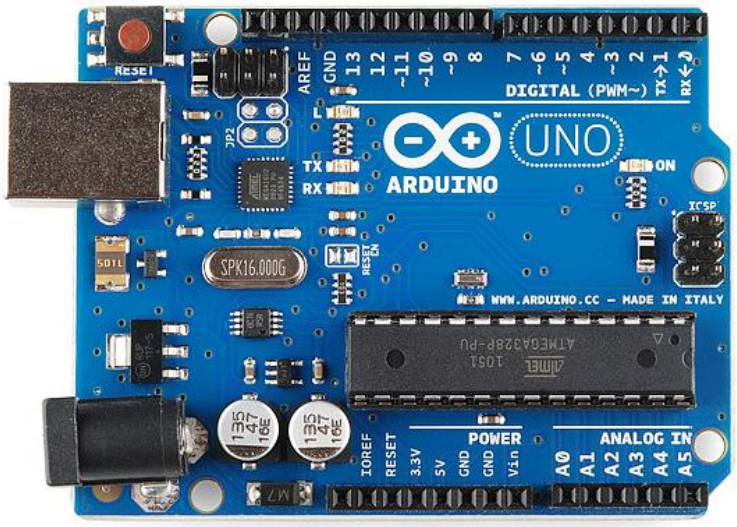
bağlanarak ilgili verileri alarak işler. Raspberry Pi içerisinde çalıştırılmış algoritma ile ilgili hareket görevlerini çıktı olarak üretmek için kullanılmıştır (Sean vd., 2021). Raspberry Pi 3 B+ teknik özellikleri Çizelge 3.3'te verilmiştir.

Çizelge 3.3. Raspberry Pi 3 B+ teknik özellikleri

Teknik Özellikler	
İşlemci	Broadcom BCM-2837B0, Cortex-A53/AMRv8
	Quad-Core 64-bit SoC @ 1.4GHz
Ram	1GB LPDDR2 SDRAM
Bağlantı	2.4GHz and 5GHz IEEE 802.11.b/g/n/ac Kablosuz yerel ağ bağlantısı
	Bluetooth 4.2, BLE
	Gigabit Ethernet, USB 2.0 Üzerinden (Maksimum 300Mbps)
	4 × USB 2.0 Port
GPIO	Genişletilmiş 40 Adet Pin
Video ve Ses	1 × HDMI
	MIPI DSI ekran portu
	MIPI CSI kamera portu
	4 Kutuplu 3.5mm Ses + Kompozit video portu
Multimedya	H.264, MPEG-4 decode (1080p@30)
	H.264 encode (1080p@30)
	OpenGL ES 1.1, 2.0 Grafik
Hafiza	Micro SD Kart Girişi, işletim sistemi ve data için
Güç Beslemesi	5V/2.5A DC, Mikro USB Port Üzerinden
	5V DC, GPIO Pinler Üzerinden
	Ethernet üzerinden güç
	(Power over Ethernet, PoE)(Ayrı PoE HAT Gerekmektedir)
Çevre Şartları	Çalışma sıcaklığı, 0–50°C

3.1.4.2. Arduino

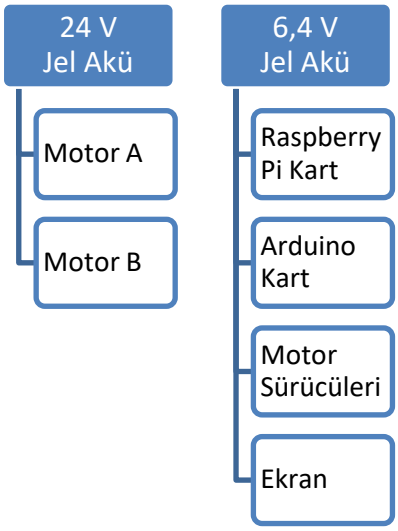
Arduino Uno, ATmega328P'ye dayalı bir mikro denetleyici kartıdır (Şekil 3.5). 14 dijital giriş/çıkış pinine (6 tanesi PWM çıkışı olarak kullanılabilir), 6 analog girişe, 16 MHz seramik rezonatöre (CSTCE16M0V53-R0), USB bağlantısına, güç bağlantısına, ICSP başlığına ve sıfırlama düğmesine sahiptir. Kod derlenmesi için bir USB kablosuyla bir bilgisayara bağlanılması ve AC-DC adaptörü veya pil ile çalıştırılması yeterlidir.



Şekil 3.5. Arduino mikrokontrol kartı

3.1.6. Besleme üniteleri

Gömülü sistemleri barındıran devrenin beslenmesi ve motor birimlerinin beslenmesi olmak üzere iki farklı besleme sistemi vardır. Herhangi invertor kullanmadan motor birimlerinin beslenmesinde 24V jel akü, gömülü sistem biriminin beslenmesinde ise 6,4 V jel akü kullanılmıştır. Şekil 3.6'da besleme ünitelerinin beslediği birimlerin dağılım şeması verilmiştir.



Şekil 3.6. Güç kaynaklarının beslediği birimler

3.1.5. IMU

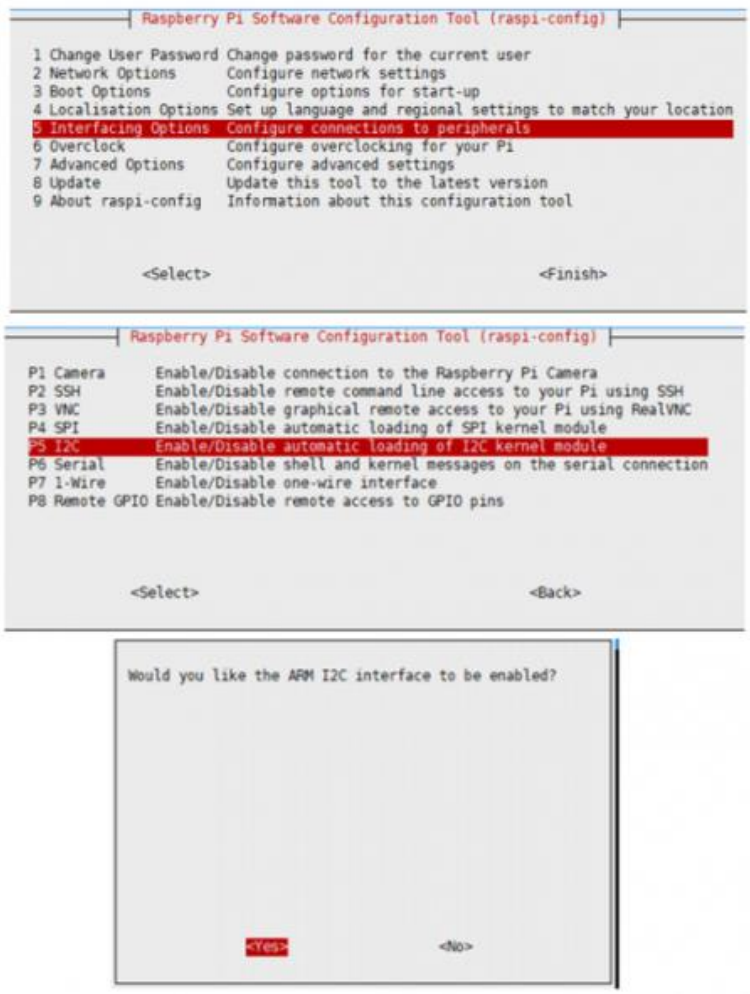
IMU sensörü, GY-521 modülü, MPU-6050 MEMS (Mikroelektromekanik sistemler) için 3 eksenli bir jiroskop, 3 eksenli bir ivmeölçer, bir dijital hareket işlemcisi (DMP) ve bir sıcaklık sensörü içeren bir devre kartıdır. Dijital hareket işlemcisi, karmaşık algoritmaları doğrudan kart üzerinde işlemek için kullanılabilir. Genellikle DMP, sensörlerden gelen ham değerleri kararlı konum verilerine dönüştüren algoritmaları işler. GY-521/MPU-6050 sensöründen doğrudan veri elde edilebilir. Ham sensör değerleri, farklı veri kategorileri olan ivmeölçer ve jiroskop bilgilerini içermektedir. Sensör değerleri, sadece iki kablo (SCL ve SDA) gerektiren I2C seri veri yolu kullanılarak alınmaktadır.

10 DOF IMU Sensörü (D), yerleşik düşük güçlü ICM20948'e sahiptir, hareket izlemenin yanında, konumu, yüksekliği ve sıcaklığı algılama/ölçme özelliklerine sahiptir. Çizelge 3.4'te 10 DOF IMU sensörünün özellikleri verilmiştir.

Çizelge 3.4. 10 DOF IMU Sensörü özellikleri

Özellikler	**Açıklamalar**
ICM20948:	Düşük güçlü 3 eksenli jiroskop, 3 eksenli ivmeölçer ve 3 eksenli pusula/manyetometre
Dahili Dijital Hareket İşleme	Yük boşaltma işlemleri karmaşık füzyon hesaplaması, sensör senkronizasyonu, hareket tanıma vb.
BMP280	Barometrik basınç sensörü
BMP	Sıcaklık Ölçümü
I2C	Veri haberleşme bağlantısı desteği
Güç	3.3V~5V (düşük düşüşlü dahili voltaj regülasyonu)
İvmeölçer	
Çözünürlük	16 bit
Ölçüm aralığı (yapılandırılabilir)	±2, ±4, ±8, ±16g
Çalışma akımı	68.9 μA
Jiroskop	
Çözünürlük	16 bit
Ölçüm aralığı (yapılandırılabilir)	±250, ±500, ±1000, ±2000°/s
Çalışma akımı	1.23mA
Pusula/Manyetometre	
Çözünürlük	16 bit
Ölçüm aralığı	±4900 μT
Çalışma akımı	90 μA
Barometrik basınç sensörü	
Barometrik çözünürlük	0.0016 hPa
Sıcaklık çözünürlüğü	0.01°C
Ölçüm aralığı	300~1100 hPa (rakım: +9000m ~ -500m)
Barometrik bağıl doğruluk	(700 hPa~900hPa, 25°C~40°C): ±0.12 hPa (±1m)
Çalışma akımı	1 Hz güncelleme hızı, ultra düşük güç modu, 2.8 μA

IMU sensörlerinin birim saniyede ürettikleri yoğun veri trafiğini mikro denetleyici tarafından okunup işlenebilmesi için haberleşme yöntemi olarak I2C portu tercih edilmiştir. Diğer portlara göre I2C kısa mesafeli ve yoğun veri trafiği için tasarlanmıştır. İki kanal üzerinden veri transfer işlemi gerçekleştirmektedir (Ahmad vd., 2013). Raspberry Pi işletim sistemi üzerinden I2C portunun aktifleştirilmesi gerekmektedir. Şekil 3.7'de aktifleştirilme işlemi sunulmuştur.



Şekil 3.7. I2C Portunun aktifleştirme işlemi

3.1.7. RTK GPS

Konum bilgilerini doğrulmak için kullanılan GPS alıcısı, Robot GPS+IMU konum bilgilerinin karşılaştırılması ve ilerleme hızının tespit edilebilmesi için kullanılmıştır. Bu nedenle, Robot üzerine Promark 500 GPS (Magellan Co., California, USA) alıcısı ve el kontrol ünitesi monte edilmiştir. GPS alıcısı, 75 kanal ve 20 Hz'e kadar veri çıkış hızına sahiptir. Promark 500, çoklu işletim modu, konfigürasyon, haberleşme modülü (UHF, GSM/GPRS, EDGE) ve protokol sunabilen esnek bir 3G GNSS alıcısıdır. Alıcı, üzerine yerleştirilen telefon sim kartı yardımıyla düzeltme sinyalleri alınmaktadır. Düzeltme sinyalleri CORS TR istasyon ağından alınmıştır. (Şekil 3.8). Konum hassasiyetinin, katalog verilerine göre düzeltme sinyalleri ile milimetre seviyelerine indiği bildirilmiştir. Alıcı ile el kontrol ünitesi arasında Bluetooth üzerinden veri transferi sağlanabilmektedir. GPS alıcı yardımı ile ölçüm yapılan noktaların konum bilgileri 0.5 s aralıkla otomatik olarak el kontrol ünitesine kaydedilmiştir. GPS alıcısına ait teknik özellikler Çizelge 3.5'te verilmiştir.

Şekil 3.8. RTK-GPS Alıcısı ve El Kontrol ünitesi

Çizelge 3.5. RTK-GPS alıcısı teknik özelikleri

Özellik	Değer
Uygulama	Jeodezik Alım Aplikasyon Uygulamaları, İnşaat Jeoloji, Jeofizik Uygulamaları, Hassas GIS Çalışmaları
Alıcı Tanımı	Çift frekanslı GNSS alıcısı
Kanal Sayısı	75
RTK Uzaklığı	100 km
RTK Hassasiyeti (Fix)	Yatay: 10 mm + 1 ppm Düşey: 20 mm + 1 ppm
RTK Hassasiyeti (Flying)	Yatay: 50 mm + 1 ppm Düşey: 200 mm + 1 ppm
Statik Ölçü Hassasiyeti	Yatay: 5 mm + 0,5 ppm Düşey: 10 mm + 1 ppm
Kinematik Ölçü Hassasiyeti	Yatay: 10 mm + 1 ppm Düşey: 20 mm + 1 ppm
SBAS	Yatay < 1 m
Radyo Modem	Magellan 430-470 Mhz
GSM/GPRS Modem	Magellan 4 bantlı GPRS/EDGE modem
Portlar	1 x RS-232, 1 x RS-422, 1 x USB, 1 x Bluetooth, PPS ve Event Marker
Veri Kayıt Aralığı	0.1 - 999 s
Veri Giriş-Çıkış Tipleri	RTCM 2.3, RTCM 3.1, CMR, CMR+, Magellan ATOM, NMEA 0183, NTRIP, DBEN
Veri Çıkış Hızı	20 Hz

3.2. Metot

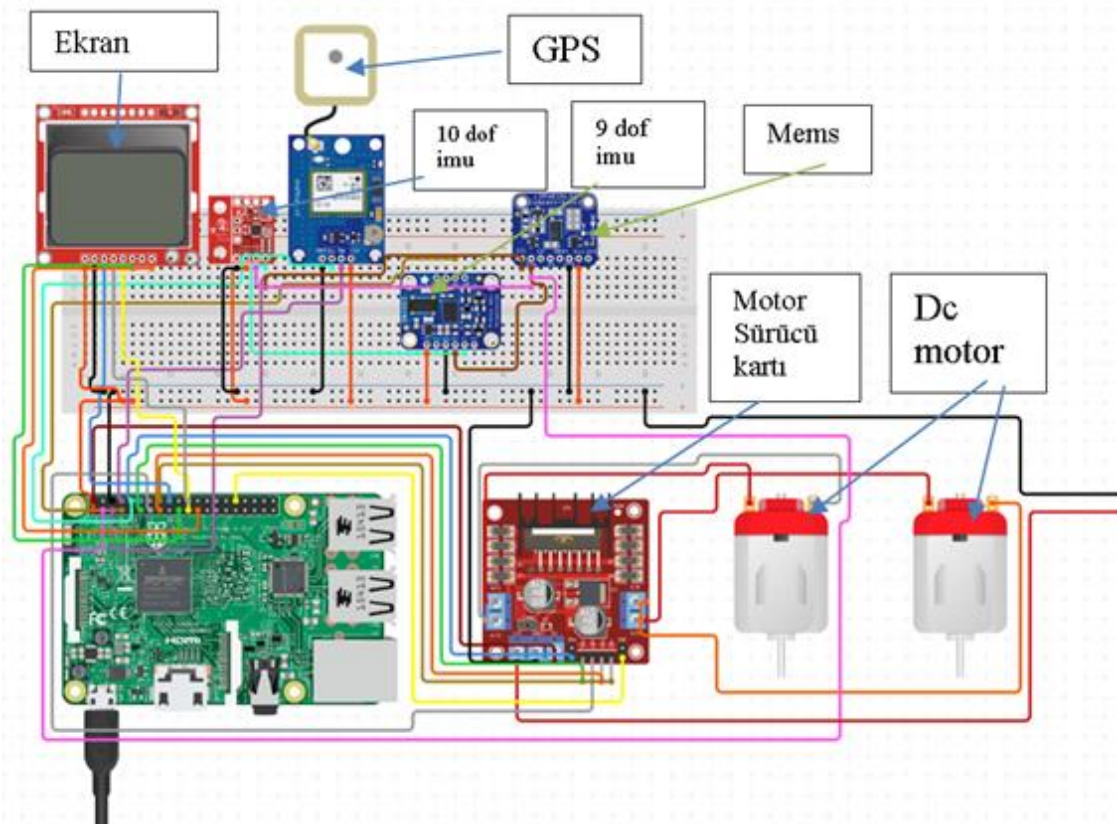
3.2.1. Mobil robot çözümleri

Mobil robot çatısı altında şase üzerinde bağlantılı 2 adet DC motor doğrudan akü ile güç bağlantısı kesilmiştir. Motor sürücülerinin üzerinde bağlantı gerçekleştirilmiş olup anahtarlama ile aktif edilmektedir. Motor sürücülerinin üzerinde soket tipi sigortalar kullanılarak herhangi ters ya da aşırı yüklenme için korumaya alınmıştır. Üst kısmında elektronik bileşenler pano içerisinde muhafaza edilerek Raspberry Pi mikro kartına da ekran bağlantısı gerçekleştirilmiştir. Elektronik bileşenlerin enerjisi farklı akü üzerinden gerçekleştirilmektedir. Şekil 3.9'da mobil robotun gelen görünümü verilmiştir.

Şekil 3.9. Mobil robot görünümü

3.2.1.1. Düşük maaliyetli otonom elektronik yapı tasarımı

Donanımsal bileşenin konum belirlemede ve motor kontrol yöntemlerinin en verimli halde oluşturulabilmesi için yazılımsal yetkinliğin yanında sensörlerin üzerinden elde edilen değerlerin hassasiyeti de göz önünde bulundurulmuştur. GPS değerlerinin hassasiyeti, yapılan birçok çalışmada hata değerleri ortaya çıkarılmıştır. GPS değerlerinin kullanımı daha çok elde edilen değerler ile olması gereken idealdeki değerlerin karşılaştırılmasıyla sapma değerleri elde edilerek kullanılmaktadır. Ancak IMU sensör çeşitlerinin rolü burada oldukça önemlidir. Birim zamanda elde edilen ivme değerlerinin birim zamandaki türevleri elde edilerek, 3 boyutlu yön doğruları üzerinden hız değerleri elde edilebilmektedir. Burada sensörlerin kabiliyeti önemli olmakla birlikte sensörlerin istenilen hassasiyette karşılaştırma testleri yapılmıştır. Geliştirilen sistemde GPS, 10 Dof IMU, 9 Dof IMU, MEMS sensörlerinin bir arada füzyonlayarak çalıştırılmıştır. Buradaki yaklaşım, GPS yanında 3 farklı IMU yapısı ile beraber elde edilen verilerin karşılaştırılarak birbirini doğrulama yaklaşımıdır. Şekil 3.10'da donanımsal tasarım gösterilmiştir.

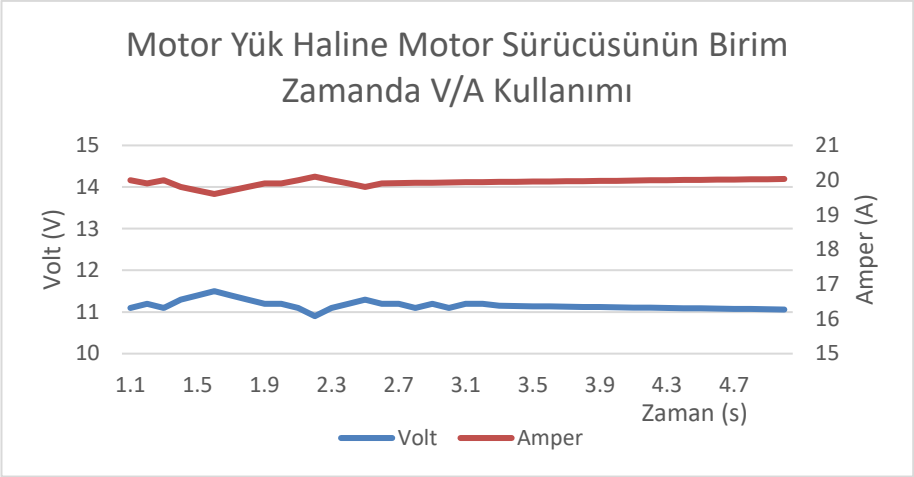


Şekil 3.10. GPS, 10 Dof IMU, 6 Dof IMU, MEMS sensörlerinin gömülü sisteme bağlantı şeması

Çoklu IMU sisteminden birim zamanda elde edilen verilerin bileşkesini alarak elde edilen vektörel bilginin harmanlanması sonucunda kararlı olmadığı gözlemlenmiştir. Bu doğrultuda tekil olarak test edildiğinde sonuç itibariyle 10 dof IMU yapısı çalışma için en uygun sensör olduğu belirlenmiştir.

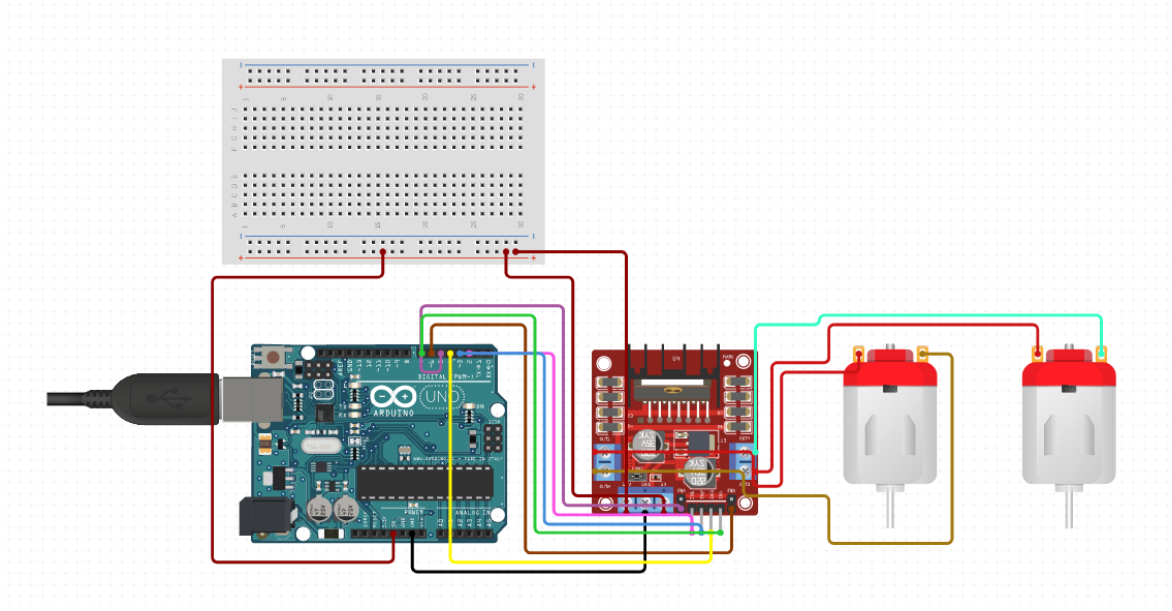
3.2.1.2. Motor hareket değerlerinin izlenmesi

Motor sürücülerinin hareket komutlarının sonucunda aldıkları tetikleme sinyalleri ile birlikte yük halinde gerilim ve akım değerleri ölçümü gerçekleştirilmiştir. Ölçüm işlemlerinde ileri geri yön, uzun süreli çalıştırma testi ile birlikte değerlendirilmiştir. Sürücü devresine tetikleme sinyallerini gönderen Arduino mikro işlemci kartının kod yükleme testi başarılı bir şekilde tamamlanmıştır. Şekil 3.11'de motor yükteyken birim zamanda gerilim ve akım değerlerinin ölçümü gerçekleştirilmiştir.



Şekil 3.11. Motor yük haline motor sürücüsünün birim zamanda V/A kullanımı

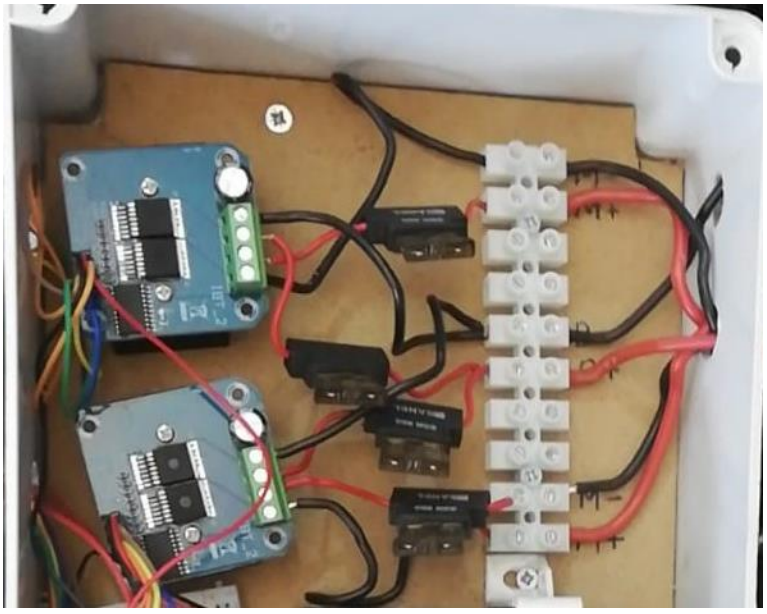
Motor sürücüsü ve motor bağlantılarının şeması Şekil 3.12'de verilmiştir.



Şekil 3.12. Motor sürücüsü ve motor bağlantılarının şeması

3.2.1.3. Motor kontrol sistemi

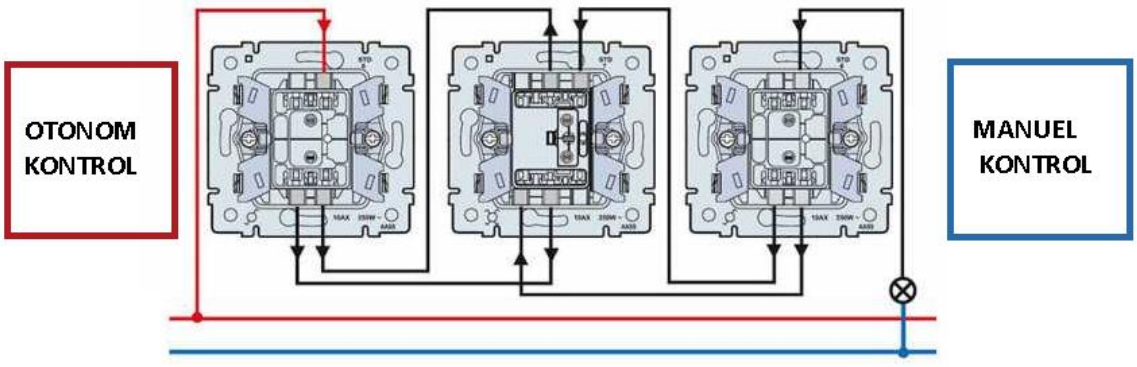
Motor kontrol sürecinin gerçekleştirilmesinde iki adet motor sürücüsü kullanılmıştır (Şekil 3.13). Her biri ayrı motoru tetiklemektedir. Motor sürücülerine bağlı sigorta bağlanarak herhangi fazla yük çekiminde sistemi koruması için kullanılmıştır. 7.5 A'lik sigorta soketleri aşırı yük çekiminde sistemi otomatik olarak kapalı devre haline sokarak korumaya almaktadır. Kendi alüminyum soğutucu panellerine sahip olması ile ekstra soğutma sistemine gerek duyulmamıştır.



Şekil 3.13. Motor kontrol sistemi

3.2.1.4. Otonom manuel geçiş sistemi

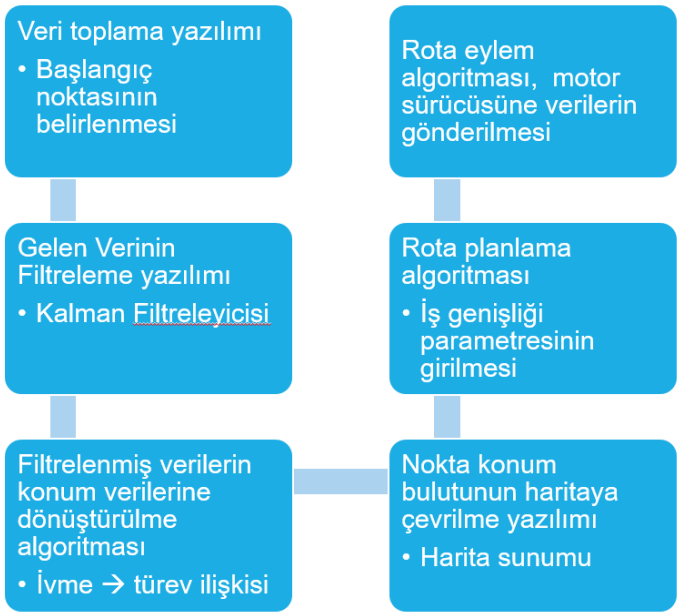
Donanım yapısı içerisinde kullanılan elle kontrol kısmı ile otonom hareketlerin gerçekleştirildiği yapı ile kolay geçişinin mümkün olması için anahtarlama kullanılması gerekmiştir. Kullanılan anahtarlama mekanizması motorlara giden enerji kablolarının üzerine bağlanarak otonom yapıda bulunan motor sürücülerinden gelen enerjiyi keserek elle kontrol kumandasından gelen enerji kabloları arasında geçiş durumu söz konusudur. Bu şekilde anahtarlama yaparak elle kontrol, otonom kontrol ve tamamen enerjiyi kapatma seçenekleri arasında kolaylıkla kullanım söz konusudur. Şekil 3.14'te kutup anahtarlama sistemi ile motorların iki farklı kontrol arasındaki geçiş şekli verilmiştir.



Şekil 3.14. Kutup anahtarlama sistemi ile motorların iki farklı kontrol sistemi arasında geçiş şekli

3.2.2. Yazılım çözümleri

Yazılımsal alt yapının kullanılabilmesi için işletim sistemi olarak Rasbian kurulmuştur. Rasbian işletim sistemi üzerine Python dili kullanılan algoritmalar yüklenerek gelen verinin işlenmesi için çalıştırılmıştır. İhtiyaç duyulan veri işleme hızı ve doğruluğu karşılaştırılmıştır ve bu doğrultuda kullanılacak sistemde tercih edilmiştir. Ön kurulum yazılımı iş akış algoritmasına ilişkin şema Şekil 3.15'te verilmiştir.



Şekil 3.15. Ön kurulum yazılımı iş akış algoritması

IMU sensörlerinden elde edilen verilerin okutularak Kalman filtreleyicisi algoritması ile verilerin doğruluk derecelerinin arttırılması sağlanmıştır.

Yazılımsal altyapı sisteminde, önemli algoritma olan Kalman filtreleyicisi kullanılarak gelen verilerin doğruluğunun en uygun seviyeye çıkarılması sağlanmıştır. Birim zamanda elde edilen verilerin akışında görev alacak olan bu algoritma hatalı olabilecek veri satırları silinerek doğruluğun arttırılması sağlanmıştır.

3.2.2.1. İşletim sistemi

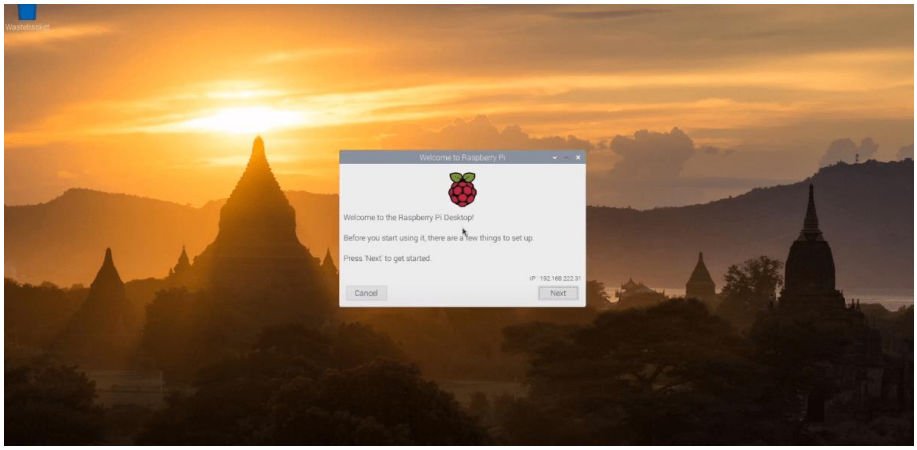
Donanımsal yapının temelini oluşturan Raspberry Pi dünya genelinde erişimi kolay mini bilgisayar kategorisinde küçük ebatlara sahip düşük maliyetli cihazlardandır. İçerisinde işlemci, ram, harici yerleştirilen hafıza birimleri bulunmasının yanı sıra çok çeşitli haberleşme protokollerine sahiptir.

Raspberry Pi için özel olarak hazırlanmış bir Debian sürümü olan Raspbian işletim sistemi tercih edilmiştir. Linux tabanlı olan Debian işletim sistemi kurulmuştur. Bütün yazılım Raspberry Pi işletim sistemi üzerine kurulmuş olup veri depolama işlemlerinin de merkezi göreve sahiptir (Sean vd., 2021).

Sisteme hazırlanacak olan yazılım dillerinin derlenebilmesi için Python ve node.js kurulmuştur. Bu şekilde sensörlerden alınacak verilerin işlenmesi, depolanması, Navigasyon işlemleri ve motor kontrol işlemlerinin kontrolü sağlanmıştır.

Sisteme uzaktan yönetim ve erişim işlemlerinin gerçekleştirilebilmesi için ücretsiz olan VNC yazılımı kurulmuştur. İşletim sisteminde VNC yazılımı aktif edilerek farklı cihazlardan erişimi test edilmiştir. Rasppbery Pi içerisine kurulan server ile VNC yazılımına sahip herhangi bir cihaz ile internet üzerinden erişimi gerçekleştirilebilmektedir.

Özellikle IMU sensörlerinden verilerin sağlanması için I2C portu aktif edilmiştir. İşletim sistemi arayüzünün resmi Şekil 3.16'da gösterilmiştir.



Şekil 3.16. Raspberry Pi ekran bağlantısı ve işletim sistemi arayüzü

3.2.2.2. Haberleşme işlemleri

Raspberry Pi içerisinde ve kullanıcı arasında haberleşme süreci MQTT (Mesaj Kuyruğu Telemetri Aktarımı) protokolleri üzerinden gerçekleştirilmiştir. TCP/IP protokolünün üstünde kullanım için bir ISO standartıdır. (ISO/IEC PRF 20922) yayınla-abone ol tabanlı "hafif" mesajlaşma protokolüdür. Footprint yönteminin gerekli olduğu veya ağ bant genişliğinin sınırlı olduğu uzak konumlara sahip bağlantılar için tasarlanmıştır. Kullanıcı aktif değilken dahi mesajlar erişilebilmektedir (Henriksen vd., 2020).

Her zaman müsait olan bir haberleşme ağına ihtiyaç duyulmaktadır. Tüm ağ için sadece bir tane kanal üzerinden bilgi transfer edilebilmesine olanak sağlamaktadır. Bir bilgisayar ya da bir Raspberry Pi üzerinden gerçekleştirilebilir. Debian tabanlı bir linux işletim sistemi kullanıldığından MQTT haberleşme protokolü kurulmuştur. MQTT arka plan devreye girerek programını yükler ve sürekli çalışır hale gelmektedir.

"robot@ev3dev:~# systemctl status mosquitto" Komutu ile çalışıp çalışmadığı test edilmiştir. Arka tarafta aktif kalan MQTT protokolü Mobil robot yazılımı tarafından doğrudan kullanılabilmektedir. Varsayılan olarak mosquitto 1883 numaralı bağlantı noktasını kullanır).

Çalışmada python programlama dili kullanılmaktadır. Paho-mqtt python kitaplığını kurulması gerekmektedir. Bu modülü kurmak için 'pip3'e ihtiyaç bulunmaktadır. Pip3 Python yazılımlarının derlenebildiği katmandır. Aşağıdaki komut satırları ile işletim sistemi komut istemi üzerinden kurulum gerçekleştirilmiştir.

- ➔ sudo apt-get install python3-pip
- ➔ Artık paho-mqtt'yi yüklenebilir:
- ➔ sudo pip3 install paho-mqtt

Tüm komut dosyaları Raspberry Pi 3'te başarıyla test edilmiştir.

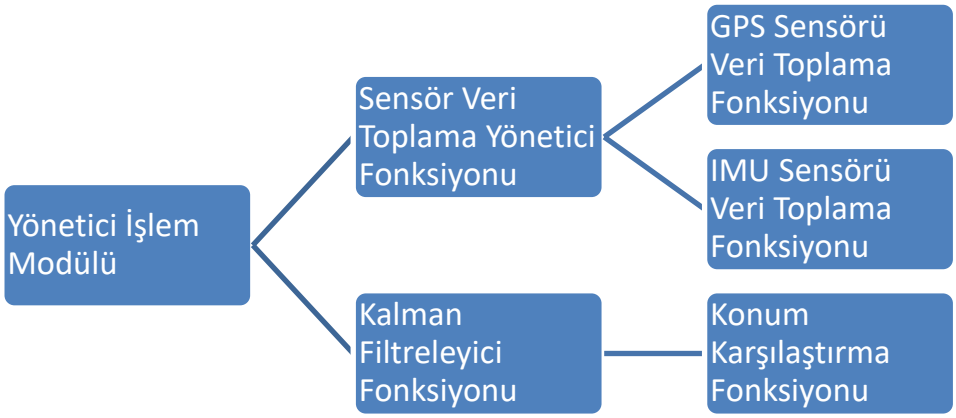
3.2.2.3. Çoklu görev mimarisi ile sensörlerden veri toplama işlemleri

Modern işletim sistemleri, bir kullanıcının aynı anda birkaç uygulamayı çalıştırmasına izin verir, böylece kullanıcı ön planda diğer çalışmalara devam ederken bazı görevler arka planda çalıştırılabilmektedir. Hazırlanan yazılımlar alt işlem parçacıklarına bölünerek yönetilmektedir. Eş zamanlı çalışma süreçleri oluşturulabilmektedir.

Bir görev oluşturmak için işlem veya iş parçacığı kullanabilinir. İşlem, bellek eşleme, dosyalar ve diğer işletim sistemi nesneleri dahil olmak üzere özel kaynaklarına sahiptir. Birden çok iş parçacığı aynı işlem üzerinde çalışabilir ve tüm kaynaklarını paylaşabilir, ancak bir iş parçacığı başarısız olursa, işlemindeki diğer tüm iş parçacıklarını kapatır.

Eş zamanlı çalıştırılan fonksiyonlar için işletim sistemi bir alt süreçler oluşturmaktadır. Her bir fonksiyon için ayrı işlemci görevlendirilmektedir. Fiziksel olarak çoklu çekirdek yapısına sahip işlemciler çalıştırılmak üzere fonksiyonlardan ortaya çıkan veri çıktılarını yazılıma gönderirler (Flurry, 2021).

Çoklu işlem süreçlerinde birbirinin çıktısına ihtiyaç duyan fonksiyonlar eş zamanlı çalıştırıldıklarında bazı problemler ortaya çıkmaktadır. Örneğin sensörlerden veri elde edinim fonksiyonları ile Kalman Filtreleme işlemini gerçekleştiren modül eş zamanlı olarak çalıştırıldığında Sensör veri toplama işlemlerinin çıktısı Kalman Filtreleme işlemlerinde kullanıldığından henüz veri çıktılarına erişemeyeceğinden hatalara sebebiyet verecektir. Bu doğrultuda Kalman Filtreleme işlemlerini gerçekleştiren fonksiyonlar alt işlem parçacıklarında bekleme görevi ile atanmalıdır. Bu doğrultuda sırasını bekleyen Kalman Filtreleyici fonksiyonları ilgili parametreler oluştuğunda hazırda bekleyecek ve görevini gerçekleştirdiğinde yeni parametreleri bekler vaziyete gelecektir. Şekil 3.17'de işlem parçacık hiyerarşisine göre sınıflandırılmış fonksiyonların şeması görülmektedir.



Şekil 3.17. İşlem parçacıklarına göre sınıflandırılmış fonksiyon şeması

Mobil robot yazılımının çoklu görev mimarisi ile yönetimi Alt Yazılım Parçalarının Yönetilmesi konusunda detaylı olarak anlatılmaktadır.

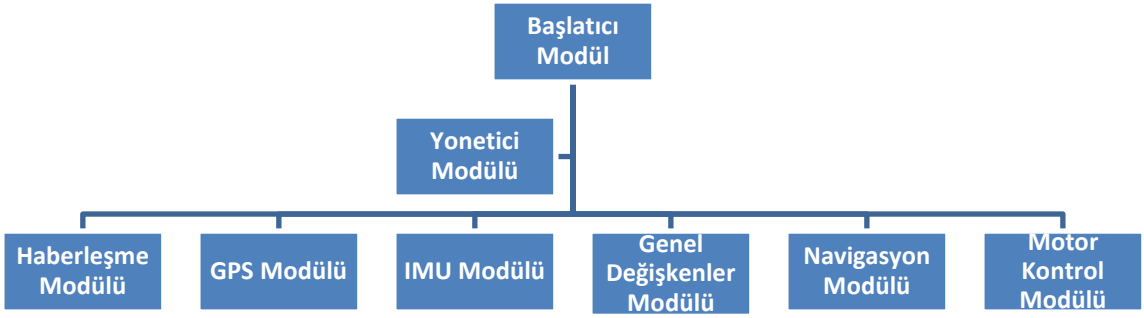
3.2.2.4. Alt yazılım parçalarının yönetilmesi

Mobil robot yazılımının verimli işler hale getirilmesi için temel anlamda amacına uygun olarak görevlendirilmiş alt yazılım modüllerine ayrılmıştır. Başlatıcı modülü, Mobil robot donanımlarının yönetilmesi için kullanılan kütüphaneleri ve python yazılım dilinin derleyici kütüphanelerinin yüklenmesinden sorumludur. Gerekli yükleme işlemlerini tamamladıktan sonra Yönetici modülünü aktif eder.

Yönetici modülü ise Haberleşme, GPS, IMU, Genel Değişkenler, Navigasyon ve Motor Kontrol modüllerinin doğru zamanda gerekli şartlar oluştuğunda devreye alan ana yüklenici modüldür. Aynı zamanda Alt modüllerin çoklu görev mantığında çalışmasını sağlayarak gerektiğinde bu modülleri eş zamanlı çalıştırmasıyla zamandan tasarruf sağlamaktadır (Guan vd. 2021). Yazılımsal olarak zaman gecikmesinden kaynaklı oluşabilecek hataları engellemektedir. Yönetici modülü ilk olarak Haberleşme modülünü devreye alarak kullanıcıdan gelen Hedef GPS koordinatlarının sisteme yüklenmesini sağlar. Sisteme yüklenen bilgiler Genel Değişkenler modülünde tutulur ve geçici bir tarihçe alanında saklı tutar. Genel Değişkenler modülü yazılım içerisinde saklı tutulması gereken değerleri tutarak Yönetici modülü tarafından istenen herhangi bir değeri sunarak çalışma sürecinde veri organizasyonunun deposu konumunda kalmaktadır.

Yönetici modülü eş zamanlı olarak GPS modülü ve IMU modüllerini aktif ederek birim zamanda oluşan verileri rutin dahilinde Genel Değişkenler modülünde tutmaktadır. Her veri bloğu oluşturulduğunda zaman bilgisi ile etiketlenir. IMU ve GPS verileri Navigasyon modülü tarafından istendikçe tekrar işaretlenir. Navigasyon modülü ise elde edilen GPS ve IMU verileri üzerinde Sinyal Veri Filtreleme işlemlerini gerçekleştirirler ve Haberleşme modülünden elde edilen Hedef koordinatları tarafından karşılaştırılma işlemlerini gerçekleştirir. Karşılaştırma işlemleri sonucunda hangi yöne gitmesi gerektiğine karar verme işlemi Navigasyon modülünde bulunmaktadır. İlgili Motor kontrol görevlerinin Arduino miktordenetleyici kartına gönderen Motor kontrol modülüdür. Oluşturulmuş olan rutin işlem her bir hedef koordinat değerlerine ulaşana kadar devam eder ve eş zamanlı olarak elde edilen verilerin kontrolü hesaplanarak Motor

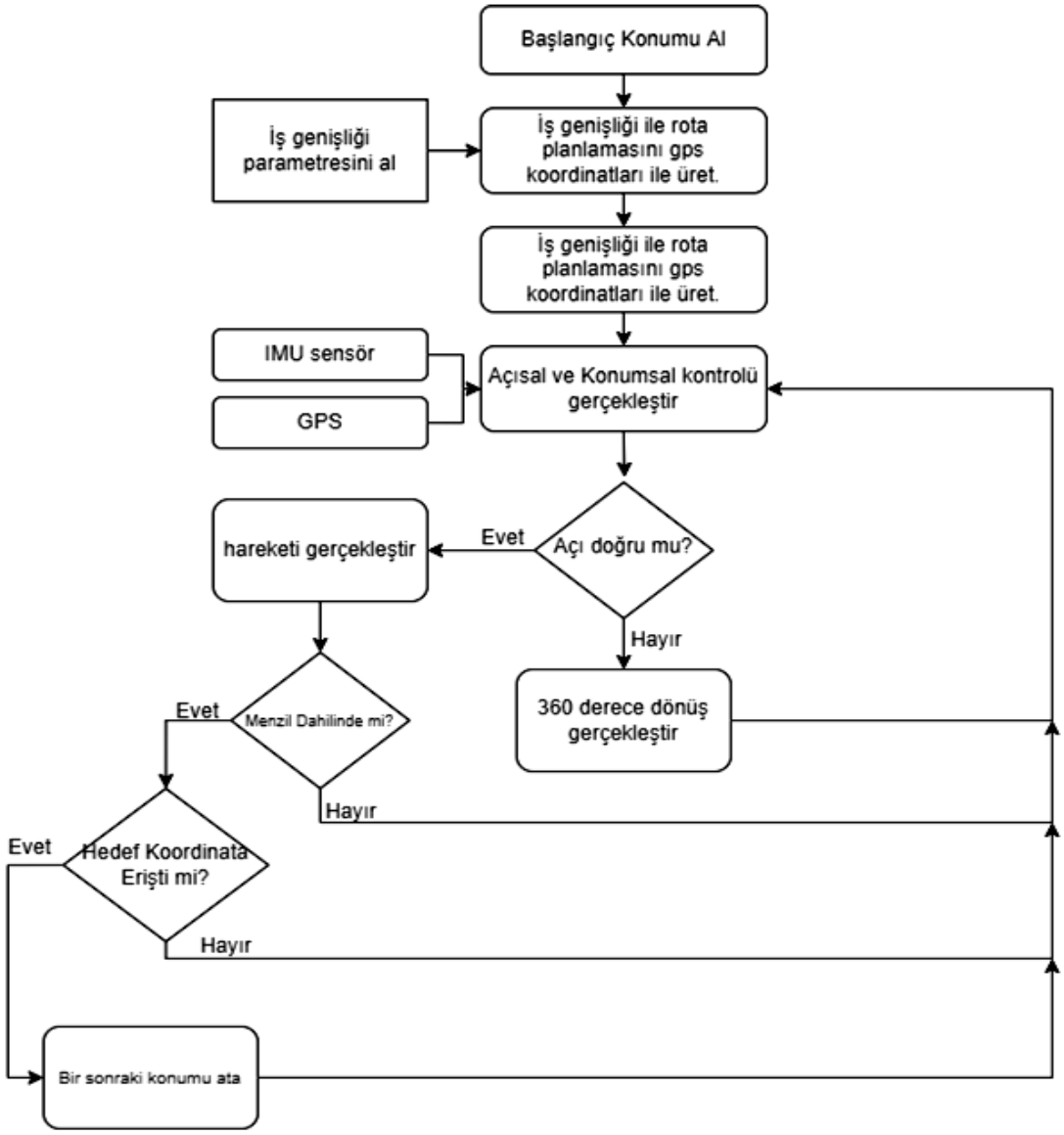
kontrol modülünden son çıktı alınarak hareket işlemi gerçekleştirilir. Şekil 3.18'de modüllerin hiyerarşik şeması verilmiştir.



Şekil 3.18. Yazılım modüllerinin hiyerarşik şeması

3.2.2.5. Rota planlama ve robot hareket algoritması

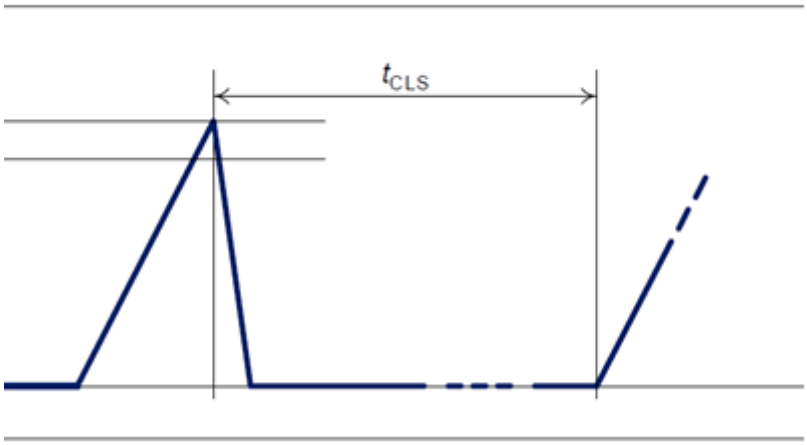
Tasarlanmış mobil robot çalıştırılınca bulunduğu konum referans alınarak başlangıç konumunu kabul etmektedir. Kendisine parametre olarak gelen iş genişliği bilgisi ile güney ve doğu yönünde takip etmesi gereken koordinatları üretmektedir. Koordinatları belleğinde özel bir listede tutarak sırasıyla ilk nokta yönünü kontrol ederek açısal kontrolünü sağlar. Açısal hata durumu söz konusu olduğunda bulunduğu noktada 360 derece dönerek yönelmesi gereken açıyı yakalayana kadar dönmeye devam eder. Prototip robot açısal doğruluğunu sağlamasıyla hareketine başlar. Hareket halindeyken doğru yön ve hedef koordinata ilerlemesini gerçekleştirirken eş zamanlı olarak, *hedef menzili dahilinde mi, Hedef koordinata erişti mi* kontrollerini sağlar. Veri eksikliği veya farklı bir durum oluştuğunda beklemeye geçer. Açısal kontrol ve konum doğrulamasını sağladıktan sonra, hedef noktaya hareket etmeye kaldığı yerden devam etmektedir. Bu doğrultuda hedef noktaya vardığında rota listesinden sıradaki notkayı alarak hareket etme işlemini gerçekleştirir. Şekil 3.19'da Rota Planlama ve mobil robot Hareket Algoritması verilmiştir.



Şekil 3.19. Rota planlama ve genel hareket algoritması

3.2.2.6. Motor kontrol yazılımı

BTS7960 yarım köprü motor kontrolörü işlemlerini Arduino mikrodenetleyici kartı gerçekleştirmektedir. Motor kontrol komutlarını Raspberry Pi ortamından alarak BTS7960 motor sürücüsünün yönetilmesi için ihtiyaç duyduğu sinyallere çevrilmektedir. BTS7960 motor sürücü uygulamaları için tam entegre bir yüksek akım yarım köprüsüdür. 24V Çalışma Gerilimi ve 20A Max Sürekli Akım, Aktif freewheeling ile birlikte 25 kHz'e kadar PWM girişleri ile gerçekleştirilmektedir. Motor yük kesim sinyaline ilişkin resim Şekil 3.20'de verilmiştir.



Şekil 3.20. Motor yük kesim sinyali

Motor sürücü kartında düşük voltaj kapatma özelliği mevcuttur. Düşük voltajlarda tahrik edilen motorun kontrolsüz hareketini önlemek için cihaz kapanır. Besleme gerilimi VUV(KAPALI) 5,4V'nin altına düşerse, Motor sürücüsü Kapanır ve Besleme gerilimi 5,5V veya daha fazlasına yüklenene kadar motor sürücüsü aktif olmamaktadır (Sinaei vd., 2006).

BTS 7960 entegresi dahilinde sıcaklık sensörü ile yüksek sıcaklığa karşı koruma birimi mevcuttur. Yüksek sıcaklık, her iki çıkış aşamasını engeller.

Navigasyon planlama işlemi

Navigasyon planlama süreci donanımsal olarak GPS modülü, IMU sensör modülü ve Raspberry Pi kartı kullanılarak gerçekleştirilebilmektedir. Yazılımsal olarak Haberleşme yazılım modülü, GPS yazılımı, IMU yazılımı ve Kalman filtreleme yazılım modülü ve Navigasyon yazılım modülü kullanılmaktadır.

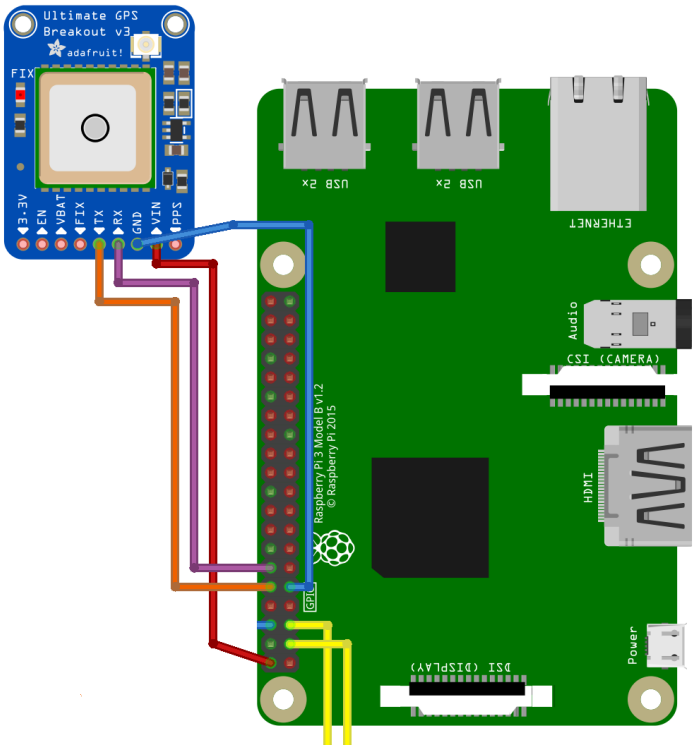
Navigasyon planlama işlemleri sensörlerden girdiler sağlanması ve kullanıcıdan haberleşme modülü üzerinden hedef koordinat talebinin iletilmesi ile başlamaktadır. Navigasyon planlayıcısı hedef koordinat nokta ulaşıldığına karar verir. Kuyrukta başka hedef GPS noktası varsa işleme almaktadır.

GPS Modülü

Navigasyon planlama işlemlerinin temel donanım parçası MTK3339 GPS modülü ve Mikro denetleyici Raspberry Pi kartı 3 B+ modelidir. GPS modülü Raspberry Pi ile doğrudan bağlantılıdır. Bağlantı yapısı Şekil 3.21 ve Çizelge 3.6'da verilmiştir.

Çizelge 3.6. GPS Modülü ve Raspberry Pi bağlantısı

GPS Modülü Pinleri	Raspberry Pi Pinleri
VCC:	5v
GND:	GND
RX:	TX (gpio 14)
TX:	RX



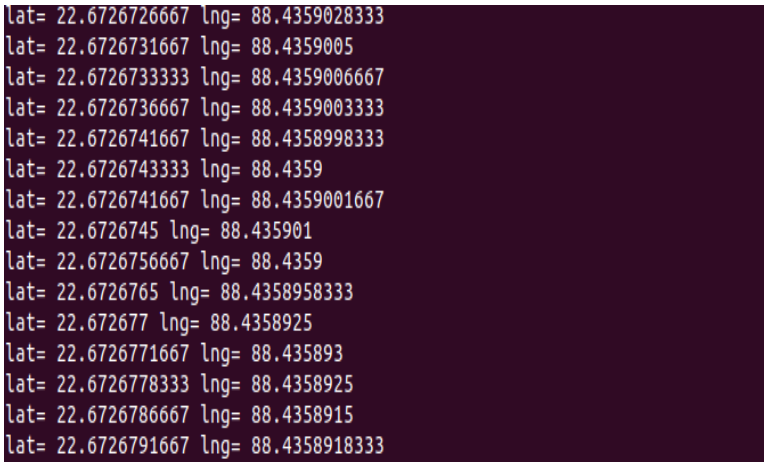
Şekil 3.21. Rasppbery Pi, GPS modülü bağlantı yapısı

Fiziksel bağlantının sağlanmasıyla yazılımsal yapının kurulması için Raspberry Pi üzerinde işletim sistemi ilk katmanda görev almaktadır. İşletim sistemi içerisinde config.txt dosyasının düzenlenmesi gerekmektedir. Temel problem GPS üzerinden veri alış verişini sağlayacak Port, UART portudur. Gerekli yedeklemeler alınması önemlidir. UART portu üzerinde seri konsol özelliği kapatılmaktadır. Bunun sebebi seri port işlemleri yavaştır. GPS üzerinden verilerin yavaş akması konum hatalarına sebep olmaktadır. Config.txt dosyasının içeriğine aşağıda çizelgede bulunan kod satırları öncelikle içerik tamamen silinerek eklenir (Çizelge3.7).

Çizelge 3.7. GPS bağlantısı işletim sistemi port komutları

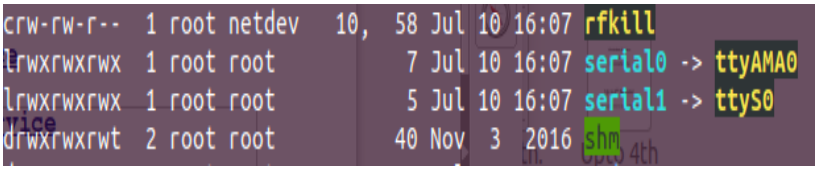
Port Komutları	Açıklamalar
dwc_otg.lpm_enable=0	Lpm aktif hale getirilir.
console=tty1	Tty1 portu adreslenir.
root=/dev/mmcblk0p2	mmcblk0p2 dizinine geçilir
rootfstype=ext4	Ext4 çıkışı adreslenir.
elevator=deadlin	Elevator alanı deadlin ile adreslenir
e fsck.repair=yes rootwait	Fsck
quiet splash	Splash alanı yoksayılır
plymouth.ignore-serial-consoles	Plymouth seri konsol transferinden kapatılır.

Config.txt dosyası düzenlendikten sonra işletim sistemi yeniden başlatılır. GPS modülü veri transfer işlemi test edilmelidir. GPS modülü üzerinde sinyal LED'i kontrol edilerek işletim sistemi üzerinden "sudo cat /dev/ttyAMA0" komutu çalıştırılır. GPS sensöründen elde edilen örnek konum bilgileri Şekil 3.22'de gösterilmiştir.

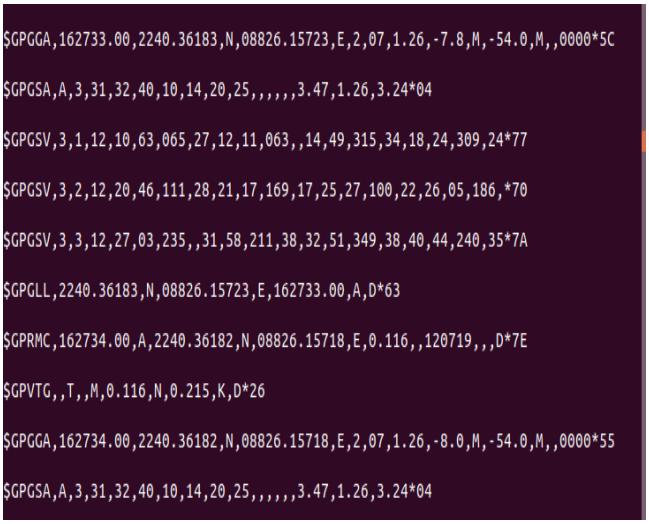


Şekil 3.22. GPS veri kümesi

Mobil robot navigasyon yazılımı Python dilinde yazılmıştır. İşletim sistemi üzerinden bazı ayarlamaların yapılması gerekir. Serial 0 bağlantı portu üzerinden veri transfer akışının elde edilmesi için konsol üzerinden port aktif edilir. GPS bağlantısı konsol kontrolü çıktısı Şekil 3.23'de gösterilmiştir.



Şekil 3.23. GPS bağlantısı konsol kontrolü çıktısı



Şekil 3.24. GPS modülünden alınan veri örnek çıktısı

Şekil 3.18 incelendiğinde, serial0 ttyAMA0 ile bağlantılıdır. Konsolu devre dışı bırakmak için CTRL+C tuş kombinasyonu kullanılması gerekmektedir.

GPS modülünün gerekli kütüphanelerinin kurulması ve ekrana çıktı veren test çıktı kodlarının eklenmesinin ardından. Şekil 3.24'te GPS modülünden alınan veri örnek çıktısı görülmektedir.

Navigasyon planlama sürecinin önemli ayağı olan GPS modülü bağlantısı gerçekleştirilmiştir.

IMU Modülü

IMU Modülü, GPS modülü ile senkron çalışan sensör grubudur. İçerisinde ivme, yön, barometrik basınç sensörü bulunmaktadır. IMU sensörü ile elde edilen sinyaller GPS koordinatları ile eş zamanlı olarak işaretlenir. GPS modülü tarafında gecikme söz konusu olduğu zaman IMU sensörü sinyal üretmeye devam edeceğinden yalın olarak zaman işaretlemesine tabi tutulur. Elde edilen veri kümeleri Genel Değişkenler yazılım

modülünde kuyrukta beklet yaklaşımı ile saklanarak işleme alındığında Kalman filtreleyicisi ve ardından motor kontrol komutlarıyla ilişkilendirilme sürecine dahil edilmektedir.

3.2.2.7. Sinyal veri filtreleyici işlemleri

IMU sensörünün ürettiği değerler kararsız yapıdadır. Birim zamanda oldukça fazla veri üretebilmeleriyle beraber birim zamanda oluşan değişkenlik uç noktalarda gerçekleşebilir. Bu doğrultuda elde edilen kararsız yapıdaki değerlerin önceki sonraki değerlerle harmanlanarak güvenilir sonuçlara getirilmeleri ile daha doğru ve belirsizliği azaltan yöntemler kullanılmaktadır.

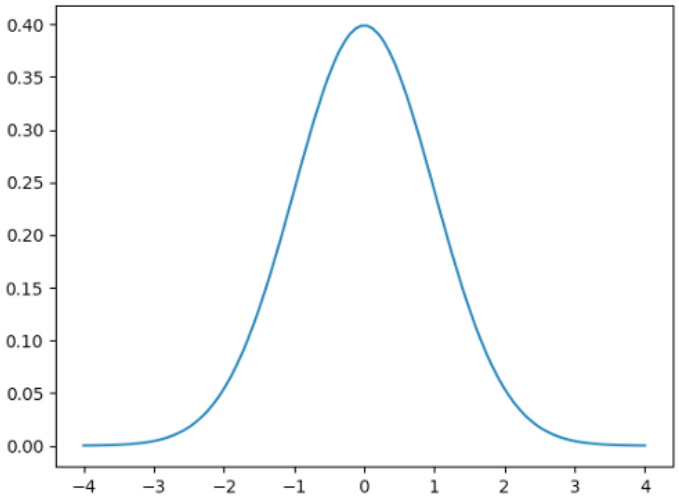
Birim zamanda elde edilen veri kümelerinin Kalman filtreleri ile doğru tahminlere erişilerek düzeltme işlemlerinin yapılmasında kullanılan matematiksel formüllerinin temel amacı düzeltme katmanları oluşturarak her çıktı aşamasında bir sonraki katman için girdi oluşturmaktadır. Bu durumda veri kümesinin doğrudan bir takım matematiksel işlemlerin çıktısıyla işleme devam etmek doğru çıktılar sağlamayacaktır.

Katmanlı yapı söz konusu olduğunda birim zamanda değişimin ele alınması kaçınılmazdır. Veri gruplaması örnek olarak ms seviyesinde ele alındığında gerçekleşmiş zamanın elde edilen verilerin gerçekleşecek zamana ait verilerin tahminlenmesini bir yanda tutarak, yeni elde edilen gerçek verilerin farkı kalman filtresi kazancını ortaya çıkarmaktadır. Bu doğrultuda elde edilen katsayı bizim için sürekli değişen ve her işlem döngüsünde güvenirliği artan bir referans değeri olmaktadır.

Kalman fitresi fonksiyonları, değişkenlerin tahmini olması bunun yanında ölçümünde bazı hatalara sebep veren gürültü sinyallerinin olduğunu kabul etmektedir.

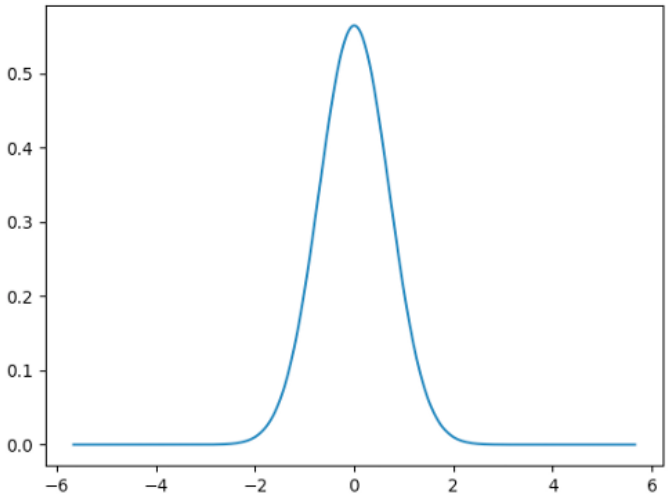
Filtreleme işlemlerinde birçok yaklaşım tekil ya da beraberinde kullanılabilir. Tüm hataların çözümlenmesinde temel matematiksel işlemlerin kullanımında sadece kesin koşullu olasılık tahminlerini verir. Yaklaşım olarak tercih edilen Gaus matematiksel işlemidir. (Won vd. 2009). Doğrusal gerçekleşen değişimlerde Gaus matematiksel işlemleri kullanılarak sonraki aşama olan Kalman filtreleme işlemlerine verimli girdiler sağlamak amacıyla kullanılmaktadır.

Gruplanabilir veri kümesinin dağılımı bir grafik olarak göstermek yerine, Kalman filtrelerindeki görev, bulunmaya çalışılan nesnenin pozisyonunun en iyi tahmini olarak bir mu ve sigma karesini sağlamaktır. Eğer Şekil 3.25.'deki gibi x bir dağılımın ortalamasına eşitse, ortalama ile girdi arasındaki fark yoktur. Dağılımın korunması için denklemde çıkış bilgisinin olup olmadığı kontrol edilmektedir.



Şekil 3.25. Örnek gauss dağılımı

Veri kümesinin daha az varyansa sahip olması durumunda, dar eğrili bir veri kümesinin Gauss ile çözümlenmesi tercih edilmektedir. Dar varyanslı veri kümeleri daha güvenilir sonuçlar ortaya çıkarmaktadır.



Şekil 3.26. Dar varyanslı örnek gauss dağılımı

Kalman filtresi, Gaus yöntemine göre tüm dağılımları temsil eder ve iki farklı süreci yineler. Bunlar ölçüm güncellemeleri ve hareket güncellemeleridir. Ölçüm güncellemeleri, belirli bir ifadeye sahip çıktılar ile önceki veri güncellemesini içerirken, hareket güncellemeleri kendisinden önceki ve sonraki değerlerin arasındaki dönüşümü içermektedir.

Ölçüm Güncellemesi, Bayes Kuralını kullanır. Başka bir araç yerelleştirildiğinde ve çok yüksek varyanslı (büyük belirsizlik) bir ön dağılıma sahip olunması

beklenmektedir. Algoritma, veri kümesi hakkında daha küçük bir varyansla çıktı üreten başka bir ölçüm alınması durumuyla veri kümesini birleştirir. Yeni bir Gaus hesaplaması ile ortalama alınarak, öncekinden daha yüksek bir tepe oluşumu ve daha dar varyans ile iki dağılım arasında yeni bir veri kümesi ya da nokta oluşması öngörülmektedir.

Aşağıdaki denklemlerde, nu ve r kare sırasıyla yeni gözlenen verilerin ortalaması ve varyansıdır. Önceki ortalamayı ve varyansı güncellemek için kullanılır.

$$\mu' = \frac{r^2\mu + \sigma^2\nu}{r^2 + \sigma^2} \tag{3.1}$$

Ağırlıkların önceki ve sonraki ortalamanın çeşitliliği olduğu eski ortalamaların ağırlıklı toplamı ile önceki ortalama yeniden atanmaktadır. Ortalama, ağırlıklandırma faktörlerinin toplamı ile sönümlendirilir.

$$\sigma^{2'} = \frac{1}{\frac{1}{r^2} + \frac{1}{\sigma^2}} \tag{3.2}$$

Varyansın güncellenmesi önceki varyansları kullanarak mümkün olur. Aynı varyansa ancak farklı ortalamalara sahip iki Gauss dağılımı olsaydı, ortaya çıkan Gauss dağılımının ortada bir yerde olması ancak varyansın orijinal varyansın yarısı kadar olması problemdir.

Ölçüm Tek boyutlu Kalman filtresi için ortalama ve varyans güncelleme yaklaşımında ise hareket güncellemesi yöntemini ortaya çıkarır.

Öncesi için, mobil robotun bir pozisyonda başlanarak referans alınır. Ancak hareket ettiğinde pozisyon değiştirilmektedir. Hareketin kendisinin kendi Gauss sonucu dağılımı ve kayıp veri belirsizliği vardır. Hesaplama ortalama hareket komutunu ilave eden ve ilk belirsizlik üzerinde belirsizliği artıran sonuca ulaşılmaktadır. Pozisyonun değiştikçe belirsizlik kümesinde bilgi biriktirilmektedir.

$$\mu' = \mu + u \tag{3.3}$$

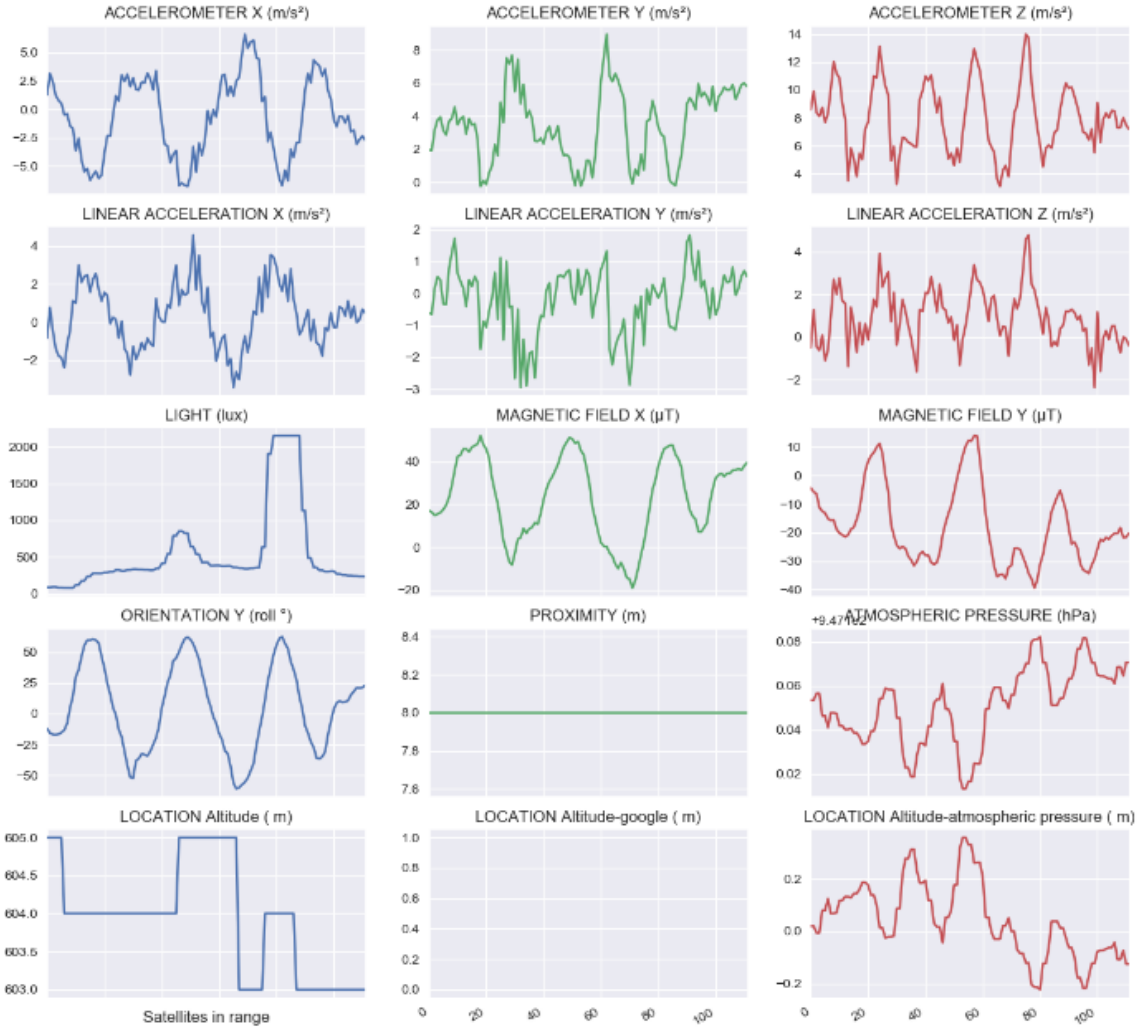
Ortalama, önceki ortalama alınarak ve u değişkeni ile ifade edilen hareketin hareketi ilave edilerek tekrardan hesaplanır.

$$\sigma^{2'} = \sigma^2 + r^2 \tag{3.4}$$

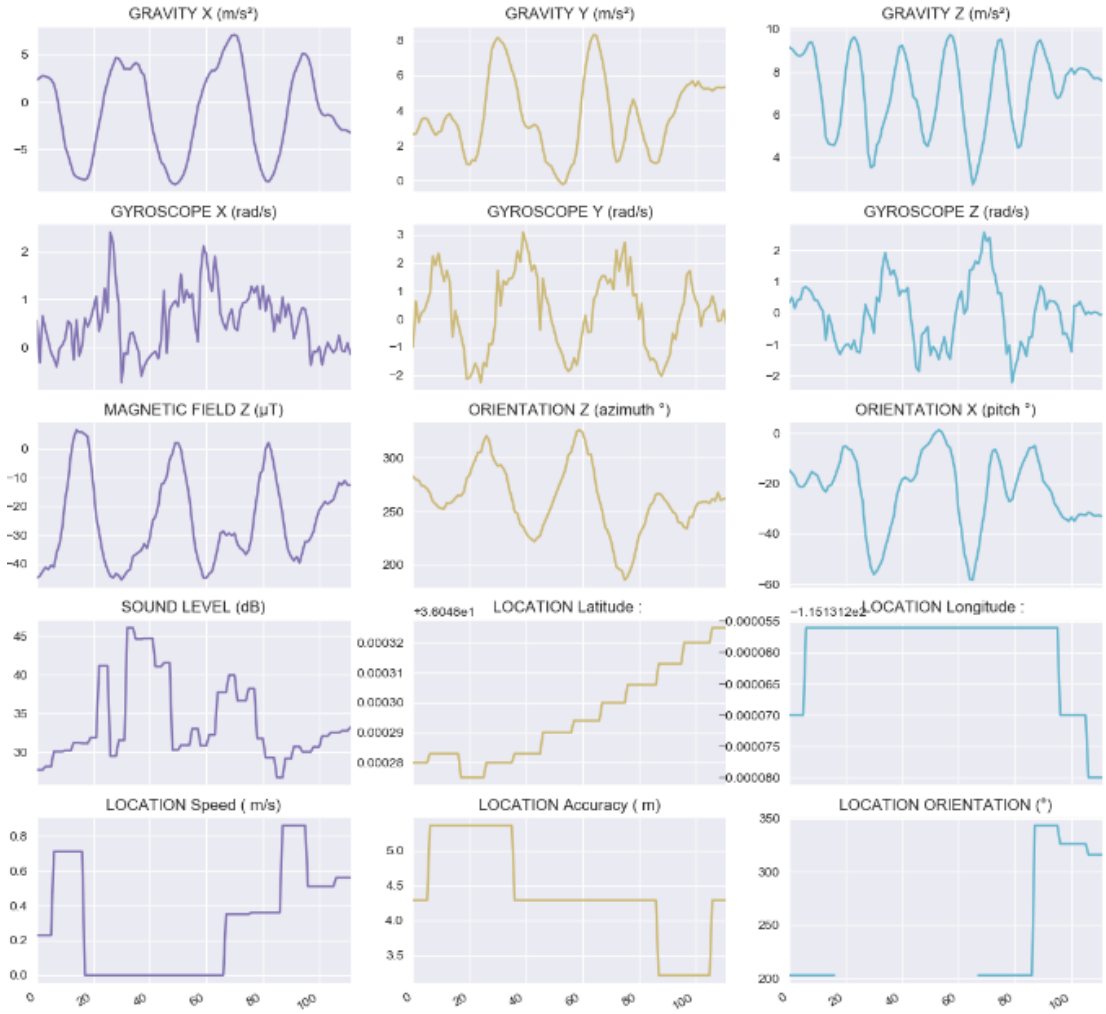
Varyans, eski sigma alınarak ve üzerine Gauss hareketinin varyansı eklenerek güncellenir. Bu işlemlerin oluşan dalga boylarını sönümlendirmede kullanılan temel yaklaşımlardır. IMU sensörlerinin yapısı gereği ve titreşimlerden kaynaklı hata sinyallerini sönümleme teknikleri çok çeşitlilik göstermekle bir adım öteye taşıyarak fourier analizi yöntem olarak tercih edilmiştir (Foxlin, 1996).

IMU, Mems sensörleri ivmeölçer, jiroskop, barometre ve manyetometre içerir. Bu sensörler birlikte, sensörün uzaydaki ivmesini, dönüş hızını ve yönünü ölçen bir

eylemsizlik ölçüm birimi (IMU) oluşturur. Sensörlerin veri akışını içeren örnek Şekil 3.27-3.28'de sunulmuştur.



Şekil 3.27. IMU sensörlerinden elde edilen veri akış grafiği -1



Şekil 3.28. IMU sensörlerinden elde edilen veri akış grafiği -2

IMU sensörlerin elde edilen veriler kayıt altına alınarak dönüştürme işlemine alınmıştır. IMU sensörleri, yerçekimi, doğrusal hızlanma ve toplam hızlanma olmak üzere 3 tür hızlanma çıkışı vermektedir ve bunların her birisi için 3 eksen (x, y, z) ölçümü gerçekleştirilmiştir.

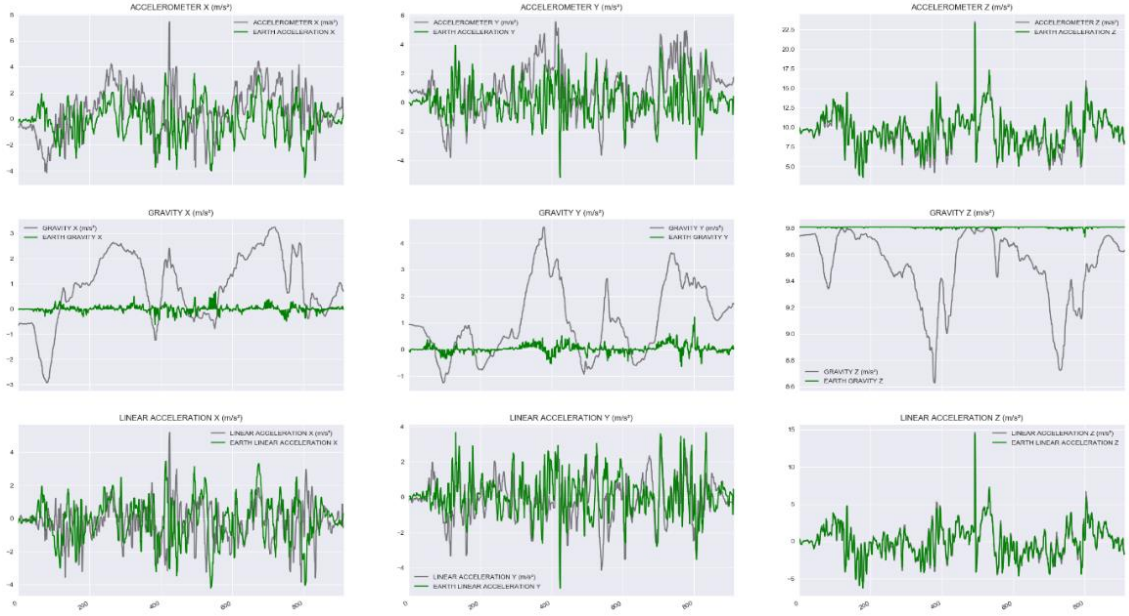
Yerçekimi ivmesi, İMU sensörü üzerindeki yerçekimi kuvvetinden kaynaklanmaktadır. Doğrusal hızlanma yalnızca dış kuvvetlerden kaynaklanmaktadır. Toplam ivme, yerçekimi ivmesi ve doğrusal ivmenin toplamına bağlıdır. Hem ivme hem de yerçekimi için Z eksenlerinde -9,81 m/s^2 ve doğrusal hızlanma için sıfır olarak okunmuştur. İvme ölçeri ivmeyi kendisine (gövde çerçevesi) göre ölçer ve gövde çerçevesi ivmelerini ataletli, dönmeyen bir çerçeveye (Yer kabuğu referanslı) dönüştürülmesi gerekir. Sensörler herhangi bir yönde tutmayı ve sensörü dünya çevresindeki yörüngesini hesaplamak için doğru ivme vektörlerini ölçmeyi mümkün kılmaktadır. İvme değerleri otomatik olarak dünya çerçevesine dönüştürme seçeneğine sahiptir (El-Kebir ve Hamza, 2021).

Dönüşüm, gösterildiği gibi matris cebiri ile yapılır, burada [X, Y, Z], gövde çerçevesi doğrusal ivmeleridir ve Rz, Ry, Rx, [X, X'i döndürmek için her eksen için dönme matrisleridir. Y, Z] üzerinden dünyanın [x, y, z] eksenlerine. Euler açıları (ψ, θ, φ) x, y, z, eksenler veya adım, yuvarlanma ve sapma ile ilgili açılara karşılık gelir. Yazılım sürecine dahil edilen hesaplama ile anlık olarak gelen veriler hesaplanabilinmektedir.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_z(\psi)R_y(\theta)R_x(\phi)\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.5}$$

Dönüşüm işleminin sonucunda Şekil 29'da ki grafiklerde görüldüğü üzere sensör gösterim çizgisi (gri) – yer çekimi gösterim çizgisi (yeşil) olarak verildi. Yer çekimi Z değişim grafiği neredeyse sabit 9,81 m/s^2 ve yer çekimi X, Y'nin sıfıra yakın gösterdiğine dikkat edildiğinde ölçümün sağlıklı gerçekleştiği söylenebilir, yer çekimi sadece Z ekseninde hareket etmektedir (El-Kebir ve Hamza, 2021).

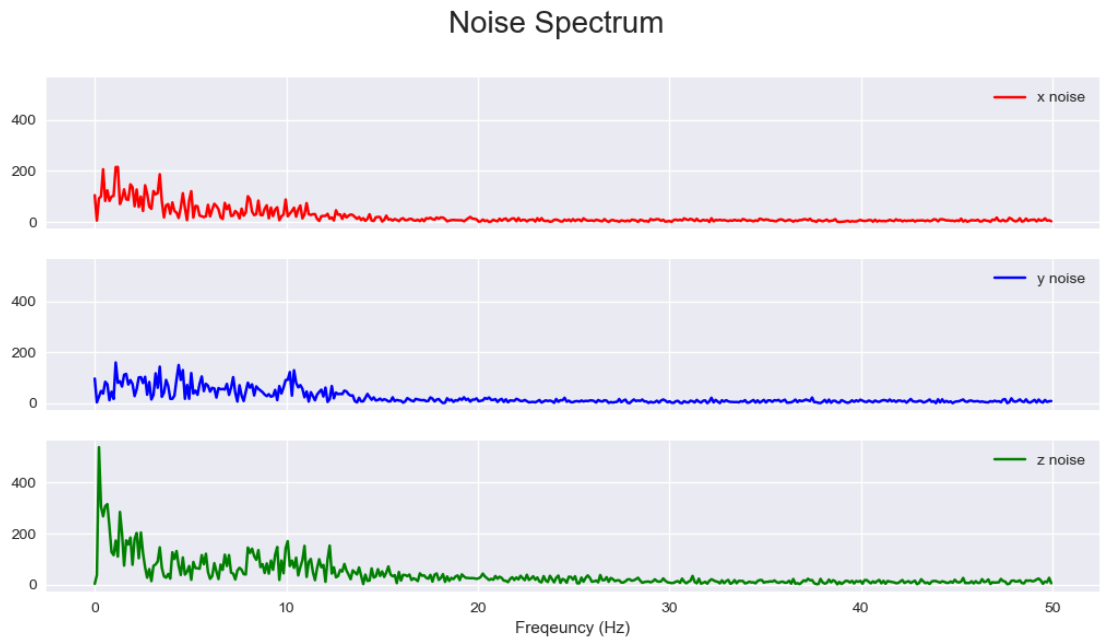
Kulanılan dönüşüm işleminin matriks hesaplaması sonucu oluşan vektörel ivme, yerçekimi ve doğrusal ivme grafikleri Şekil 3.29'da verilmiştir.



Şekil 3.29. Kulanılan dönüşüm işleminin matriks hesaplaması sonucu oluşan vektörel ivme, yerçekimi ve doğrusal ivme grafikleri

Bundan sonraki adımda ise ivme ölçümlerinin Fourier analizi ile işleme alınması gerekmektedir. Sensör, sarsıntılardan ve kendi doğal yapısından dolayı çok gürültü sinyali üretmektedir. Bu gürültüyü sönümleyebilmek için Fourier analizi yöntem olarak kullanılmıştır.

Oluşan gürültü sinyalleri mevcut sinyal eğri karakteristiği dışında kalan noktaları varsayabiliriz. Ancak sinyal eğrisinin optimum nokta kümesi dışında kalan noktaların göz ardı edilebilmesi için kalman kazancı katsayısı ile önceki verilerin ışığında elde edilen yeni tahminlerin çok dışında kalan gerçekleşen değerlerin farkındaki olağandışılık bizi gürültü sinyali tanımlamasına götürmektedir. IMU sensöründen elde edilen ivme ölçümü ile x,y ve z eksenlerinde elde edilen hamverinin kalman filtreleme işlemi sonucundaki gürültü noktalarında ki sönümleme sonucu verilmiştir. Stabil bir analog sinyal elde edilerek gürültü minimize edilmiştir (Foxlin, 1996).



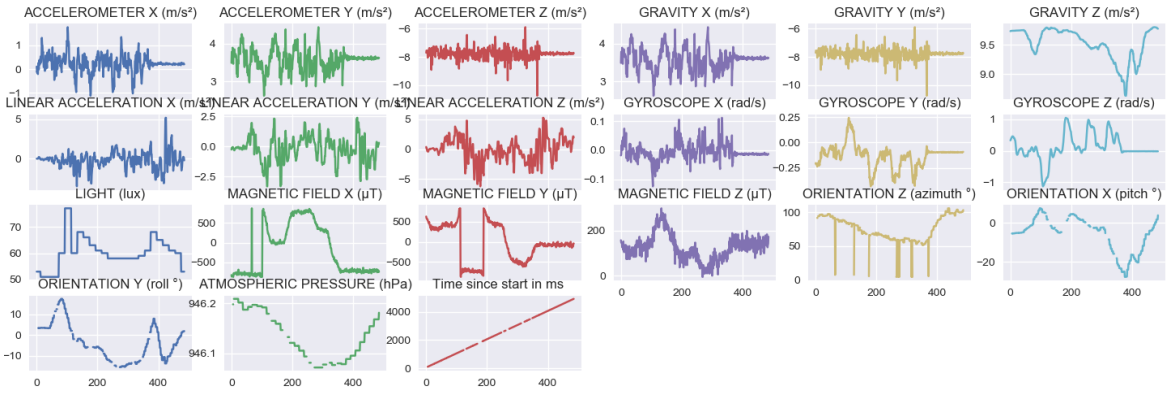
Şekil 3.30. İvme ölçümü sonucu oluşan gürültü spectrumu

4. BULGULAR VE TARTIŞMA

Mobil robot çatısı altında çalışmanın IMU ve GPS sensörlerinden elde edilmiş konum, ivme, yön, pusula bilgilerinin kullanılarak gerçekleştirilen gürültü önleme ve kalman filtreleyicisi işlemleri sonucunda elde edilen sinyal, konum, hız, sapma açıları işlemlerini kapsamaktadır. Çalışmada, IMU sensörleri kritik önem taşımaktadır. Çalışmanın amacı, GPS verilerinden oluşan hata sinyallerinin IMU sensörü yardımı ile düzeltilmesi, GPS verilerinden kaynaklanan hataların bertaraf edilerek IMU destekli verilerin GPS sensöründen gelen konumları daha doğru şekilde elde edilmesidir. Deneysel süreçte 6 DOF IMU sensörü, 10 DOF IMU sensörü, 10 DOF IMU sensörü ile GPS sensörü beraber konumlandırılarak mobil tarım robotu üzerinden konum, ivme, yön, pusula bilgileri toplanmıştır. Bu bilgiler toplanırken euler sapma açısı yön düzelmesinde, doğrusal hızlanma değerleri ve manyetik alan bilgileri alınarak konumlandırma işlemlerinde ek olarak doğrulama süreçleri tamamlanmıştır.

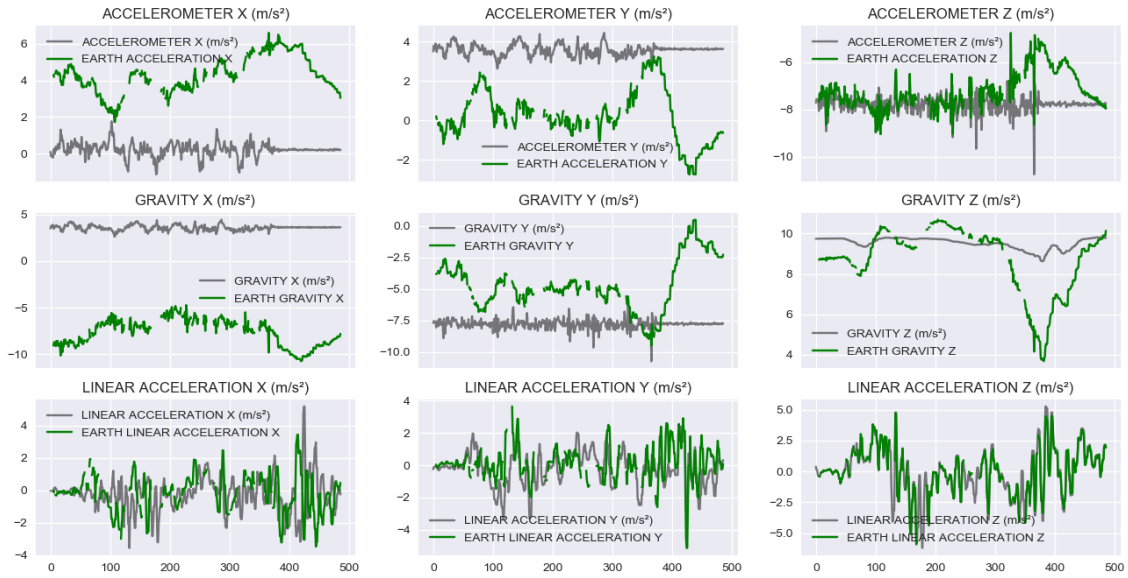
4.1. IMU ve GPS sensor fusion

6 DOF IMU sensörünün üzerinden ivme ve yön bilgileri elde edilmiştir. X, Y ve Z eksenleri üzerinde elde edilen verilerin sonucunda sinyal kesintileri gözlemlenmiştir. Yapılan log incelemeleri sonucunda sinyal kesintilerinin nedeni araştırılmıştır. Bunun sonucunda sistemin birim zamanda çalışma döngüsü karşısında 6 DOF IMU sensörü uyum sağlayamamıştır. Şekil 4.1'de 6 DOF IMU Sensöründen alınan 5 s'lik sinyal değerlerinin çıktısı verilmiştir.



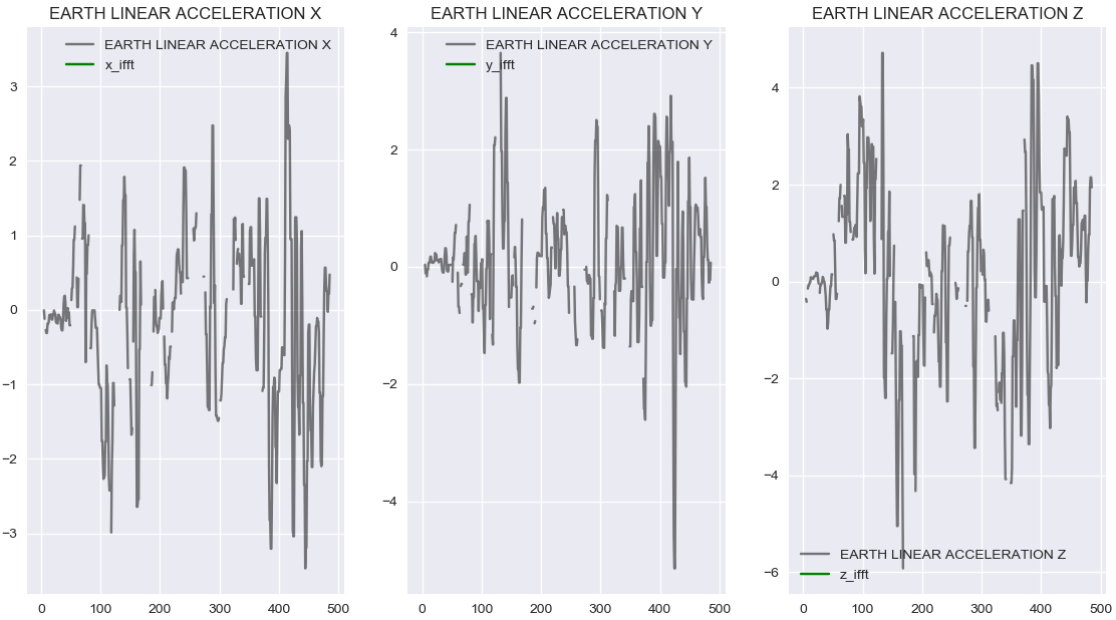
Şekil 4.1. 6 DOF IMU Sensörü ivme, yön değerlerinin çıktısı

6 Dof IMU sensörü üzerinde elde edilen çıktıların sonucunda yapılan filtreleme ve sinyal düzeltme işlemlerinin sonucunda da eksiklikler görülmektedir. Bu durumda GPS verileri ile senkron edilmesi sonucunda konumlandırma işlemlerinde eksik parametrelerin değerlendirilmesi ile hatalara neden olmaktadır. 6 DOF IMU sensörünün sinyal doğrulama işlemlerinin çıktısı verilmiştir (Şekil 4.2).



Şekil 4.2. 6 DOF IMU Sensörü sinyal doğrulama işlemlerinin sonucu

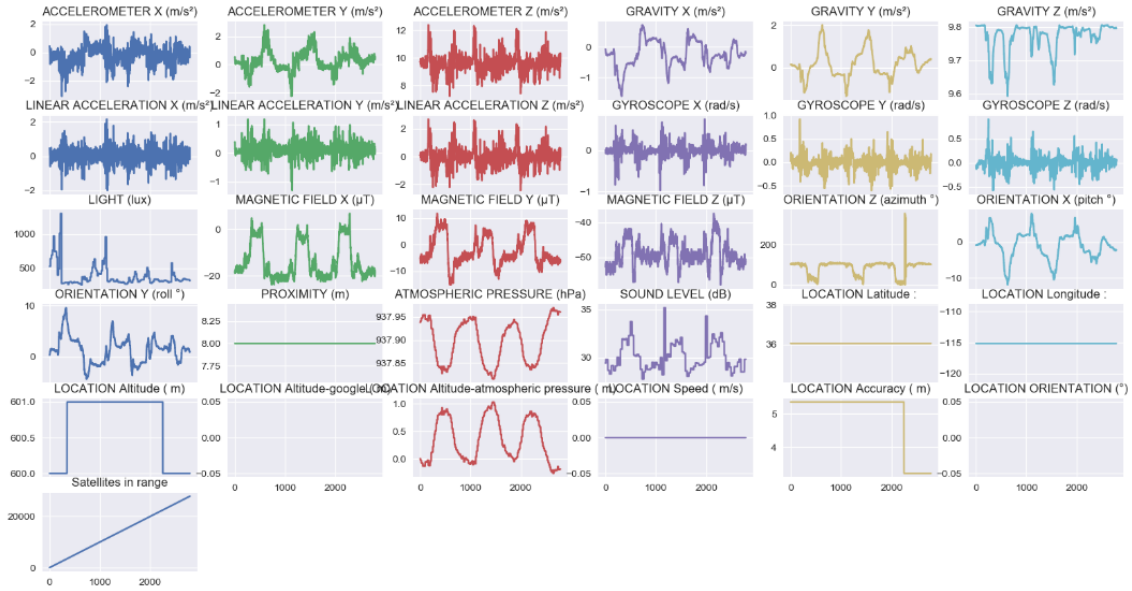
6 DOF IMU sensörü üzerinden elde edilen sabit ivme ile hareket eden mobil robotun üzerinden doğrusal hızlanma sonucunda da eksik bilgiler elde edilmektedir. 6 DOF IMU sensörü üzerinden elde edilen verilerin doğrusal hızlanma işlemleri sonucunda ortaya çıkan verilerin çıktısı Şekil 4.3'te verilmiştir.



Şekil 4.3. 6 DOF IMU Sensörü doğrusal hızlanma işlemlerinin sonucu

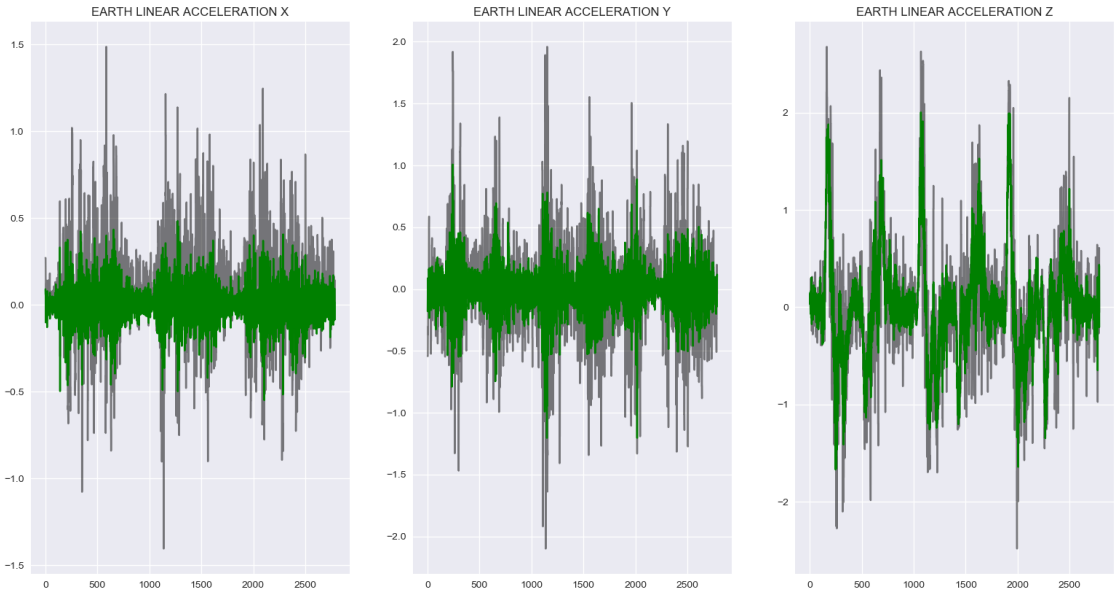
10 DOF IMU sensörü üzerinden elde edilen veriler mobil robot hareketleri ile elde edilmiştir. 6 DOF IMU sensörü ile karşılaştırıldığında üzerinde basınç ve 3 eksen

üzerinde manyetik alan bilgilerinin alınabilmesi söz konusudur. 10 DOF IMU sensörü birim zamanda veri elde etme miktarı daha fazladır. Şekil 4.4'te 10 DOF IMU üzerinden elde edilen 5 s'lik ham veri sinyalleri görülmektedir.



Şekil 4.4. 10 DOF IMU İvme ve yön bilgilerinin çıktısı

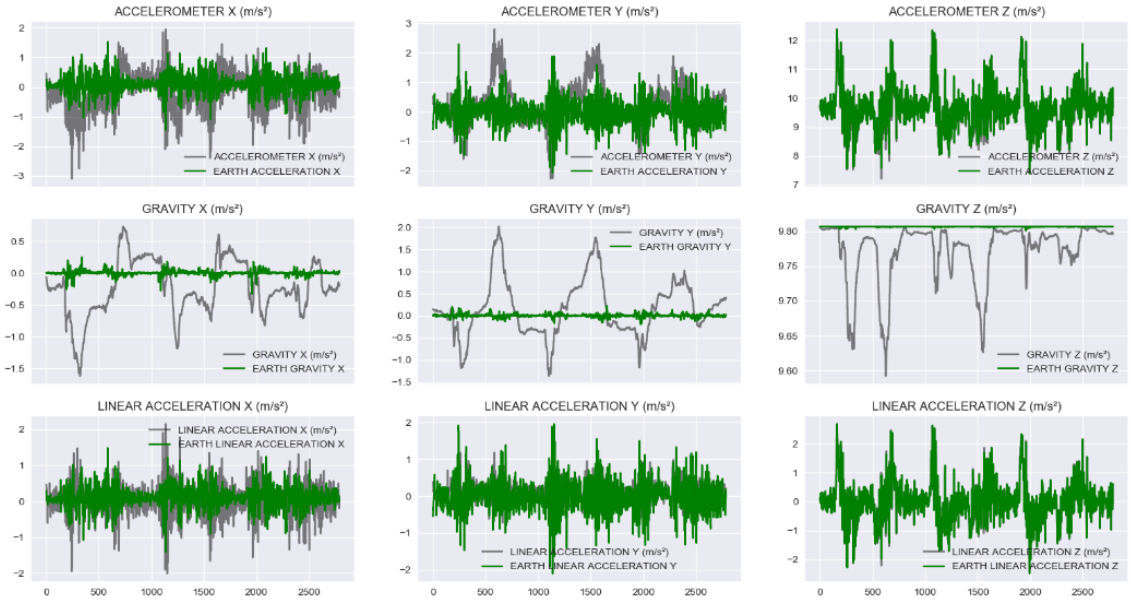
10 DOF IMU sensöründen elde edilen ivme, yön bilgilerinin sönümlendirilmesi işlemleri çıktısı değerlendirildiğinde mobil robot hareket halinde sabit ivme ile giderken doğrusal hız çıktısı incelendiğinde kararlı sinyaller iletildiği gözlemlenmiştir. Şekil 4.5'te doğrusal hız grafikleri gözlemlenmşitir.



Şekil 4.5. 10 DOF IMU Sensörü 3 eksenli doğrusal hız sintal çıktısı

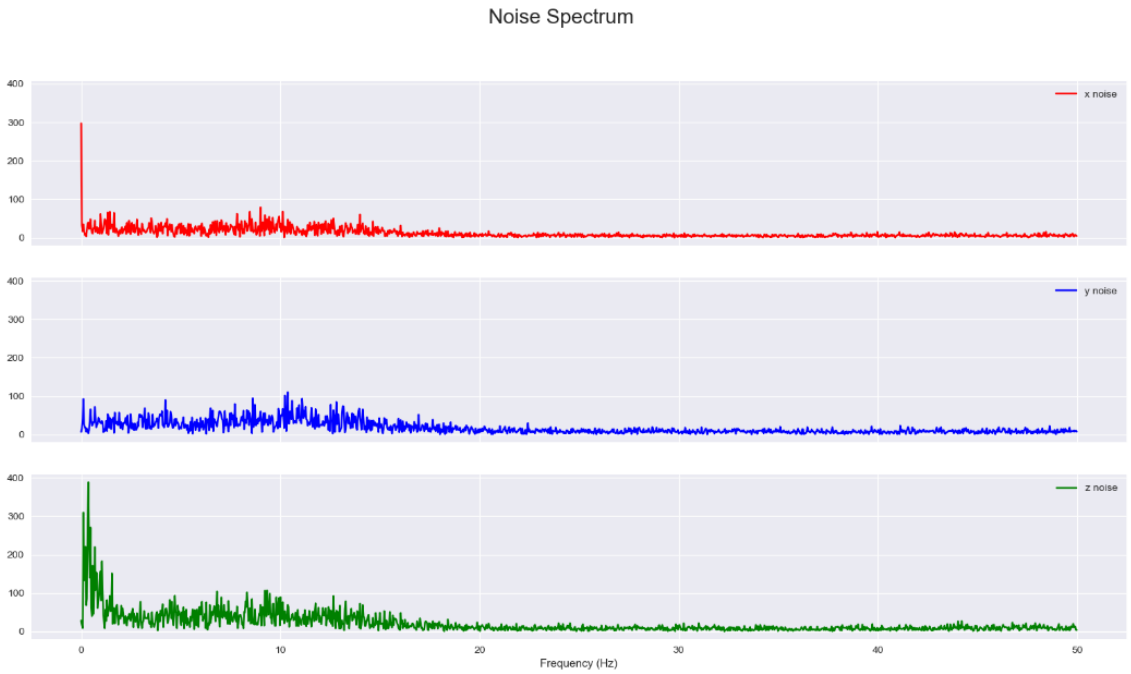
Birim zamanda hareketli halde 10 DOF IMU sensöründen elde edilen 3 eksenli

yön, ivme ve manyetik alan değerleri ele alındığında sönümleme işlemleri uygulanmış ve ilgili çıktı Şekil 4.6'de verilmiştir.



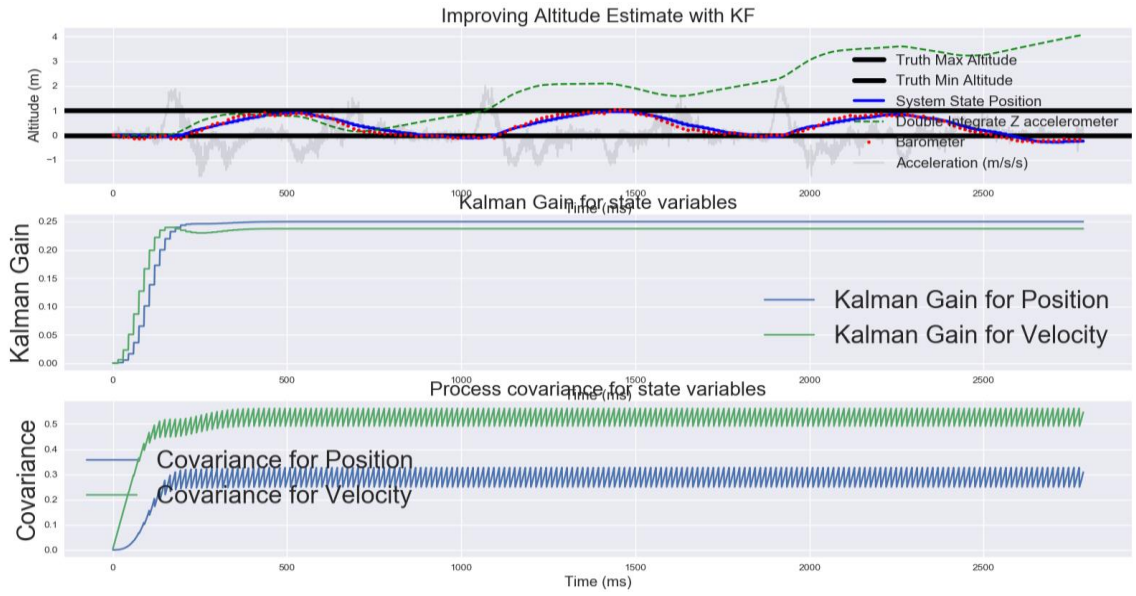
Şekil 4.6. 10 DOF IMU Sensörü 3 eksenli ivme, yön, manyetik alan çıktıları

10 DOF IMU sensörü ham veri çıktılarının gürültü sinyallerinin sönümlendirilmesinde çıktı grafiği Şekil 4.7'de verilmiştir.

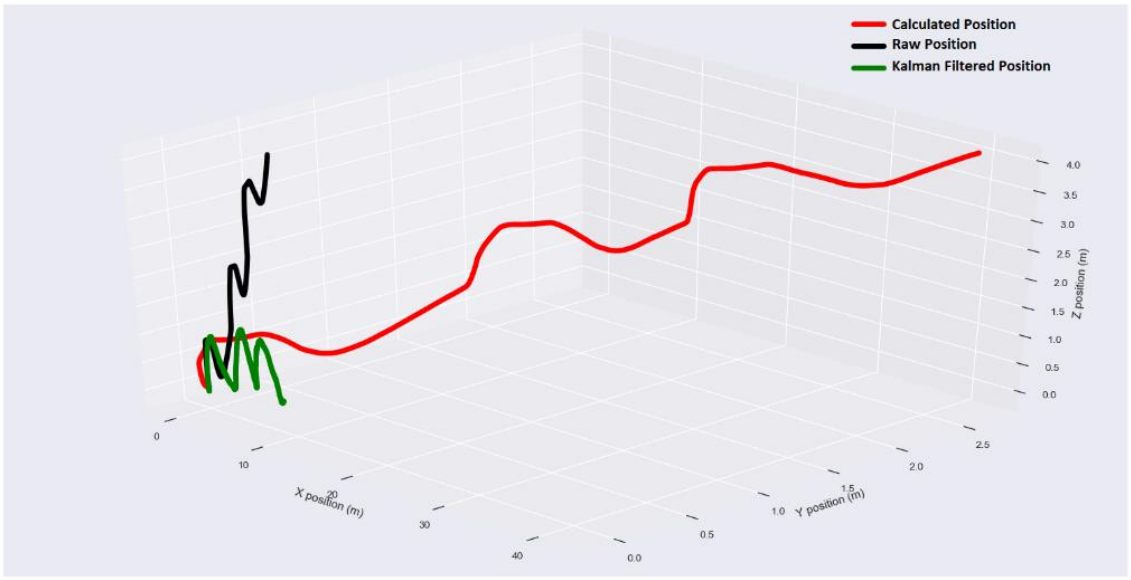


Şekil 4.7. Gürültü sinyallerinin sönümlenme grafiği

10 DOF IMU sensöründen elde edilen ham verinin sönümlendirilmesinin ardından elde edilen veriler, Kalman filtreleme işlemlerine alınmıştır. Elde edilen Kalman filtreleme çıktıları, sanal koordinat sistemi üzerinde konumlandırılmıştır. Kalman işlemleri sonucunda ivme üerinde elde edilen birim zamanda oluşan hız bilgisi ve konum değerlendirme grafiği verilmiştir (Şekil 4.8).

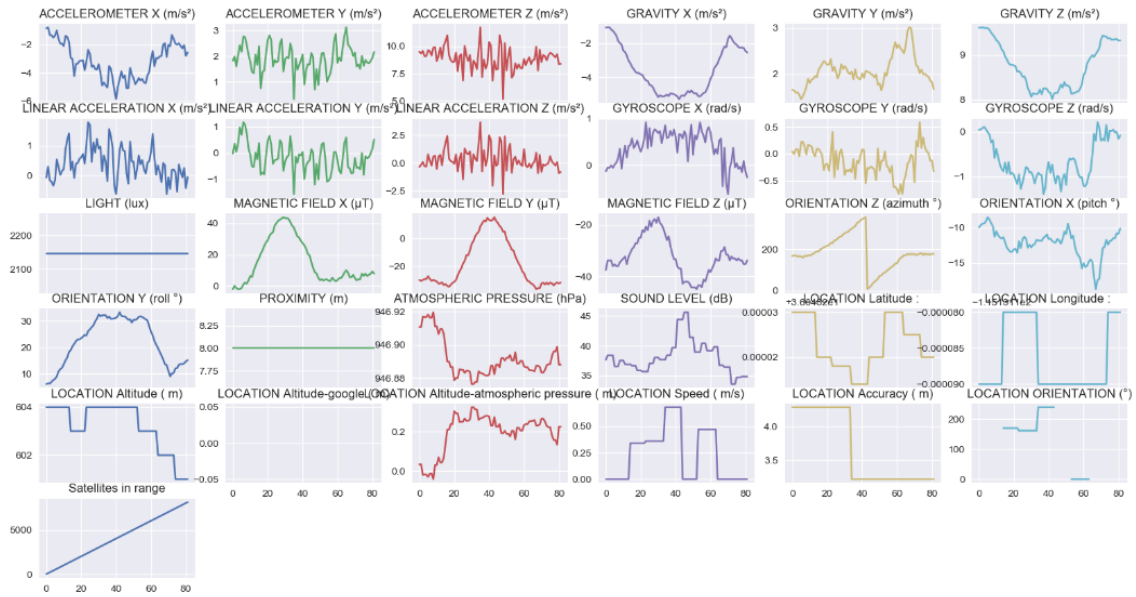


Şekil 4.8. 10 DOF IMU Sensörü kalman işlemleri hız ve pozisyon çıktısı



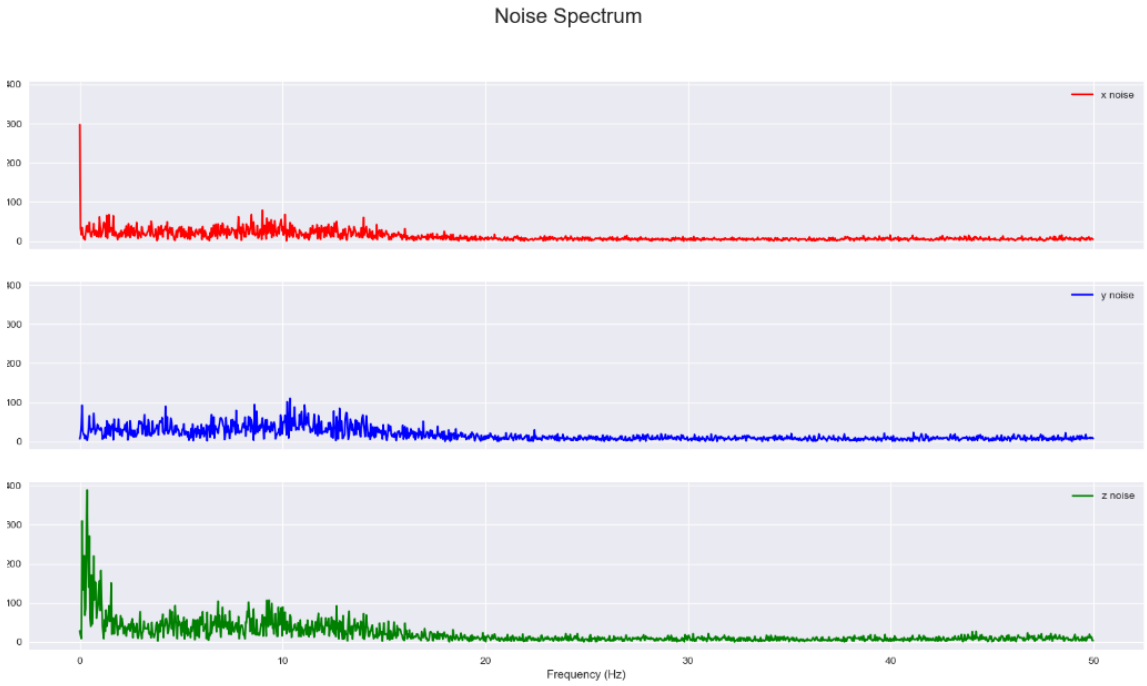
Şekil 4.9. 10 DOF IMU Sensörü 3 boyutlu koordinat sistemi üzerinde pozisyon çıktısı

IMU sensöründen elde edilen veriler ile konumlandırma işlemi gerçekleştirildiğinde konum hatalarının ortaya çıktığı belirlenmiştir. Düz hat üzerinde giderken sapmalar meydana gelmektedir. IMU sensöründen elde edilen konum noktaları, GPS ile gelen konum noktaları ile beraber kalman filtrelemesinde dahil edildiğinde, yalın IMU sensörüne göre doğrusal hareket, manevra dönüşlerinde daha doğru rota çizildiği görülmektedir. Şekil 4.10'da dönüş hareketi ile gerçekleştirilen ham IMU çıktıları verilmiştir.



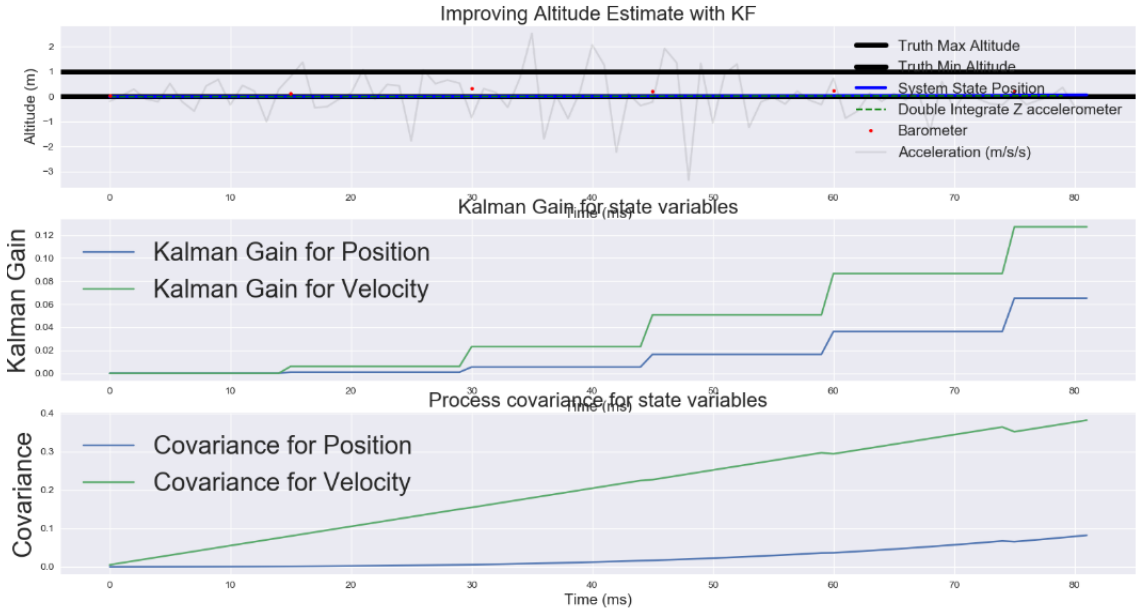
Şekil 4.10. 10 DOF IMU Sensörü dönüş hareketi ile elde edilen ham çıktı

Elde edilen ham IMU verilerinin çıktısının gürültü sinyallerinin sönümlendirilmesi işlemi sonu Şekil 4.11'de verilmiştir.



Şekil 4.11. 10 DOF IMU Sensörü gürültü sinyallerinin sönümlenmesi grafiği

IMU verilerinden elde edilen 3 eksenli ivme bilgilerini ayrıştırıldığında birim zamanda elde edilen doğrusal hız belirlenmiştir. Kalman filtreleme işlemlerine konum bilgileride eklendiğinde daha kararlı çıktı değişimi görülmektedir. Şekil 4.12'de IMU ve GPS çıktılarının kullanmında konumlama işlem sonucu verilmiştir.



Şekil 4.12. IMU ve GPS verilerinin kullanılarak elde edilen hız ve konum çıktısı

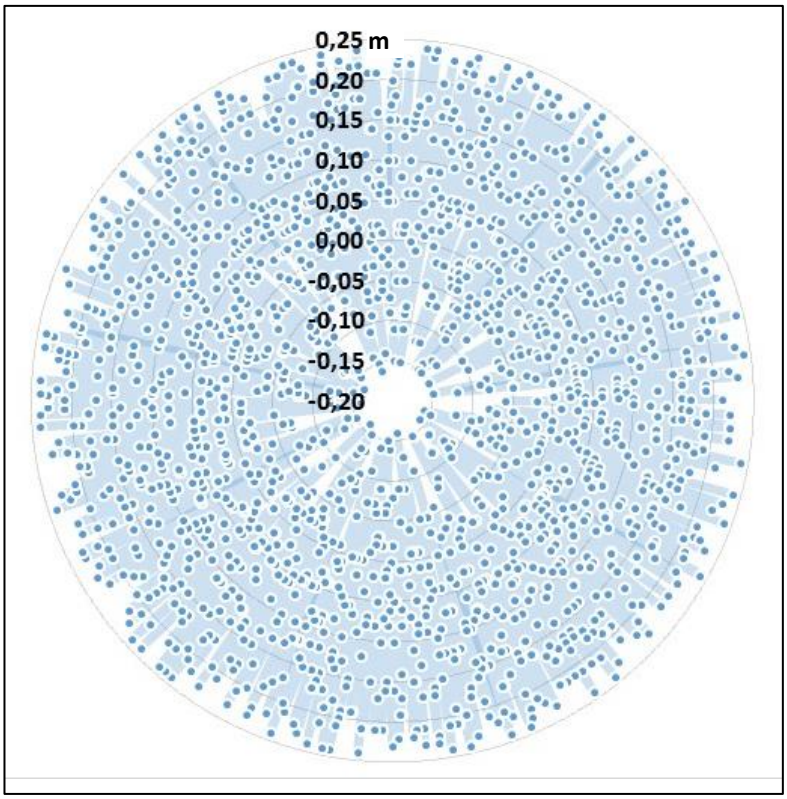
Elde edilen veriler doğrultusunda 10 DOF IMU ve GPS sensörü Kalman filtrelemesi işlemlerinde diğerlerine göre daha kararlı konum bilgileri elde edilmiştir. Daha sonra yapılan tarla denemleri uygulamalarında 10 DOF IMU sensörü kullanılmıştır.

4.2. Mobil robot tarla denemeleri

Mobil robot üzerindeki IMU ve GPS sensörlerinin hata kontrol mekanizması, güç besleme karasızlıklarının giderilmesi sağlanarak tarla demelerine hazır hale getirilmiştir. Tarla denemeleri Akdeniz Üniversitesi, Ziraat Fakültesi Tarım Makinaları ve Teknolojileri Mühendisliği Bölümü, Araştırma ve Uygulama alanında yapılmıştır. Mobil Robot, rota planlaması oluşturularak 4 km/h ilerleme hızı ile hareket ettirilmiştir. Robot üzerindeki Sensor Fusion sisteminde IMU ve GPS verileri anlık 20 Hz olarak kaydedilmiştir. Sensor Fusion sisteminden elde konum bilgilerinin doğrulanması için, mobil robot üzerine monte edilen Magellan Promark 500 RTK GPS eş zamanlı olarak konum bilgileri kaydedilmiştir. Tarla denemleri sırasında mobil robot üzerindeki Sensor Fusion sisteminden ve RTK GPS'ten elde edilen konum bilgileri Şekil 4.13'te verilmiştir.

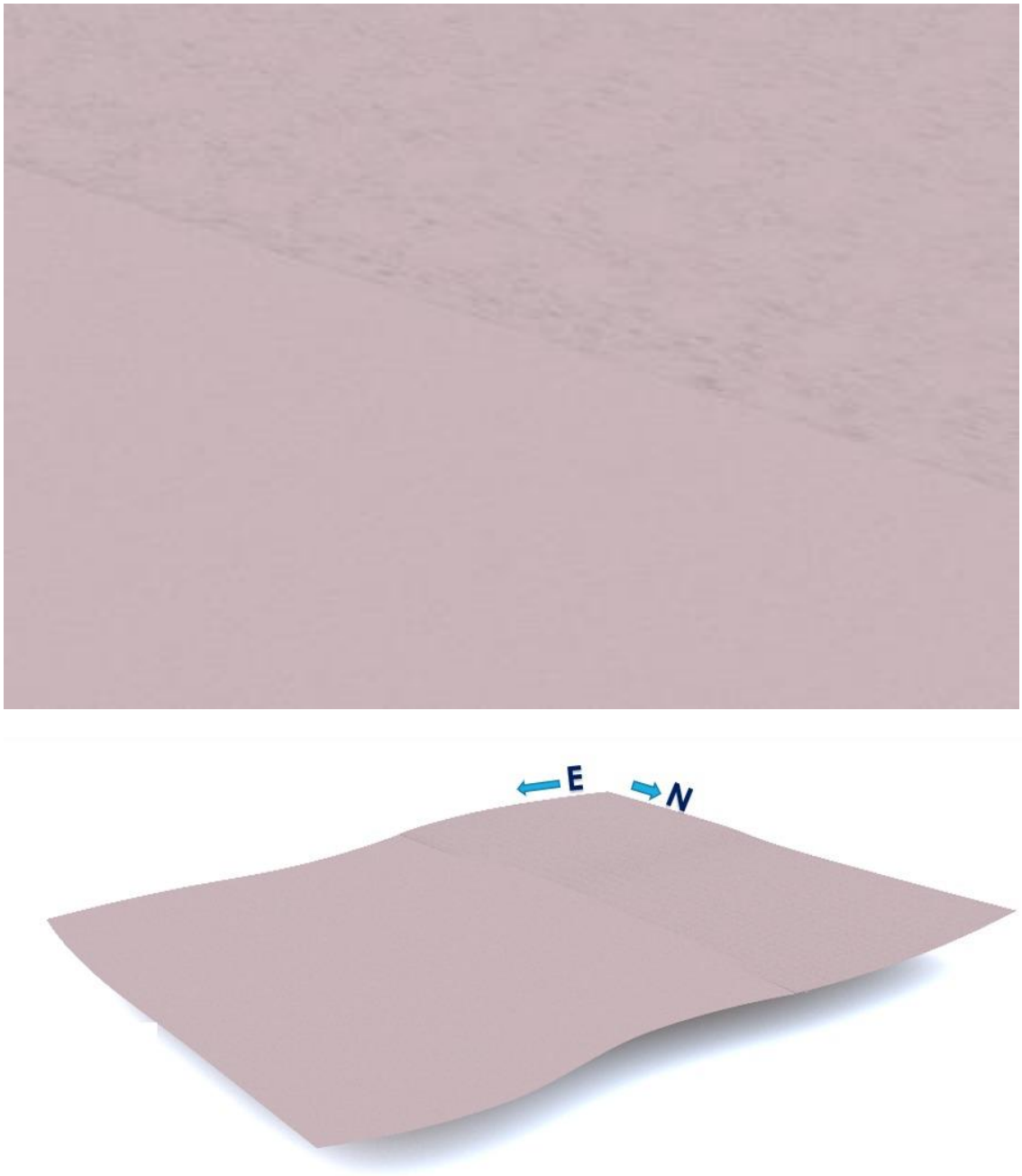
Şekil 4.13. Arazi üzerinde mobil robotun hareket izleri

Mobil robot üzerinde geliştirilen Sensor Fusion sisteminden ve RTK GPS'ten elde edilen konum bilgileri değerlendirildiğinde ortalama olarak 0,11 m sapma değeri belirlenmiştir. Her iki sistemden elde edilen konum bilgileri sapma değerlerine göre RMSE (Ortalama Karekök Sapması) 0,13 m olarak belirlenmiştir. Konum bilgileri karşılaştırıldığında RTK GPS'e göre sapma değerleri dağılım grafiği Şekil 4.14.'de verilmiştir. Dağılım grafiğine göre geliştirilen sistemin çevresel etkilerden az etkilendiği söylenebilir.



Şekil 4.14. Konum sapma değerleri dağılım grafiği

Arazi üzerinde serbest gezinimi gerçekleştirilen mobil robot GPS ve IMU sensörleri üzerinden elde edilen verilerin sanal koordinat sistemine aktarılarak nokta bulutu oluşturulmuştur. Noktaların birleştirilmesiyle python veri görselleştirme komutları çalıştırılarak tarla yüzeyi elde edilmiştir. Elde edilen tarla yüzeyi Şekil 4.15'te verilmiştir.



Şekil 4.15. Çalışma alanı 3 boyutlu yüzey haritası

5. SONUÇLAR

Otonom olarak rota planlaması yapabilen ve motor hareketlerini kullanıcı tarafından verilmiş hedef koordinat noktalarına göre hareket edebilmesi üzerine tasarlanmıştır. Hassas tarım faaliyetlerine uygun GPS konum verileri ile hedef koordinat değerlerini değerlendirerek hareket anında IMU sensöründen elde edilen veriyi kullanarak hassas konumlandırma işlemi hedeflenmiştir. Modüler yapıda düşük maliyetli donanımsal birimleri ile prototiplemenin daha kolay hale getirilmesi sağlanmıştır. Mobil robot üzerinde hareket halinde elde edilen konum verileri, ivme, yön, açı, manyetik alan bilgileri elde edilerek gerçek hat üzerinde hareket süreci değerlendirilmiş olup farklar ortaya çıkarılmıştır. Elde edilen sonuçlar üzerinde ham veriler tekardan filtreleme ve sönümleme işlemlerine tabi tutulmuş, konum verilerinden IMU verileri ayrılarak tekrardan analizleri gerçekleştirilmiş, farklı IMU teknolojileri üzerinde değerlendirmelerde bulunulmuştur. Yazılımsal olarak çoklu görev mimarisi konum doğrulama ve robota aynı anda birden fazla görev verme konusunda (veri toplama, analiz etme, motor hareketi gerçekleştirme vb.) avantaj sağlamıştır.

- Geliştirilmiş otonom mobil robotun sahip olduğu konumlandırma yöntemi GPS ve IMU sensörü destekli gerçekleştirilmiştir.
- Kumanda yardımıyla hareket edebilecek şekilde tasarlanmıştır.
- GPS konumlarının elde edilmesinde haberleştiği uydu sayısı elde edilmiş olup sistem düzeltme protokolleri oluşturarak konum doğrulamada ek yöntemler geliştirilebilir.
- Mobil robot sahip olduğu haberleşme hizmeti sayesinde farklı teknolojilerle haberleşebilir. Bunun yanında birden fazla mobil robot ile tarımsal faaliyetleri gerçekleştirebilir alt yapıya sahiptir.
- Tasarlanmış mobil robot bulunduğu noktada dönebilme özelliğine sahip olup mekanik olarak dairesel dönüş gerçekleştirme yetisine sahiptir.
- Mobil robot üzerinde elektronik donanımsal yapısı modüler tasarlanmış olup bakım sürecinde kolaylıklar sağlamaktadır.
- Mobil robot üzerinde geliştirilen Sensor Fusion sisteminden ve RTK GPS'ten elde edilen konum bilgileri değerlendirildiğinde ortalama olarak 0,11 m sapma değeri belirlenmiştir.
- Her iki sistemden elde edilen konum bilgileri sapma değerlerine göre RMSE (Ortalama Karekök Sapması) 0,13 m olarak belirlenmiştir.
- Yapılan arazi çalışmalarında dönüş işlemlerinde sapmalar görülmüş olunup mobil robotun dönüş işlemlerinde kullandığı algoritmaların ayrı modül olarak değerlendirilmesi gerektiği ortaya çıkmıştır.
- Konum bilgileri karşılaştırıldığında RTK GPS bilgileri dikkate alındığında sapma değerleri dağılım grafiğine göre geliştirilen sistemin çevresel etkilerden az etkilendiği söylenebilir.
- Geliştirilen platform, ekim öncesi sırt oluşturma ve ekim işlemleri dışında elde edilen 0,11 m sapma değeri ile gübreleme, ilaçlama, hasat gibi tarımsal üretim faaliyetlerinde kullanılabilecektir.

Gelecekte yapılabilecek çalışmalar için öneriler aşağıda verilmiştir.

- Sistemin doğru konum bilgisi elde edebilmesi için kararlı yapıda çalışması oldukça önemlidir. Sistemin kararlı yapıda çalışabilmesi için, sıcaklık kontrolü, kaliteli güç beslemesi, yazılımsal hata kontrol mekanizması, ms düzeyinde doğru süreç yönetme algoritmaları etkili faktörlerdir.
- Dönüş işlemlerini mobil robotun daha kararlı gerçekleştirebilmesi için, anlık olarak karar verecek kontrol mekanizmasının kurularak oluşabilecek sapmaları, dönüş anında kalman filtresi ile tahminleyen ayrı bir modülün tasarlanması, daha etkin dönüş hareketleri sağlayacaktır.
- Geliştirilecek olan benzer sistemlerde daha hassas GPS ve IMU modülleri ile daha doğru konum bilgisi sağlanabilinir.
- Sistem içerisinde kullanılan Raspberry Pi modülü ihtiyaç duyduğu gerilim miktarı konusunda tolerans seviyesi düşük olması nedeniyle endüstriyel bilgisayar kullanımı sistemin daha kararlı yapıda çalışmasını sağlayacaktır.

6. KAYNAKLAR

Arabboevna, M. A. ve Mirzakarimova, G. M. 2021. The importance of installation of base gps stations in permanent activity in Fergana region. *Asian Journal of Multidimensional Research*, 10(9), pp. 483-488.

Castellanos, G., Deruyck, M., Martens, L. ve Joseph, W. 2020. System assessment of WUSN using NB-IoT UAV-aided networks in potato crops. *IEEE Access*, 8, pp. 56823-56836.

Fountas, S., Mylonas, N., Malounas, I., Rodias E., Santos, C. H., ve Pekkeriet E. 2020. Agricultural Robotics for Field Operations. *Sensors.*

El-Kebir, H., Shafa, T., Purushottam, A., Ornik, M. ve Soylemezoglu, A. 2022. High-Frequency Vibration Reduction for Unmanned Ground Vehicles on Unstructured Terrain. In International Conference on Modelling and Simulation for Autonomous Systems, pp. 74-92, Springer, Cham.

Foxlin, E. 1996. Inertial head-tracker sensor fusion by a complementary separate-bias Kalman filter. In Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium, pp. 185-194.

Flurry, G. 2021. Raspberry Pi 3 Model B+ Setup: Java on the Raspberry Pi. Apress, Berkeley, CA, pp. 21-48.

Furht, B., Grostick, D., Gluch, D., Rabbat, G., Parker, J. ve McRoberts, M. 2012. Real-time UNIX® Systems: design and application guide, Vol. 121, *Springer Science & Business Media*.

Gallmeister, B. 1995. POSIX. 4 Programming for the Real World: O'Reilly and Associates, Sebastopol, pp. 41-55.

Technology Inc, G. 2011. GPS Standalone Module Data Sheet Revision: V0A. In GPS Standalone Module Data Sheet Revision: V0A, pp. 4–37.

Guan, Y., He, J., Fu, X. ve Wang, D. 2021. High-Performance FOG IMU and Algorithm for Railway Dynamic Inspection. In 2021 IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), pp. 949-954.

Guha, A., Pavan, A., Liu, J., Rastogi, A. ve Steeves, T. 1995. Supporting real-time and multimedia applications on the Mercuri testbed. *IEEE Journal on Selected Areas in Communications*, 13(4), pp. 749-763.

Hong, S., Lee, M. H., Chun, H. H., Kwon, S. H. Ve Speyer, J. L. 2006. Experimental study on the estimation of lever arm in GPS/INS. *IEEE Transactions on vehicular technology*, 55(2), pp. 431-448.

Kaplan, E. D. 2017. Understanding GPS/GNSS: principles and applications. Hegarty, C. (Eds.), *Artech house*.

Henriksen, A. V., Edwards, G. T. ve Pesonen, L. A. 2020. Internet of Things in arable farming: Implementation, applications, challenges and potential. *Science Direct*, pp. 61-83.

Ahmad, N., Ghazilla, R. A. R., Khairi, N. M. ve Kasi, V. 2013. Reviews on various inertial measurement unit (IMU) sensor applications. *International Journal of Signal Processing Systems*, 1(2), pp. 256-262.

Prasad, R. ve Ruggieri, M. 2005. Applied satellite navigation using GPS, GALILEO, and augmentation systems. Artech House, pp. 23-68.

Jain, R. 2021. Apress: Advanced Home Automation Using Raspberry Pi, Agra, pp. 2-48.

Sean, McManus and Cook, Mike. 2021. Raspberry Pi for dummies, pp. 12-52.

Sinaei, K., Sarfi, M. H. ve Hosseinian, 2006. E. Design Optimization and Dynamic Balance Control of a 6-DoF Wheeled Biped Robot, *Knowledge Commons*, pp. 20-78.

Tanenhaus, M., Carhoun, D., Geis, T., Wan, E. ve Holland, A. 2012. Miniature IMU/INS with optimally fused low drift MEMS gyro and accelerometers for applications in GPS-denied environments. *In Proceedings of IEEE/ION PLANS*, pp. 259-264.

U. Nations, Economic & Social Affairs, 2018. World Agriculture Panel, pp. 1, Newyork.

Won, S. H. P., Golnaraghi, F. ve Melek, W. W. 2008. A fastening tool tracking system using an IMU and a position sensor with Kalman filters and a fuzzy expert system. *IEEE Transactions on Industrial Electronics*, 56(5), pp. 1782-1792.

Zhang, L., Dabipi, I. K. ve Brown Jr, W. L. 2018. Internet of Things applications for agriculture. Internet of things A to Z: technologies and applications, pp. 507-528.

Zhang, X. ve Davidson, E. A. 2018. Improving nitrogen and water management in crop production on a national scale. In AGU Fall Meeting Abstracts, Vol. 2018, pp. B22B-01.

Zhao, H. ve Wang, Z. 2011. Motion measurement using inertial sensors, ultrasonic sensors, and magnetometers with extended kalman filter for data fusion. *IEEE Sensors Journal*, 12(5), pp. 943-953.

Zhu, R. ve Zhou, Z. 2004. A real-time articulated human motion tracking using tri-axis inertial/magnetic sensors package. *IEEE Transactions on Neural systems and rehabilitation engineering*, 12(2), pp. 295-302.

7. EKLER

EK – 1 Navigasyon Yazılım Kodları

```
import threading
import time
import SensorService
import Manager
import Navigator
import Mqtt
import IGNS
import globalVal
from datetime import datetime
def statusReportPrint():
    print("=====STATUS=====")
    print("==MQTT==")
    print("mqtt queue:", globalVal.mqttReceiveQeue)
    print("mqtt send queue:", globalVal.mqttSend)
    print()
    print("==GPS==")
    print("gps online:", globalVal.GPS_Online)
    print("gps time:", globalVal.GPS_Time)
    print("gps normal:", globalVal.GPS_Nomal)
    print("gps raw:", globalVal.GPS_RAW)
    print()
    print("==IMU==")
    print("IMU eul:", globalVal.IMU_Eul)
    print()
    print("==ulstrasoon==")
    print(" ultrasoon 1:", globalVal.ultraSoon1)
    print(" ultrasoon 2:", globalVal.ultraSoon2)
    print()
    print("==motors==")
    print(" motor A:", globalVal.TargetSpeedA)
    print(" motor B:", globalVal.TargetSpeedB)
    print(" Moving :", globalVal.Moving)
    print(" Moving :", globalVal.Moving)
    print("MotorTimeStill: ", globalVal.MotorTimeStill)
    print("MotorTimeMoving: ", globalVal.MotorTimeMoving)
    print("MFeedbackA : ", globalVal.MFeedbackA)
    print("MFeedbackB : ", globalVal.MFeedbackB)
    print()
    ("==Waypoints==")
    print("currentWaypoint :",globalVal.currentWaypoint)
    print()
def write_logs():
    time.sleep(5)
    while True:
```

```
        time.sleep(1/20)
        dateTimeObj = datetime.now()
        timestampStr = dateTimeObj.strftime("%d-%b-%Y (%H:%M:%S.%f)")
           f = open("log_1.txt", "a")
        #f.write("time
,accelX,accelY,accelZ,RawGyroValx,RawGyroValy,RawGyroValz,RawMagValx,Raw
MagValy,RawMagValz,GPS_Nomal,RawTemp,IMU eul,motor A,motor B"
        f.write(timestampStr+","+
              str(globalVal.RawAccelVals[0])+","+
              str(globalVal.RawAccelVals[1])+","+
              str(globalVal.RawAccelVals[2])+","+
              str(globalVal.RawGyroVals[0])+","+
              str(globalVal.RawGyroVals[1])+","+
              str(globalVal.RawGyroVals[2])+","+
              str(globalVal.RawMagVals[0])+","+
              str(globalVal.RawMagVals[1])+","+
              str(globalVal.RawMagVals[2])+ ","+
              str(globalVal.GPS_Nomal)+","+
              str(globalVal.RawTemp)+ ","+
              str(globalVal.IMU_Eul)+","+
              str(globalVal.TargetSpeedA)+","+
              str(globalVal.TargetSpeedB)+"\n")
        f.close()
print("Main: starting...")
SensorService.SensorMain()
print(6)
IGNS.IGNS()
print(5)
Navigator.NavigatorMain()
print(4)
Manager.ManagerMain()
print(3)
mqtt = Mqtt.MqttMain()
print(2)
time.sleep(2)
print(1)
silent = False
t = threading.Thread(target=write_logs)
t.start()
while True:
    while silent:
        time.sleep(5)
    s = input("command?: ")
    if s == "status":
        statusReportPrint()
    else:
        # 36,896682;30,673322;20,0
        # indoor test lat : 36.8966,   long : 30.67743  -->   36,8966;30,67743;20,0python
```

```
        mqtt.sendDataToTopic("waypoint", s)
        time.sleep(1)
import time
import globalVal
from threading import Thread
import threading
import queue
class ManagerMain(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.lock = threading.Lock()
        self.start()
    def run(self):
        time.sleep(2)
        while True:
            self.readMqttCom()
            self.updateLog()
            time.sleep(1/20)
    def updateLog(self):
        l = list(globalVal.GPS_RAW.queue)
        if len(l) == 0:
            l.append((0, 0))
        if False:
            line = f"{globalVal.TargetSpeedA};{globalVal.TargetSpeedB};"
            line                                                                    +=
f"{globalVal.IMU_Lin_accel[0]};{globalVal.IMU_Lin_accel[1]};{globalVal.IMU_Lin
_accel[2]};"
            line                                                                    +=
f"{globalVal.IGNS_IMU_LinVelocity_Aligned[0]};{globalVal.IGNS_IMU_LinVeloci
ty_Aligned[1]};{globalVal.IGNS_IMU_LinVelocity_Aligned[2]};"
            line                                                                    +=
f"{globalVal.IGNS_GPS_Velocity[0]};{globalVal.IGNS_GPS_Velocity[1]};"
            #line = f"{globalVal.BatteryRawData};{globalVal.BatteryPercentage}"
            with open('Log.txt', 'a') as file:
                file.write(line + "\n")
    def readMqttCom(self):
        while not globalVal.mqttReceiveQeue.empty():
            msg = globalVal.mqttReceiveQeue.get()
            self.interpreteMqttmessage(msg[0], msg[1])
    def interpreteMqttmessage(self, topic, msg):
        with self.lock:
            if topic == "mode":
                globalVal.MqttMode = msg
            elif topic == "motor_control":
                globalVal.MqttMotorOverwrite = msg
            elif topic == "stop":
                globalVal.MqttStop = True
```

```
            print("change to True")
        elif topic == 'tempChanel':
            if msg == "reset":
                print("===IGNS location reset===")
                globalVal.IGNS_location = [0, 0, 0]
                print("===IGNS location reset DONE===")
            elif msg == "compass":
                globalVal.MqttMotorOverwrite = msg
            elif msg == "zero":
                globalVal.MqttMotorOverwrite = msg
            elif str(msg).count("head") > 0:
                globalVal.MqttMotorOverwrite = msg
            elif str(msg).count("for") > 0:
                globalVal.MqttMotorOverwrite = msg
            elif str(msg).count("back") > 0:
                globalVal.MqttMotorOverwrite = msg
            print(str(msg).count("head"))
            print(msg)
        elif topic == "waypoint":
            split = msg.replace(',', '.').split(';')
            if len(split) == 3:
                try:
                    long = float(split[0])
                    lat = float(split[1])
                    rad = float(split[2])
                    waypoint = globalVal.WayPoint(long, lat, rad)

                    globalVal.currentWaypoint = waypoint
                except:
                    pass
import math
import time
import queue
import globalVal
import queue
from threading import Thread
import threading
import paho.mqtt.client as mqtt
import numpy as np
np.set_printoptions(suppress=True)
MQTT_SERVER = "localhost"
MQTT_PATH = "motor_control"
class MqttMain(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.lock = threading.Lock()
```

```
        self.client = 0
        self.connectedRetry = True
        self.mqtt_connected = False
        self.start()
    def run(self):
        self.client = mqtt.Client()
        self.client.on_connect = self.on_connect
        self.client.on_message = self.on_message
        while self.connectedRetry:
            print("Mqtt: try connecting...")
            try:
                self.client.connect(MQTT_SERVER, 1883, 60)
                self.client.loop_start()
                self.connectedRetry = False
                print("start connection mqtt")
            except:
                print("====Mqtt error ID011====")
                time.sleep(1)
        while True:
            if self.mqtt_connected:
                self.sendSensorData()
                time.sleep(0.2)
            else:
                time.sleep(1)
    def on_connect(self, client, obj, flags, rc):
        print("Connected with result code " + str(rc))
        self.client.subscribe("motor_control")
        self.client.subscribe("mode")
        self.client.subscribe("stop")
        self.client.subscribe("tempChanel")
        self.client.subscribe("waypoint")
        self.mqtt_connected = True
    def on_message(self, client, userdata, msg):
        print(msg.topic + " " + str(msg.payload))
        with self.lock:
            if globalVal.mqttReceiveQeue.maxsize <= globalVal.mqttReceiveQeue.qsize():
                globalVal.mqttReceiveQeue.get()
            globalVal.mqttReceiveQeue.put([msg.topic, str(msg.payload, 'utf-8')])
    def publish_mqqt(self, topic, msg):
        self.client.publish(topic, msg)
    def sendSensorData(self):
        self.publish_mqqt("sensors", self.getSensorData())
    def sendDataToTopic(self, topic, data):
        self.publish_mqqt(topic, str(data))
    def getSensorData(self):
        string                                                                    =
f"GPS_ONLINE:{globalVal.GPS_Online};GPS_Time:{globalVal.GPS_Time};" \
```

```
f"GPS_Nomal:{globalVal.GPS_Nomal};GPS_Satellites:{globalVal.GPS_Satellites};" \
            f"GPS_DOP:{globalVal.GPS_DOP};"
        string += f"GPS_LOC:{globalVal.GPS_Loc};GPS_Track_angle:{globalVal.GPS_Track_angle};" \
            f"GPS_Velocity:{globalVal.GPS_Velocity};;"
        string += f"IMU_Eul_corrected:{globalVal.IMU_EulXYZ_Corrected[0]};" \
f"IMU_EulXYZ:{globalVal.IMU_EulXYZ};IMU_EulCorrected:{globalVal.IMU_EulXYZ_Corrected};" \
            f"IMU_Lin_accel:{globalVal.IMU_Lin_accel}"
        string += f"IMU_Compass:{self.angleCompass(globalVal.IMU_Compass)};;" \
            f"Ultrasoon1:{globalVal.ultraSoon1};Ultrasoon2:{globalVal.ultraSoon2};;"
        string += f"IGNS_active:{globalVal.IGNS_active};"
        if globalVal.IGNS_active:
            string += f"IGNS_Kalmam:;{np.around(globalVal.IGNS_Kalman_out, decimals=2)};"
            string += f"IGNS_Location:{np.around(globalVal.IGNS_location, decimals=2)};"
        string += f"IGNS_GPS_velocity:{globalVal.IGNS_GPS_Velocity};" \
            f"IMU_velocitys_aligned:{globalVal.IGNS_IMU_LinVelocity_Aligned};;"
        string += f"MotorTimeStill:{globalVal.MotorTimeStill};MotorTimeMoving:{globalVal.MotorTimeMoving};"
        string += f"BatteryPercentage:{globalVal.BatteryPercentage};BatteryRaw:{globalVal.BatteryRawData};"
        return string
    def angleCompass(self, mag):
        heading = 180 * math.atan2(mag[1], mag[0]) / math.pi
        if heading < 0:
            heading += 360
        return heading
import time
import math
from time import sleep
import time
import globalVal
from threading import Thread
import threading
from simple_pid import PID
import statistics
class NavigatorMain(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.lock = threading.Lock()
```

```
        self.heading = 0
        self.targetHeading = 0
        self.compassOffset = 0
        self.NORMAL_MARGE = 5
        #MotorDrivePid
        Kp = 5
        Ki = 1
        Kd = 0
        self.MotorDrivePid = PID(Kp, Ki, Kd, setpoint=0)
        self.MotorDrivePid.sample_time = 0.1

        #MotorHeadingPid
        Kp = 4
        Ki = 2
        Kd = 1.5
        self.MotorHeadingPid = PID(Kp, Ki, Kd, setpoint=0)
        self.MotorHeadingPid.sample_time = 0.1
        self.MotorHeadingPid.output_limits = (-255, 255) ##Anti Windup
        self.start()
    def run(self):
        while True:
            self.motorOverwrite()
            self.checkForTask()
            time.sleep(0.1)
    def checkForTask(self):
        if globalVal.currentWaypoint != None:
            waypoint = globalVal.currentWaypoint
            pos = globalVal.IGNS_Kalman_out
            print("pos")
            print(pos)
            print("waypoint")
            print(waypoint)
            print("buraya girdi checkfortask")
            if not waypoint.isInRange(pos[0], pos[1]) or True:
                self.dynamicMotorControlHeading(self.checkKeepMoving,
waypoint.getHeading(pos[0], pos[1]), 100)
            else:
                with self.lock:
                    globalVal.currentWaypoint = None
    def deltaHoek(self,h1, h2):
        angle1 = h1 - h2
        angle2 = ((h1 + 180) % 360) - ((h2 + 180) % 360)
        if abs(angle1) < abs(angle2):
            return angle1
        return angle2
    def IMUheading(self, compass = False):
        eul = globalVal.IMU_EulXYZ[0]
```

```
    if compass:
        eul = globalVal.IMU_EulXYZ_Corrected[0]
    return eul
def motorSturing(self, motorA, motorB):
    a = int(motorA)
    b = int(motorB)
    if abs(a - 255) > 255:
        a = 255
    if abs(b - 255) > 255:
        b = 255
    with self.lock:
        globalVal.TargetSpeedA = a
        globalVal.TargetSpeedB = b
def goToTargetHeading(self, target, marge):
    print("goToTargetHeading START")
    heading = self.IMUheading(True)
    delta = self.deltaHoek(heading, target)
    pid = self.MotorHeadingPid
    while abs(delta) > abs(marge):
        while abs(delta) > abs(marge):
            sleep(0.1)
            delta = self.deltaHoek(self.IMUheading(True), target)
            control = pid(delta)
            motorA = self.minMax(255 + control, 0, 510)
            motorB = self.minMax(255 - control, 0, 510)
            self.motorSturing(motorA, motorB)
        self.motorSturing(255, 255)
        sleep(1)
        delta = self.deltaHoek(self.IMUheading(True), target)
    self.motorSturing(255, 255)
def angleCompass(self, mag):
    heading = 180 * math.atan2(mag[1], mag[0]) / math.pi
    if heading < 0:
        heading += 360
    return heading
def Range360(self, number):
    while number >= 360 or number < 0:
        number = number % 360
        if number < 0:
            number = 360 - number
    return number
def ZeroCompassOffset(self):
    compassOffset = self.Range360(self.deltaHoek(0, globalVal.IMU_EulXYZ[0]))
    print("compassOfsset:", compassOffset)
    with self.lock:
        globalVal.IMU_EulCompassOffset = compassOffset
def calibrateCompassOffset(self):
```

```
        sleep(3)
        acceptReading = False
        reading = 0
        while not acceptReading:
            tempList = []
            for i in range(500):
                tempList.append(self.angleCompass(globalVal.IMU_Compass))
                time.sleep(0.01)
            reading = statistics.median(tempList)
            dev = statistics.stdev(tempList)
            print("dev:", dev)
            if dev < 2:
                compassOffset = self.deltaHoek(0, reading) -
self.deltaHoek(globalVal.IMU_Eul, 0)
                if abs(compassOffset) > 50:
self.goToTargetHeading(self.Range360(globalVal.IMU_EulXYZ_Corrected[0] +
compassOffset), 10)
                    sleep(3)
                else:
                    acceptReading = True
            elif dev > 80:
self.goToTargetHeading(self.Range360(globalVal.IMU_EulXYZ_Corrected[0] + 90),
10)
                sleep(3)
            else:
                sleep(1)
        print("reading:", reading)
        compassOffset = self.Range360(self.deltaHoek(0, reading) -
self.deltaHoek(globalVal.IMU_Eul, 0) + 180)
        print("compassOfsset:", compassOffset)
        with self.lock:
            globalVal.IMU_EulCompassOffset = compassOffset
    def motorOverwrite(self):
        if globalVal.MqttMode == "1":
            a = 255
            b = 255
            if globalVal.MqttMotorOverwrite == 'LEFT':
                a = 0
                b = 510
            elif globalVal.MqttMotorOverwrite == 'RIGHT':
                a = 510
                b = 0
            elif globalVal.MqttMotorOverwrite == 'FOR':
                self.dynamicMotorControlHeading(self.checkKeepMoving,
self.IMUheading(True), 60)
            elif globalVal.MqttMotorOverwrite.count('for') > 0:
                s = globalVal.MqttMotorOverwrite.split('for ')[1]
                head = int(s)
```

```
            self.dynamicMotorControlHeading(self.checkKeepMoving, head, 60)
        elif globalVal.MqttMotorOverwrite.count('back') > 0:
            s = globalVal.MqttMotorOverwrite.split('back ')[1]
            head = int(s)
            self.dynamicMotorControlHeading(self.checkKeepMoving, head, -100)
        elif globalVal.MqttMotorOverwrite == 'BACK':
            self.dynamicMotorControlHeading(self.checkKeepMoving,
self.IMUheading(True), -100)
        elif globalVal.MqttMotorOverwrite == 'STOP':
            a = 255
            b = 255
        elif globalVal.MqttMotorOverwrite == 'compass':
            self.calibrateCompassOffset()
        elif globalVal.MqttMotorOverwrite == 'zero':
            self.ZeroCompassOffset()
            print('zero')
        elif globalVal.MqttMotorOverwrite.count('head') > 0:
            s = globalVal.MqttMotorOverwrite.split('head ')[1]
            head = int(s)
            self.goToTargetHeading(head, 5)
        self.motorSturing(a, b)
        globalVal.MqttMotorOverwrite = ''
        return True
    return False
  def checkKeepMoving(self):
    if globalVal.MqttStop:
        with self.lock:
            print("movement canceld")
            globalVal.MqttStop = False
            globalVal.currentWaypoint = None
        return False
    elif globalVal.currentWaypoint != None:
        waypoint = globalVal.currentWaypoint
        pos = globalVal.IGNS_Kalman_out
        offset = abs(self.deltaHoek(waypoint.getHeading(pos[0], pos[1]),
self.IMUheading(True)))
        if waypoint.isInRange(pos[0], pos[1]) or offset > 20:
            return False
    return True
  def dynamicMotorControlHeading(self, continueMoving, heading, speed = 60):
    print("dynamicMotorControlHeading START")
    self.goToTargetHeading(heading, 3)
    MIN_SPEED = 70
    MAX_SPEED = 255
    speed = (MAX_SPEED - MIN_SPEED) * (speed/100) + MIN_SPEED + 255 #
convert to PWN (70-255)
    motorA = speed
    motorB = speed
```

```
        pid = self.MotorDrivePid
        print("started to move")
        while continueMoving():
            print("moving...")
            sleep(0.1)
            delta = self.deltaHoek(heading, self.IMUheading(True))
            control = pid(delta)
            if control >= 0:
                motorA = speed - abs(control)
            else:
                motorB = speed - abs(control)
            self.motorSturing(motorA, motorB)
        self.motorSturing(255,255)
        print("done")
        return
    def minMax(self, x, min, max):
        if x < min:
            x = min
        if x > max:
            x = max
        return x
from time import sleep
import importlib
import globalVal
from threading import Thread
import threading
import serial
from smbus2 import SMBus
import board
import busio
import adafruit_bno055
import math
import queue
from imusensor.MPU9250 import MPU9250
import smbus
class SensorMain(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.lock = threading.Lock()
        self.gpgga_info = "$GPGGA,"
        self.ser = serial.Serial("/dev/ttyS0")  # Open port with baud rate
        #self.ser = serial.Serial("/dev/ttyS0")  # Open port with baud rate
        self.arduinoAdr = 0x08  # Adres of the Arduino
        self.i2c = busio.I2C(board.SCL,board.SDA)
        #i2c = board.I2C()
        print(self.i2c)# uses board.SCL and board.SDA
        #self.sensor = adafruit_bno055.BNO055_I2C(self.i2c, 0x68)
```

```
        #self.sensor = mpu6050(0x68)address = 0x68
        bus = smbus.SMBus(1)
        imu = MPU9250.MPU9250(bus, 0x68)
        imu.begin()
        self.sensor = imu
        self.sensor.readSensor()
        #sensor = adafruit_bno055.BNO055_I2C(i2c, 0x29)
        self.start()
    def run(self):
        sleep(2)
        while True:
            self.updateGPSData()
            #self.updateUltrasoon()
            self.updateIMU()
            #self.updateBatteryPercentage()
            self.updateMotors()
            sleep(1/(globalVal.READ_FREQUENCY*2))
            self.updateMotors()
            sleep(1/(globalVal.READ_FREQUENCY*2))
    def updateMotors(self):
        mA = globalVal.TargetSpeedA
        mB = globalVal.TargetSpeedB
        sA = '000' + str(mA)
        sB = '000' + str(mB)
        if str(mA) == '255' and str(mB) == '255':
            with self.lock:
                globalVal.Moving = False
                globalVal.MotorTimeMoving = 0
                globalVal.MotorTimeStill += 1/ globalVal.READ_FREQUENCY * 100 if
globalVal.MotorTimeStill < 1000 else 0
        else:
            with self.lock:
                globalVal.Moving = True
                globalVal.MotorTimeStill = 0
                globalVal.MotorTimeMoving += 1 / globalVal.READ_FREQUENCY * 100 if
globalVal.MotorTimeMoving < 1000 \
                    else 0
        self.writeData(sA[-3:] + sB[-3:])
        return
    def updateBatteryPercentage(self):
        data = self.disassembelData(self.readData())
        percentage = self.calcBatteryPercentage(int(data[6]))
        if percentage > globalVal.BatteryPercentage or globalVal.Moving:
            percentage = globalVal.BatteryPercentage
        with self.lock:
            globalVal.BatteryRawData = data[6]
            globalVal.BatteryPercentage = percentage
```

```
    def calcBatteryPercentage(self, d):
        i = 0
        for a in globalVal.BatteryPercentageCalibration:
            if d < a:
                return i/2 if i != 0 else 0
            i += 1
        return 100
    def CalculateEuler(self, gyro, accel):
        gyroX = gyro["x"]
        gyroY = gyro["y"]
        gyroZ = gyro["z"]
        accelX = accel["x"] * 3.9
        accelY = accel["y"] * 3.9
        accelZ = accel["z"] * 3.9
        pitch = 180 * math.atan(accelX / math.sqrt(math.pow(accelY,2) + math.pow(accelZ,2))) / math.pi
        roll = 180 * math.atan(accelY / math.sqrt(math.pow(accelX,2) + math.pow(accelZ,2))) / math.pi
        yaw = 180 * math.atan(accelZ / math.sqrt(math.pow(accelY,2) + math.pow(accelX,2))) / math.pi
        return [pitch, roll, yaw]
    def updateIMU(self):
        try:
            self.sensor.readSensor()
            self.sensor.readRawSensor()
            self.sensor.computeOrientation()
            globalVal.RawAccelVals = self.sensor.RawAccelVals
            globalVal.RawGyroVals = self.sensor.RawGyroVals
            globalVal.RawMagVals = self.sensor.RawMagVals
            globalVal.RawTemp = self.sensor.RawTemp
            #print ("roll: {0} ; pitch : {1} ; yaw : {2}".format(self.sensor.roll, self.sensor.pitch, self.sensor.yaw))
            eulXYZ = [self.sensor.roll, self.sensor.pitch, self.sensor.yaw] #[1.23123,1,3434,1,123123]
            eulXYZold = globalVal.IMU_EulXYZ
            for i in range(0, 3):
                if abs(eulXYZ[i]) > 360:
                    eulXYZ[i] = eulXYZ_old[i]
                    print("read error IMU, eul: x(0) Y(1) Z(2) ->", i)
            eulXYZ = tuple(eulXYZ)
            eul = eulXYZ[0]
            compass = self.sensor.MagVals # self.sensor.MagVals[0], self.sensor.MagVals[1], self.sensor.MagVals[2]
            accel = self.correctAcceleration(self.sensor.AccelVals)
            correctedEul = self.CorrectEul(eulXYZ)
            with self.lock:
                globalVal.IMU_Eul = eul
                globalVal.IMU_Compass = compass
```

```
            globalVal.IMU_EulXYZ = eulXYZ
            globalVal.IMU_Lin_accel = accel
            globalVal.IMU_EulXYZ_Corrected = tuple(correctedEul)
        except:
            print("====I2C (IMU) Error: ID003====")
    def correctAcceleration(self, accel):
        for a in accel:
            if a > 10 or a < -10:
                print("====I2C (Read) error ID005====")
                print("velocity --> " + str(a))
                return globalVal.IMU_Lin_accel
        return accel
    def CorrectEul(self, eul):
        out = [0, eul[1], 0]
        out[0] = self.CompassOffsetOnAngle(eul[0])
        out[2] = self.CorrectEulZ(eul[2])
        return out
    def CorrectEulZ(self, angle):
        if angle < 0:
            temp = 180 - abs(angle)
            temp = temp * -1
            return temp
        else:
            temp = 180 - angle
            return temp
    def CompassOffsetOnAngle(self, angle):
        return self.Range360(angle + globalVal.IMU_EulCompassOffset)
    def Range360(self, number):
        while number >= 360 or number < 0:
            number = number % 360
            if number < 0:
                number = 360 - number
        return number
    def updateUltrasoon(self):
        data = self.disassembelData(self.readData())
        with self.lock:
            globalVal.ultraSoon1 = data[4]
            globalVal.ultraSoon2 = data[5]
        return
    # strig to byte array
    def ConvertStringsToBytes(self, src):
        converted = []
        for b in src:
            converted.append(ord(b))
        return converted
    # byte array to string
    def bytesToString(self, byte):
        ret = []
```

```
        for b in byte:
            if b is not 255:
                ret.append(b)
        return bytearray(ret).decode("utf-8")
    # send data with I2C
    def writeData(self, data):
        byteToSend = self.ConvertStringsToBytes(data)  # convert to bytes
        try:
            with SMBus(1) as bus:
                bus.write_i2c_block_data(self.arduinoAdr, 0, byteToSend)
        except:
            return False
        return True
    # read data from I2C
    def readData(self):
        try:
            with SMBus(1) as bus:
                data = bus.read_i2c_block_data(self.arduinoAdr, 0, 30)
                ret = self.bytesToString(data)
                return ret
        except Exception as e:
            print(e)
            print("====I2C (Read) error ID002====")
        return
    def disassembelData(self, data):
        if data == None:
            return None
        ret = []
        temp = ""
        start = False
        for l in data:
            if l == '/':
                ret.append(temp)
                temp = ""
            elif l == '}':
                ret.append(temp)
                return ret
            elif l == '{':
                start = True
            elif start:
                temp += l
        return
    def updateGPSData(self):
        try:
            while self.ser.in_waiting > 0:
                received_data = (str)(self.ser.readline())  # read NMEA string received
                GPGGA_data_available = received_data.find(self.gpgga_info)  # check for
NMEA GPGGA string
```

```
            if received_data.find("$GPGSA") > 0:
                GPGGA_buffer = received_data.split("$GPGSA,", 1)[1]  # store data coming after "$GPGGA," string
                NMEA_buff = (GPGGA_buffer.split(','))  # store comma separated data in buffer
                with self.lock:
                    globalVal.GPS_DOP = (NMEA_buff[14], NMEA_buff[15], NMEA_buff[16].split("*")[0])
            if received_data.find("$GPVTG") > 0:
                GPGGA_buffer = received_data.split("$GPVTG,", 1)[1]  # store data coming after "$GPGGA," string
                NMEA_buff = (GPGGA_buffer.split(','))  # store comma separated data in buffer
                with self.lock:
                    globalVal.GPS_Track_angle = float(NMEA_buff[0])
                    globalVal.GPS_Velocity = float(NMEA_buff[6])
            if (GPGGA_data_available > 0):
                GPGGA_buffer = received_data.split("$GPGGA,", 1)[1]  # store data coming after "$GPGGA," string
                NMEA_buff = (GPGGA_buffer.split(','))  # store comma separated data in buffer
                self.GPS_Info(NMEA_buff)
                return True
        return False
    except (RuntimeError, TypeError, NameError):
        print("====GPS error ID001====")
        return False
  def convert_to_degrees(self, raw_value):
    decimal_value = raw_value / 100.00
    degrees = int(decimal_value)
    mm_mmmm = (decimal_value - int(decimal_value)) / 0.6
    position = degrees + mm_mmmm
    position = "%.4f" % (position)
    return position
  def GPS_Info(self, NMEA_buff):
    nmea_time = NMEA_buff[0]  # extract time from GPGGA string
    nmea_latitude = NMEA_buff[1]  # extract latitude from GPGGA string
    nmea_longitude = NMEA_buff[3]  # extract longitude from GPGGA string
    nmea_altitude = NMEA_buff[10]
    nmea_Satellites = NMEA_buff[6] # Number of satellites
    try:
        lat = float(nmea_latitude)  # convert string into float for calculation
        longi = float(nmea_longitude)  # convertr string into float for calculation
    except ValueError:
        with self.lock:
            globalVal.GPS_Online = False
        return
```

```
        lat_in_degrees = self.convert_to_degrees(lat)  # get latitude in degree decimal format
        long_in_degrees = self.convert_to_degrees(longi)  # get longitude in degree decimal format
        with self.lock:
            globalVal.GPS Online = True
            globalVal.GPS Time = nmea_time
            globalVal.GPS_Satellites = nmea_Satellites
            if globalVal.GPS_RAW.maxsize <= globalVal.GPS_RAW.qsize():
                globalVal.GPS_RAW.get()
            globalVal.GPS_RAW.put([lat_in_degrees, long_in_degrees, nmea_altitude])
        self.GPS_calculateNominal()
        self.GPSCalculateLocal()
        return
    def GPS_calculateNominal(self):
        if globalVal.GPS RAW.qsize() != globalVal.GPS RAW.qsize():
            return
        lat = 0
        long = 0
        alt = 0
        for t in range(0, globalVal.GPS_RAW.qsize()):
            temp = globalVal.GPS RAW.get()
            lat += float(temp[0])
            long += float(temp[1])
            alt += float(temp[2])
            globalVal.GPS_RAW.put(temp)
        lat = lat / globalVal.GPS_RAW.qsize()
        long = long / globalVal.GPS_RAW.qsize()
        alt = alt / globalVal.GPS_RAW.qsize()
        with self.lock:
            globalVal.GPS_Nomal = [lat, long, alt]
        return
    def GPSCalculateLocal(self):
        lon1 = globalVal.GPS_Ref[0]
        lat1 = globalVal.GPS_Ref[1]
        lon2 = globalVal.GPS Nomal[0]
        lat2 = globalVal.GPS Nomal[1]
        dx = (lon1 - lon2) * 4000 * math.cos((lat1 + lat2) * math.pi / 360) / 360 * 1000
        dy = (lat1 - lat2) * 4000 / 360 * 1000
        with self.lock:
            globalVal.GPS_Loc[0] = dx
            globalVal.GPS_Loc[1] = dy
from time import sleep
import importlib
import globalVal
from threading import Thread
import threading
import serial
```

```
from smbus2 import SMBus
import board
import busio
import adafruit_bno055
import math
import queue
from imusensor.MPU9250 import MPU9250
import smbus
class LogMain(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.lock = threading.Lock()
        self.start()
    def run(self):
        sleep(2)
        while True:
            self.updateLogs()
    def updateLogs(self):
         f = open("log_1.txt", "a")
         f.write("IMU eul:"+ str(globalVal.IMU_Eul)+" motor A:"+
str(globalVal.TargetSpeedA)+" motor B:"+ str(globalVal.TargetSpeedB))
         f.close()
import math
import time
import globalVal
from threading import Thread
import threading
import queue
import numpy as np
class IGNS(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.lock = threading.Lock()
        self.predictFrequency = 100
        self.updateFrequency = 20
        self.deltaTime = 1/self.predictFrequency
        self.kalman = None
        time.sleep(5)
        globalVal.MqttMotorOverwrite = "zero"
        time.sleep(1)
        with self.lock:
            globalVal.IGNS_active = True
        self.start()
    def run(self):
        i = 0
        while True:
```

```
        while not globalVal.IGNS_active:
            time.sleep(0.001)
        self.predictPosition()
        self.IGNSPosition(self.deltaTime)
        if i * self.deltaTime >= 1/self.updateFrequency:
            i = 0
            self.updatePosition()
        time.sleep(self.deltaTime)
        i += 1
    def IGNSPosition(self, deltaT):
        Axyz = globalVal.IGNS_IMU_LineerVelocity_Aligned
        Vxyz = globalVal.IGNS_velocity
        Pxyz = globalVal.IGNS_location
        if 1 < globalVal.MotorTimeMoving < 5:
            Vxyz[0] += Axyz[0] * deltaT
            Vxyz[1] += Axyz[1] * deltaT
            Vxyz[2] += Axyz[2] * deltaT
        if globalVal.Moving:
            Pxyz[0] += Vxyz[0] * deltaT
            Pxyz[1] += Vxyz[1] * deltaT
            Pxyz[2] += Vxyz[2] * deltaT
        else:
            Vxyz = [0, 0, 0]

        with self.lock:
            globalVal.IGNS_velocity = V
            globalVal.IGNS_location = P
    def GPSVeloctyUpdate(self):
        angle = globalVal.GPS_TrackAngle
        vel = globalVal.GPS_Velocity
        Vx = math.cos(angle) * vel / 3.6
        Vy = math.sin(angle) * vel / 3.6
        v = [Vx, Vy, 0]
        with self.lock:
            globalVal.IGNS_GPS_Velocity = v
        return
    def IMUVelocitys(self):
       eulXYZ = globalVal.IMU_EulXYZ_Corrected
        y = eulXYZ[0] #yaw
        p = eulXYZ[1] #pitch
        r = eulXYZ[2] #roll
        c = math.cos
        s = math.sin
        mes = np.array([
            [globalVal.IMU_Lin_accel[0]*10],
            [0],
            [0]
        ])
```

```
        trans = np.array([
            [
                [c(y)*c(p), c(p) * s(y), -s(p)],
                [c(y)*s(p)*s(r)-c(r)*s(y), c(y)*c(r)+s(y)*s(r), c(p)*s(r)],
                [c(y)*c(r)*s(p)+s(y)*s(r), c(r)*s(y)*s(p)-c(y)*s(r), c(p)*c(r)]
            ]
        ])
        out = np.dot(trans, mes).tolist()[0]
        out = [out[0][0]*-1, out[1][0]*-1, out[2][0]]
        with self.lock:
            globalVal.IGNS_IMU_LinVelocity_Aligned = out
    def predictPosition(self):
        self.IMUVelocitys()
        if self.kalman is not None:
            u = np.array(
                [
                    [globalVal.IGNS_IMU_LinVelocity_Aligned[0] * 100],
                    [globalVal.IGNS_IMU_LinVelocity_Aligned[1] * 100],
                    [globalVal.IGNS_IMU_LinVelocity_Aligned[2] * 100],
                    [0],
                    [0],
                    [0],
                    [0],
                    [0],
                    [0],
                ])
            self.kalman.Q = self.BerekenVariance(globalVal.IGNS_Q_Variance) #Default
            if globalVal.Moving:
                if 1 < globalVal.MotorTimeMoving < 5:
                    variance = globalVal.IGNS_Q_Variance
                    variance[0] = 0.001
                    variance[1] = 0.001
                    variance[2] = 0.001
                    self.kalman.Q = self.BerekenVariance(variance)
                elif 1 < globalVal.MotorTimeMoving:
                    variance = globalVal.IGNS_Q_Variance
                    variance[0] = 0.0001
                    variance[1] = 0.0001
                    variance[2] = 0.0001
                    self.kalman.Q = self.BerekenVariance(variance)
                    u = np.array(
                        [
                            [0],
                            [0],
                            [0],
                            [0],
                            [0],
```

```
                    [0],
                    [0],
                    [0],
                    [0],
                ])
        else:
            variance = globalVal.IGNS_Q_Variance
            variance[0] = 10
            variance[1] = 10
            variance[2] = 10
            self.kalman.Q = self.BerekenVariance(variance)
           u = np.array(
                [
                    [0],
                    [0],
                    [0],
                    [0],
                    [0],
                    [0],
                    [0],
                    [0],
                    [0],
                ])
        x = self.kalman.predict(u)
        with self.lock:
            globalVal.IGNS_Kalman_out = x.T[0]
        return x.T[0]
    else:
        self.KalmanInit()
    return
def updatePosition(self):
    if self.kalman is not None:
        self.GPSVeloctyUpdate()
        z = np.array(
            [
                [globalVal.GPS_Loc[0]],
                [globalVal.GPS_Loc[1]],
                [0],
                [globalVal.IGNS_GPS_Velocity[0]],
                [globalVal.IGNS_GPS_Velocity[1]],
                [globalVal.IGNS_GPS_Velocity[2]],
                [0],
                [0],
                [0],
            ])
        self.kalman.R = self.BerekenVariance(globalVal.IGNS_R_Variance) #Default
        if globalVal.Moving:
            pass
```

```
            else:
                variance = globalVal.IGNS_R_Variance
                if 900 < globalVal.MotorTimeStill < 930:
                    variance[0] = 30
                    variance[1] = 30
                z = np.array(
                    [
                        [globalVal.GPS_Loc[0]],
                        [globalVal.GPS_Loc[1]],
                        [0],
                        [0],
                        [0],
                        [0],
                        [0],
                        [0],
                        [0],
                    ])
                variance[3] = 0.0000001
                variance[4] = 0.0000001
                variance[5] = 0.0000001
                self.kalman.R = self.BerekenVariance(variance)
            self.kalman.update(z)
    def deltaHoek(self, h1, h2):
        angle1 = h1 - h2
        angle2 = ((h1 + 180) % 360) - ((h2 + 180) % 360)
        if abs(angle1) < abs(angle2):
            return angle1
        return angle2
    def Range360(self, number):
        while number >= 360 or number < 0:
            number = number % 360
            if number < 0:
                number = 360 - number
        return number
###====================KALMAN================================
=====
    def BerekenVariance(self, variance):
        varP2 = []
        for vari in variance:
            varP2.append(pow(vari, 2))
        v = varP2 * np.eye(9)
        return v
    def KalmanInit(self):
        dt = self.deltaTime
        F = np.array(
            [
                [1, 0, 0, dt, 0, 0, 0, 0, 0],  ## Location X
                [0, 1, 0, 0, dt, 0, 0, 0, 0],  ## Location Y
```

```
        [0, 0, 1, 0, 0, dt, 0, 0, 0],  ## Location Z
        [0, 0, 0, 1, 0, 0, 0, 0, 0],  ## Velocity X
        [0, 0, 0, 0, 1, 0, 0, 0, 0],  ## Velocity Y
        [0, 0, 0, 0, 0, 1, 0, 0, 0],  ## Velocity Z
        [0, 0, 0, 0, 0, 0, 1, 0, 0],  ## Orientation 1
        [0, 0, 0, 0, 0, 0, 0, 1, 0],  ## Orientation 2
        [0, 0, 0, 0, 0, 0, 0, 0, 1],  ## Orientation 3
    ])
bDt = pow(dt, 2) / 2
B = np.array(
    [
        [-bDt, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, bDt, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, -bDt, 0, 0, 0, 0, 0, 0],
        [-dt, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, dt, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, -dt, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1],
    ])
H = np.array(
    [
        [1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1],
    ])
Q = self.BerekenVariance(globalVal.IGNS_Q_Variance)
R = self.BerekenVariance(globalVal.IGNS_R_Variance)
P = np.array(
    [
        [1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
    ])
x = np.array(
```

```
            [
                [0],
                [0],
                [0],
                [0],
                [0],
                [0],
                [globalVal.IMU_EulXYZ_Corrected[0]],
                [globalVal.IMU_EulXYZ_Corrected[1]],
                [globalVal.IMU_EulXYZ_Corrected[2]],
            ]
        )
        self.kalman = KalmanFilter(F, B, H, Q, R, P, x)
        return
class KalmanFilter(object):
    def __init__(self, F = None, B = None, H = None, Q = None, R = None, P = None, x0
= None):
        if(F is None or H is None):
            raise ValueError("Set proper system dynamics.")
        self.n = F.shape[1]
        self.m = H.shape[1]
        self.F = F
        self.H = H
        self.B = 0 if B is None else B
        self.Q = np.eye(self.n) if Q is None else Q
        self.R = np.eye(self.n) if R is None else R
        self.P = np.eye(self.n) if P is None else P
        self.x = np.zeros((self.n, 1)) if x0 is None else x0
    def predict(self, u=0):
        self.x = np.dot(self.F, self.x) + np.dot(self.B, u)
        self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q
        return self.x
    def update(self, z):
        y = z - np.dot(self.H, self.x)
        S = self.R + np.dot(self.H, np.dot(self.P, self.H.T))
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
        self.x = self.x + np.dot(K, y)
        I = np.eye(self.n)
        self.P = np.dot(np.dot(I - np.dot(K, self.H), self.P)
                    , (I - np.dot(K, self.H)).T) + np.dot(np.dot(K, self.R), K.T)
import queue
import numpy
import math
global publicTransfer
#Mqtt service
global mqttReceiveQeue
mqttReceiveQeue = queue.Queue(maxsize=20)
```

```
global mqttSend
mqttSend = queue.Queue(maxsize=20)
global MqttMode
MqttMode = "1"
global MqttMotorOverwrite
MqttMotorOverwrite = "255255"
global MqttStop
MqttStop = False
##Sensor Service
global READ_FREQUENCY
READ_FREQUENCY = 100
#GPS
global GPS_Online
GPS_Online = False
global GPS_Time
GPS_Time = 0
global GPS_Nomal
GPS_Nomal = 0
global GPS_RAW
GPS_RAW = queue.Queue(maxsize=10)
global GPS_Satellites
GPS_Satellites = 0
global GPS_DOP
GPS_DOP = (-1.-1.-1)
global GPS_Ref
GPS_Ref = (52.119, 3.038)
global GPS_Loc
GPS_Loc = [0, 0]
global GPS_Track_angle
GPS_Track_angle = 0
global GPS_Velocity
GPS_Velocity = 0 ## Km/h
#IMU
global IMU_Eul
IMU_Eul = 0
global IMU_Compass
IMU_Compass = [0, 0, 0]
global IMU_EulXYZ
IMU_EulXYZ = (0, 0, 0)
global IMU_EulXYZ_Corrected
IMU_EulXYZ_Corrected = (0, 0, 0)
global IMU_Lin_accel
IMU_Lin_accel = (0, 0, 0)
global IMU_EulCompassOffset
IMU_EulCompassOffset = 0
#Ultrasoon
global ultraSoon1
ultraSoon1 = -1
```

```
global ultraSoon2
ultraSoon2 = -1
#Motor control
global TargetSpeedA
TargetSpeedA = 255
global TargetSpeedB
TargetSpeedB = 255
global Moving
Moving = False
global MotorTimeStill
MotorTimeStill = 1000
global MotorTimeMoving
MotorTimeMoving = 0
global MFeedbackA
MFeedbackA = -1
global MFeedbackB
MFeedbackB = -1
global BatteryRawData
BatteryRawData = -1
global BatteryPercentage
BatteryPercentage = 100
global BatteryPercentageCalibration
BatteryPercentageCalibration = [677.85, 679.23, 713.04, 731.23, 744.38, 755.45, 764.23,
770.62, 777.65, 783.02, 787.1, 791.45, 794.6, 797.54, 799.64, 801.9, 803.81, 805.13,
806.73, 808.71, 809.22, 810.25, 810.64, 811.09, 812.07, 812.72, 813.72, 814.11, 814.84,
877.09, 877.11, 877.38, 877.53, 877.65, 877.73, 878.5, 878.9, 879.05, 879.07, 879.17,
879.31, 879.56, 879.59, 879.77, 880.26, 880.87, 881.1, 881.46, 882.36, 882.41, 882.82,
882.86, 883.01, 883.53, 883.99, 885.12, 885.67, 886.69, 888.41, 889.48, 891.22, 894.41]
global currentWaypoint
currentWaypoint = None
#IGNS
global IGNS_active
IGNS_active = False
global IGNS_location
IGNS_location = [0, 0, 0]
global RawAccelVals
global RawGyroVals
global RawMagVals
global RawTemp
global IGNS_IMU_LinVelocity_Aligned
IGNS_IMU_LinVelocity_Aligned = [0, 0, 0]
global IGNS_GPS_Velocity
IGNS_GPS_Velocity = [0, 0, 0]
global IGNS_Kalman_out
IGNS_Kalman_out = numpy.array([])
global IGNS_velocity
IGNS_velocity = [0, 0, 0]
global IGNS_Q_Variance
```

```
IGNS_Q_Variance = [0.1, 0.1, 0.1, 0, 0, 0, 0.02, 0.02, 0.02]
global IGNS_R_Variance
IGNS_R_Variance = [300000, 300000, 600000, 500, 500, 0, 0, 0, 0]
#Task System
global NavTask
NavTask = None
global NavCancelTask
NavCancelTask = False
class WayPoint:
    def __init__(self, long, lat, rad):
        self.long = long
        self.lat = lat
        self.rad = rad
    def getLocal(self, refrenceLo=GPS_Ref[0], refrenceLa=GPS_Ref[1]):
        refLo = refrenceLo
        refLa = refrenceLa
        lon1 = refLo
        lat1 = refLa
        lon2 = self.long
        lat2 = self.lat
        dx = (lon1 - lon2) * 40000 * math.cos((lat1 + lat2) * math.pi / 360) / 360 * 1000
        dy = (lat1 - lat2) * 40000 / 360 * 1000
        return dx, dy
    def distenceTo(self, posX, posY):
        targetX, targetY = self.getLocal()
        dx = posX - targetX
        dy = posY - targetY
        distence = math.sqrt(pow(dx, 2) + pow(dy, 2))
        return distence
    def isInRange(self, posX, posY):
        distence = self.distenceTo(posX, posY)
        return False#distence <= self.rad
    def getHeading(self, posX, posY):
        x, y = self.getLocal()
        heading = math.atan2(y - posY, x - posX)
        heading = math.degrees(heading) + 180
        print(heading)
        return heading
from mpu6050 import mpu6050
sensor = mpu6050(0x68)
accelerometer_data = sensor.get_accel_data()
print(accelerometer_data)
```

ÖZGEÇMİŞ

Evren BAŞER

ÖĞRENİM BİLGİLERİ

Yüksek Lisans 2018-2022	Akdeniz Üniversitesi Fen Bilimleri Enstitüsü, Tarım Makinaları ve Teknolojileri Mühendisliği Bölümü, Antalya
Lisans 2007-2014	Süleyman Demirel Üniversitesi Teknik Eğitim Fakültesi, Elektronik ve Bilgisayar Eğitimi Bölümü, Isparta

MESLEKİ VE İDARİ GÖREVLER

Yazılım Geliştiricisi 2022 – Devam Ediyor.	Turistik Hava Taşımacılık A.Ş.
Yazılım Mühendisi 2019-2022	İşlem Bilgisayar Limited Şirketi, Antalya
Yazılım ve 3 Boyut Modelleme Eğitmeni 2014-2019	Antalya Büyük Şehir Belediyesi Antalya İl Milli Eğitim Müdürlüğü Başaran Bilişim Akademisi Atılım Akademi