T.C.
KONYA TEKNİK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

HİBRİT ARAÇLAR İÇİN CAN BUS HABERLEŞME BİRİMİ İLE ARAÇ KONTROL SİSTEMİ VE LINUX TABANLI GUI TASARIMI

MURAT TOPUZ

YÜKSEK LİSANS TEZİ

Elektrik Elektronik Mühendisliği Anabilim Dalı

Aralık - 2021
KONYA

TEZ KABUL VE ONAYI

Murat TOPUZ tarafından hazırlanan “Hibrit Araçlar İçin CAN Bus Haberleşme Birimi İle Araç Kontrol Sistemi ve Linux Tabanlı GUI Tasarımı” adlı tez çalışması …/…/… tarihinde aşağıdaki jüri tarafından oy birliği / oy çokluğu ile Konya Teknik Üniversitesi Lisansüstü Eğitim Enstitüsü Elektrik Elektronik Mühendisliği Anabilim Dalı’nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Jüri Üyeleri	**İmza**
Danışman Doç. Dr. LEVENT SEYFİ	……..
Üye Dr.Öğr.Üyesi AYŞE ELİF CANBİLEN	……..
Üye Dr.Öğr.Üyesi MUHAMMED FAHRİ ÜNLERŞEN	……..

Yukarıdaki sonucu onaylarım.

Prof. Dr. Saadettin Erhan KESEN
Enstitü Müdürü

TEZ BİLDİRİMİ

Bu tezdeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

MURAT TOPUZ

16.12.2021

ÖZET

YÜKSEK LİSANS TEZİ
HİBRİT ARAÇLAR İÇİN CAN BUS HABERLEŞME BİRİMİ İLE ARAÇ KONTROL SİSTEMİ VE LINUX TABANLI GUI TASARIMI

MURAT TOPUZ

Konya Teknik Üniversitesi
Lisansüstü Eğitim Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. LEVENT SEYFİ

2021, 55 Sayfa

Jüri
Danışman Doç. Dr. LEVENT SEYFİ
Üye Dr.Öğr.Üyesi AYŞE ELİF CANBİLEN
Üye Dr.Öğr.Üyesi MUHAMMED FAHRİ ÜNLERŞEN

Günümüz otomobillerinde elektronik teknolojisi öylesine ilerlemiş ve önem kazanmıştır ki; neredeyse araç içi sistemlerin %70'ini elektronik donanımlar oluşturmaktadır. Bununla beraber modern otomobillerin konsollarının uçak kokpitlerini aratmayacak düzeyde gelişmiş tasarıma sahip olduğu gözlenmektedir. Otomobillerde onlarca sensörün kullanımı başta sürücü ve yolcu güvenliğini sağlamakla birlikte, konforu da üst düzeye çıkartmaktadır. Sensörlerden alınan bilgi ile otomobiller arıza ve hataları da sürücüye bilgi amaçlı göstermektedir. 70'ten fazla elektronik kontrol ünitesine (ECU) sahip günümüzün modern araçları onlarca sensörden aldığı bilgiler ile kullanıcıların farkına bile varamadığı birkaç milisaniye içerisinde on binlerce karar vermektedir. Sensörlerden elde edilen bilgilerin ECU'ya iletilmesinde kullanılan yüzlerce metre kablo demeti karmaşa oluşturmaktadır. Teknolojinin oldukça hızlı ilerlediği çağımızda yazılımla haberleşme ve ağ iletimi bu karmaşayı azaltmaktadır.

Bu tez çalışmasında, teknolojik gelişimi hızla ilerleyen elektrikli ve hibrit araçlar için oluşturulan araç içi ağ ile sensörlerden CAN Bus haberleşme birimi vasıtasıyla alınan bilgilerin, tasarlanan gösterge paneli sayesinde kullanıcıya bilgi verilmesi ve işletilmesi hedeflenmiştir. Haberleşme birimi olarak CAN Bus, ekran tasarımı için Qt-Qml, sensörler ve haberleşme yazılımı için C-Python ve işletim sistemi olarak Linux sistem kullanılmıştır.

Anahtar Kelimeler: Araç kontrol sistemi, CAN Bus, Elektrikli ve Hibrit Araçlar, Linux, Python, Qt-Qml GUI Tasarımı

ABSTRACT

MS THESIS

LINUX BASED GUI DESIGN AND VEHICLE CONTROL SYSTEM WITH CAN BUS COMMUNICATION UNIT FOR HYBRID VEHICLES

MURAT TOPUZ

Konya Technical University
Institute of Graduate Studies
Department of Electrical and Electronics Engineering

Advisor: Assoc.Prof.Dr. LEVENT SEYFİ

2021, 55 Pages

Jury
Assoc.Prof.Dr. LEVENT SEYFİ
Asst.Prof.Dr. AYŞE ELİF CANBİLEN
Asst.Prof.Dr. MUHAMMED FAHRİ ÜNLERŞEN

In today's cars, electronic technology has advanced and gained so much importance that; Almost 70% of in-vehicle systems are electronic equipment. However, it is observed that the consoles of modern cars have an advanced design that looks like airplane cockpits. The use of dozens of sensors in automobiles not only ensures driver and passenger safety, but also increases comfort. With the information received from the sensors, the cars show malfunctions and errors to the driver for information purposes. Today's modern vehicles, which have more than 70 electronic control units (ECU), make tens of thousands of decisions within a few milliseconds, which the users do not even realize, with the information they receive from tens of sensors. Hundreds of meters of wiring harnesses used to transmit the information obtained from the sensors to the ECU creates confusion. In our age where technology is advancing quite rapidly, communication and network transmission with software reduce this complexity.

In this thesis, it is aimed to inform the driver and operate the information received from the sensors via the CAN Bus communication unit, with the in-vehicle network created for electric and hybrid vehicles, whose technological development is rapidly advancing, thanks to the designed instrument panel. CAN Bus was used as the communication unit, Qt-Qml was used for the screen design, C-Python was used for the sensors and communication software and Linux system was used as the operating system.

Keywords: CAN Bus, Electric and Hybrid Vehicles, Linux, Python, Qt-Qml GUI Design, Vehicle control system,

ÖNSÖZ

Bu yüksek lisans tezinin hazırlanmasında başta moralimi yükselten, gayretimi artıran manevi desteğini esirgemeyen sevgili eşim Rabia TOPUZ'a, beni sabırla bekleyen sevgili kızım Lina TOPUZ'a, her zaman yanımda olan annem ve babama ve öğrenim hayatım boyunca emeği geçen tüm hocalarıma sonsuz teşekkürü bir borç bilirim.

Yardımlarını hiçbir zaman esirgemeyen ve değerli bilgilerini benimle paylaşan tez danışmanım Sn. Doç.Dr. Levent SEYFİ'ye ve sevgili arkadaşım Bekir Can DAL'a teşekkürlerimi, sevgi ve saygılarımı sunarım.

Murat TOPUZ
KONYA-2021

İÇİNDEKİLER

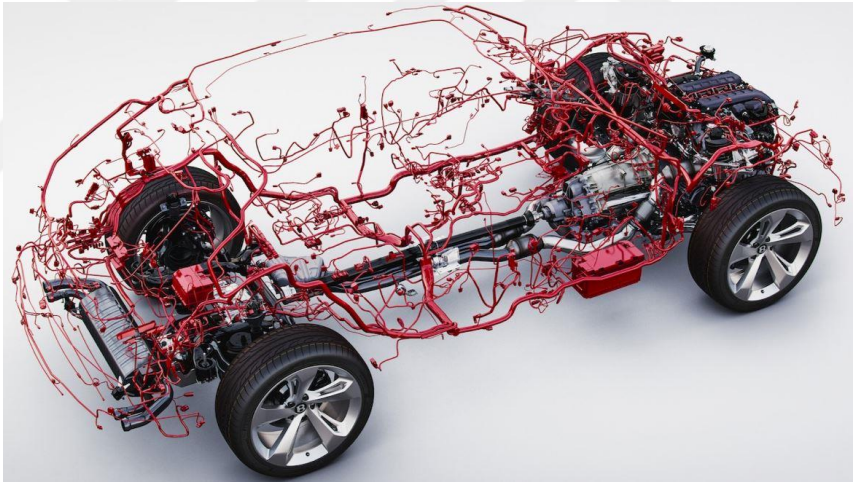
KISALTMALAR

CAN	: Controller Area Network
ECU	: Electronic Control Unit
PCB	: Printed Circuit Board
GUI	: Graphical User Interface
OBD	: On-Board Diagnostics
TCU	: Transmission Control Unit
ABS	: Anti-lock Braking System
PLC	: Programmable Logic Controller
BCM	: Body Control Module
D2B	: Domestic Digital Bus
RF	: Radio frequency
TFT LCD	: Thin-Film-Transistor Liquid-Crystal Display
CSMA/CD	: Carrier Sense Multiple Access with Collision Detection
CRC	: Cyclic Redundancy Check
ACK	: Acknowledge
AFDX	: Avionics Full-Duplex Switched Ethernet
HDMI	: High Definition Multimedia Interface

1. GİRİŞ

Otomobiller ilk hareket kabiliyetini 19. yüzyılda buharın enerji kaynağı olarak endüstride kullanılmasıyla birlikte kazanmıştır (Harrison, 2006). Ardından daha da geliştirilen otomobiller insan ve yük taşımacılığında ana ulaşım aracı olarak kendini kabul ettirmiş ve kanıtlamıştır.

Bir zamanlar, otomobildeki tek elektronik cihaz bir araç radyosu iken şimdilerde ise bir otomobilde gördüğümüz neredeyse her şey bir elektronik devre bulundurmaktadır (Yörükoğulları ve ark., 2013). İçerisinde bulunduğumuz yüzyılda, araçlarda kullanılan elektronik sistemlerin ve fonksiyonlarının kullanımı büyük artış göstermektedir (Tuncay ve Üstün, 2004). Farlarından sileceklerine, camlarından gaz pedalına ve hatta el frenine kadar gördüğümüz hemen her şey bu anlamda sıralanabilinir. Şekil 1.1'de bir otomobilin içerisinde bulundurduğu kablolama ağı görülebilmektedir.

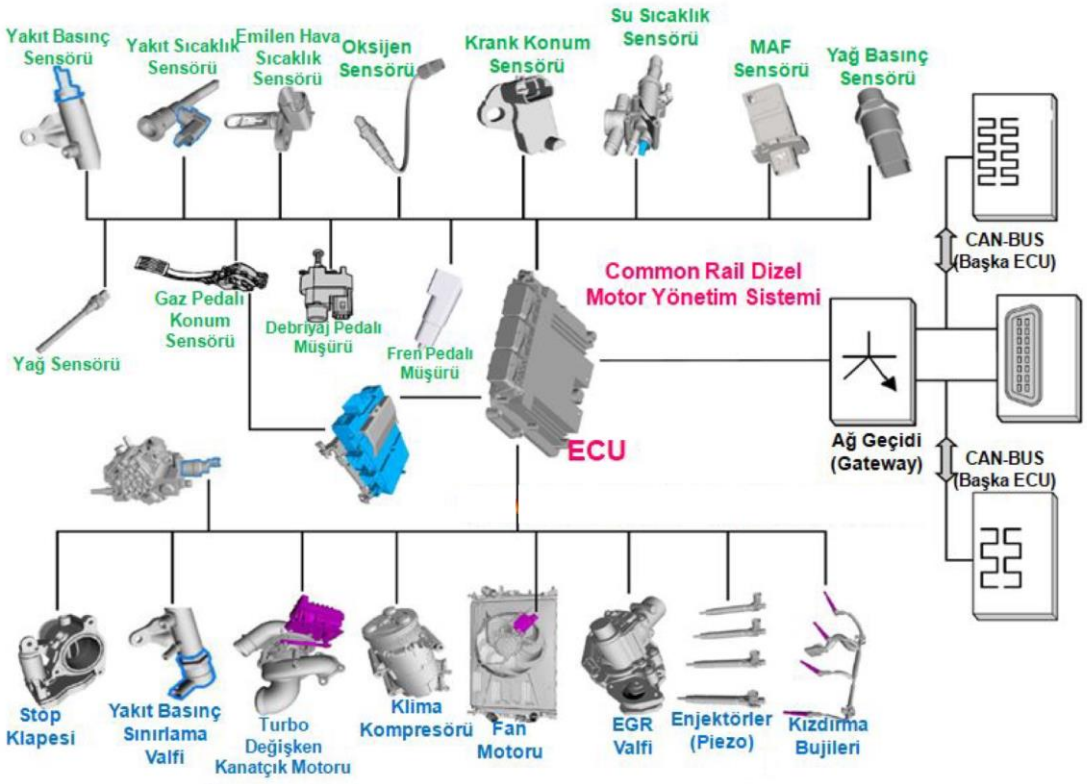


Şekil 1.1. Araç içi kablolama görüntüsü

Günümüzde "otomotiv elektroniği" ifadesi karmaşık bir ağ oluşturan, gelişmiş algılayıcıları, kontrol birimlerini, güç elektroniği devrelerini ve aktuatörleri ve bunlarla bütünleşmiş mekanik hareket sistemlerini içermektedir (Tuncay ve Üstün, 2004). Hemen hemen herkesin duyduğu elektronik birimler arasında; elektronik kontrol sistemi (Electronic Control Unit ECU), Şanzıman kontrol ünitesi (Transmission Control Unit TCU), Kilitlenme Karşıtı Frenleme Sistemi (Anti-lock Braking System ABS) ve Karoseri Kontrol Modülü (Body Control Module BCM) modülleri akla gelmektedir. Bu

modüller bilgilerini (hız, sıcaklık, basınç…) sensörlerden araç içi ağ vasıtasıyla almaktadır ve işlemektedir.

Aracın hatasız çalışması için modüllerin kendi aralarında veri alışverişi yapması gerekmektedir. Modüller sensörlerden aldığı bilgileri işleyerek diğer modüller ile iletişime geçer ve gerekli olan aktüatörleri devreye alıp devreden çıkartabilir (Kalaycı, 2015). Sinyal kolunun aktif edilmesiyle gerekli olan ön ve arka sinyal lambalarının devreye girmesi ve bu işlevin ekranda gösterilmesi örnek olarak gösterilebilir. Şekil 1.2'de araç kontrol sistemi ve sensörlerin bağlantıları görülmektedir.



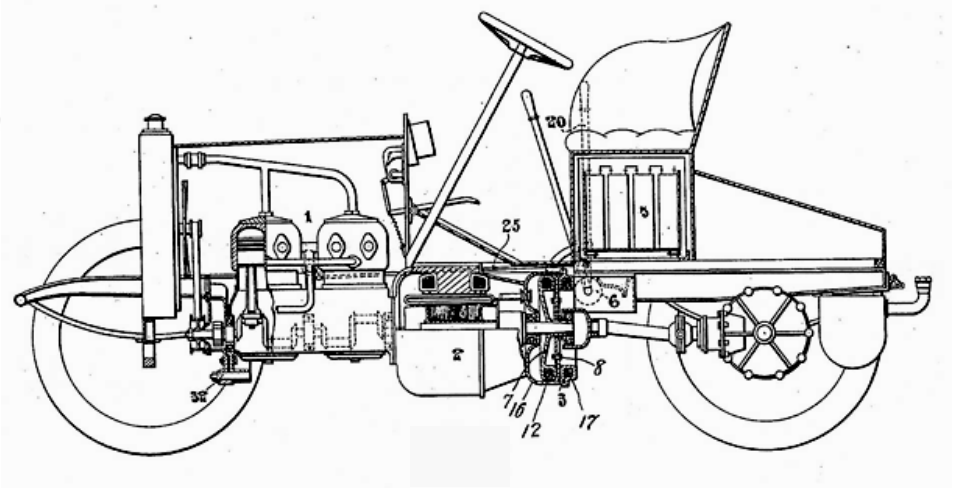
Şekil 1.2. Araç kontrol sistemi ve sensörlerin görüntüsü

Günümüz otomobillerinde elektronik ve haberleşme öylesine büyük bir önem kazanmıştır ki araç içerisindeki sistemlerin yüzde 70'ini elektronik aksam oluşturmaktadır. Yalnızca yüzde 30'u mekanik birleşenlerdir (Acar, 2019). Otomobillerde elektronik sistemleri zorlayan faktörler olarak daha fazla güvenlik faktörü, doğaya daha az salınım, daha az yakıt gösterilebilir (Fil, 2019). Bu bakımdan otomobillerde elektronik bileşenlerin kullanımı, sıradan elektronik sistemlerden farklı olarak birçok yeniliği de beraberinde getirmek zorundadır (Acar, 2019). Bununla birlikte, ilerleyen elektrik-elektronik teknolojileri sayesinde, daha hafif ve daha çevre

dostu araçlar geliştirilebilmekte, otomotiv teknolojisinde daha yüksek güvenilirlik ve daha üstün kontrol değerleri elde edilebilmektedir (Tuncay ve Üstün, 2004).

Her geçen gün gelişen ve değişen teknoloji ile ihtiyacı ve kullanımı artan, çevreye zarar vermeyen, yüksek verimli, elektrikli ve hibrit araçlar daha fazla ön planda olmaya başlamıştır (Çağışlar, 2018).

19. yüzyılın başlarında elektrik motorunun icat edilmesinin ardından 1835 yılında Hollanda'da Prof. Stratingh tarafından geliştirilen ilk elektrikli araç ile ulaşım sektöründe elektrikli tahrik fikri oluşmaya başlamıştır (Erçin, 2020). Batarya teknolojilerinin gelişmemiş olması nedeniyle ortaya çıkan düşük menzil sorunlarından dolayı 1935-1960 yılları arasında neredeyse hiç elektrikli araç çalışması ortaya konulmamıştır. 1990'dan sonra batarya teknolojilerinde meydana gelen gelişmelerle ve içten yanmalı motorların oluşturduğu egzoz gazlarının çevreye verdiği zararların farkındalığının oluşmaya başlamasıyla birlikte birçok üretici firma tarafından elektrikli ve hibrit araçların geliştirilmesine başlanmıştır (Erçin, 2020). 1905 yılında Henri Pieper tarafından patent başvurusu yapılan hibrit araç Şekil 1.3'te görülebilmektedir.



Şekil 1.3. Henri Pieper'ın 1905 yılında patent başvurusu yaptığı hibrit araç

Günümüzde otomobil üretiminin önemli bir oranının artık hibrit veya elektrikli olduğu göze çarpmaktadır. Hibrit araçlarda içten yanmalı motor ile elektrik motoru birlikte tahrik sağlamaktadır. Elektrik motoru sürüşün şartına göre kimi zaman jeneratör gibi de çalışarak araçta bulunan bataryaları şarj ederken kimi zaman ise içten yanmalı motora tahrik desteğinde bulunabilmektedir. Hibrit araçlar, elektrik motorunun ve içten yanmalı motorun mekanik bağlantısına göre paralel, seri ve seri-paralel olarak 3 ayrı gruba ayrılmaktadır (Kerem, 2014; Ünlü ve ark., 2003).

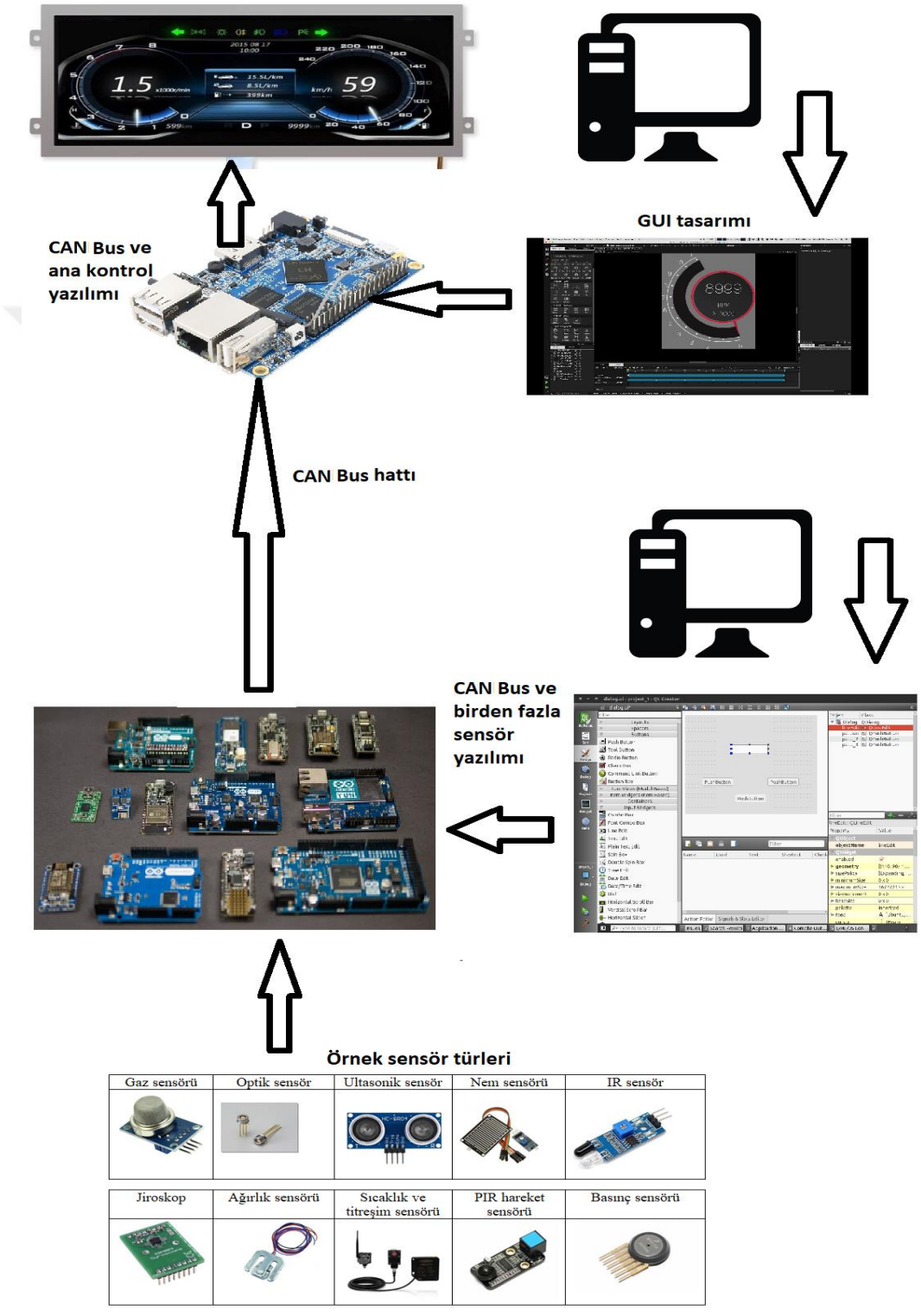
1.1. Tezin Amacı

Bu tez çalışmasında hibrit veya elektrikli bir araç için araç kontrol sisteminin tasarımı yapılması planlanmıştır. Hibrit veya elektrikli motora sahip araçlarda içten yanmalı motora sahip araçlara kıyasla daha fazla kablo karmaşası söz konusu olmaktadır. Bu karmaşanın azaltılması amacıyla araç içi ağ ve haberleşme sistemleri geliştirilmiştir. Lâkin bu sistemlerin incelenmesi ve anlaşılır şekilde kullanıcıya gösterilmesi oldukça zahmetli ve maliyetledir. Bu nedenle daha basit ve anlaşılır bir GUI tasarımının Linux sistem kullanılarak geliştirilmesi ön plana çıkmaktadır. Linux sistem sayesinde araç yazılım güvenliği bir üst seviyeye çıkarılacaktır. Ayrıca anlık olarak hava durumu bilgisi, yol durumu bilgisi hatta son dakika haberlerine ulaşabilmek yalızca bir kaç saniye içerisinde mümkün olabilecektir. Var olan sistemlerde bu işlemler birkaç farlı ürün ile yapılırken bu tez çalışmasında hedeflenen sistem sayesinde medya da dâhil olmak üzere tek noktadan kontrol edilmesi planlanmaktadır. Böylece, maliyeti düşürmekle birlikte araç elektroniğindeki karmaşa da azaltılmış olacaktır. Şekil 1.4'te farklı ECU görülebilmektedir.

Şekil 1.4. Farklı elektronik kontrol ünitesi (ECU) görünümleri

Araç kontrol sistemi için; aktuatörler, araç içi ağ ve gösterge paneli tasarımı yapılmıştır. Tüm sistem Linux tabanlı işletim sistemi üzerinde gerçekleştirilmiştir. Araç içi ağ olarak CAN Bus haberleşme birimi, ekran tasarımı Qt-Qml ve Python yazılım dilleri, aktuatörlerin tasarımı altium designer üzerinde ve yazılımı C programlama dili kullanılarak gerçekleştirilmiş olacaktır. Şekil 1.5'de geliştirilecek sistemin blok diyagramı görülmektedir. Hibrit bir araçta (elektrikli de olabilir) algılama ve kontrol

etme sistemlerinin tamamının en yüksek verimle çalıştırılabilmesi için gömülü sistem tasarımı, yazılımı, grafiksel kullanıcı arayüzü (Graphical User Interface GUI) sistem gerçekleştirilmesi, CAN Bus haberleşme birimi gerçekleştirilecek ürün ile en iyi hale getirilecektir.



Şekil 1.5. Sistem blok diyagramı

2. KAYNAK ARAŞTIRMASI

Durgun, 2019 yılında gerçekleştirdiği yüksek lisans tez çalışmasında araç kontrol sisteminin tasarımı, yazılımının yapılması, araç içindeki diğer aygıtlardan bilgi alınması ve verilerin değerlendirilmesi işlemlerini gerçekleştirmiştir. C#'da gerçekleştirdiği arayüzde Stm32f407 işlemcisini kullanarak aldığı verilerin görüntülenmesini sağlamıştır. Haberleşme birimi olarak CAN Bus kullanmıştır.

Öztürk, 2010 yılında gerçekleştirdiği tez çalışmasında mesaj içerikli haberleşmenin getirdiği kablolama kolaylığı, bunun getirdiği düşük maliyet, hata tespit rutinlerinin çok güçlü olmasının getirdiği güvenilirlik, yeni ünitelerin eklenmesi için sistemde değişime gitmemenin getirdiği kolaylık, saniyede 10000 mesaj iletimi sağlaması gibi CAN Bus'ın otomotiv endüstrisinde haberleşme ağları için önemli avantajlarının olduğundan bahsetmiş ve hidrojen hibrit otobüsler için uygulamasını gerçekleştirmiştir.

Kalaycı, 2015 yılında yapmış olduğu tez çalışmasında Programlanabilir Mantıksal Denetleyici(Programmable Logic Controller PLC) ile CAN Bus haberleşme protokolünü kullanarak özellikle üniversiteler ve meslek liseleri gibi, birden fazla binası bulunan eğitim kurumlarında, kullanım saatleri dışında açık unutulan aydınlatma üniteleri veya bekleme konumunda bırakılan TV, akıllı tahta, bilgisayar, projeksiyon v.b eğitim materyallerinin yol açtığı enerji kayıplarını önlemeyi ve etkin bir bina enerji yönetimi sistemi kurarak enerji tasarrufu sağlamayı hedeflemiştir. PLC, operatör paneli ve CAN Bus haberleşme protokolü kullanılarak enerji tasarrufu sağlayan, hızlı ve kararlı çalışan bir enerji yönetim sistemi gerçekleştirmiştir.

Bulgu, 2010 yılında yazmış olduğu yüksek lisans tez çalışmasında hibrit araç kavramından, çeşitlerinden ve hibrit araç bileşenlerinden bahsetmiştir. Tekerlek motorların seri hibrit araca getirdiği değişiklikleri gözlemlemek amacıyla konvansiyonel araç, seri hibrit elektrikli araç ve tekerlek motorlu seri hibrit elektrikli araçların modellemesi MATLAB Simulink ortamında gerçekleştirmiştir. Aktarma organlarının verimsizliğinin yakıt tüketimine etkisini gözlemlemiştir. Tekerlek motorlu seri hibrit araçların iki eksenli çalışmasını önermiş ve bu çalışmalar sırasında çekiş kontrol, ABS ve fren dengesi gibi uygulamaların geliştirilmesi gerektiği kanısına varmıştır.

Arslan, ve ark., 2016 yılında yaptıkları çalışmada mikroişlemci içeren bir gömülü sistem için çoklu ortam uygulaması gerçekleştirilmişlerdir. Tasarlanan sistemin

Linux işletim sistemi ile ses dosyası çalma, video oynatma, dosyadan veri okuma ve kaydetme, ekran görüntüsü alma, diske kaydetme gibi birçok özelliğe sahip olmasını sistemin yazılımsal ve donanımsal tasarımını gerçekleştirmişlerdir. Çoklu ortam işlemleri, geliştirilen uygulamalar ve kullanıcı kütüphaneleri ile sağlanmıştır. Sonraki aşamada, ücret toplama sistemlerinde kullanılan sürücü bilgisayarına entegre cihazlarda bu sistemin kullanılabileceğinden bahsedilmiştir. Ayrıca, sürücü bilgisayarlarına özel radyo frenkası (Radio frequency RF) kart okuma, yolcu bilgisi girme, merkez ile haberleşme, yolcuları sesli görüntülü olarak bilgilendirme gibi işlemlerin tasarlanan sistem ile gerçekleştirilebileceğinden bahsedilmiştir.

Özbey 2020 yılında yazmış olduğu elektrikli araçlar için kablosuz şarj sistemi tasarımı ve optimizasyonu konulu tez çalışmasında, sistem optimizasyonun ardındaki rezonans çeviricisinin çizimini Altium Designer devre tasarım programında gerçekleşmiştir.

Yıldırım, 2020 yılında gömülü sistemler için bilgisayar tabanlı bir GUI tasarım aracı geliştirilmesi adlı tez çalışmasında Arçelik InterACT bulaşık makinesinin kontrol ve ekran kartı tasarım ve yazılımından bahsetmiştir. Seri üretime uygun özelliklerde ve düşük maliyetlerde ince tabakalı transistör - sıvı kristal ekranlı (Thin-Film-Transistor Liquid-Crystal Display TFT LCD) ürün tasarımı yapılmış ve tez çalışması kapsamında geliştirilen GUI kütüphanesi bu donanımda çalıştırılmıştır. Tez çalışmasında TFT ekran, dokunmatik ekranlardan, ekran sürücülerinden ve donanım yapısından bahsedilmiştir.

Urkun, 2020 yılında elektrikli araç için bir kontrol sistemi geliştirdiği tez çalışmasında rejeneratif fren sistemli bir elektrikli araç modeli üzerinde aracın hız kontrolünü sağlamak amacıyla bir kontrolcü geliştirmiştir. Aracın örnek hız profilleri ile hızlanma ve yavaşlama ihtiyacı göz önüne alınarak geliştirilen kapalı çevrim bulanık mantık kontrolörün genel performansı ve aracın enerji tüketimine olan etkisi ölçülmüştür. Simülasyon sonuçlarına bakıldığında, tez kapsamında geliştirilen kontrolcünün araç modeli üzerinde genel olarak başarılı bir performans sergilediği tespit edilmiştir.

3. MATERYAL VE YÖNTEM

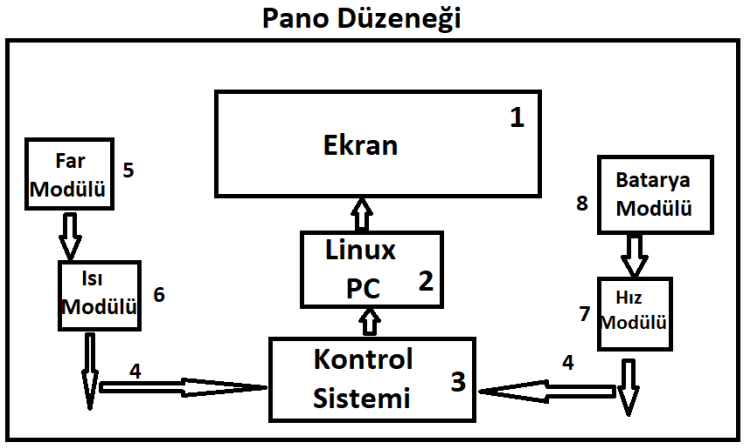
3.1. Kurgulanan Sistemin Yapısı ve Gereksinimleri

Araç kontrol sistemi, araç içerisindeki aynı veya farklı tipteki çeşitli sistemlerden veya sensörlerden almış olduğu verileri işleyip gerekli olan aktüatörleri devreye alıp devreden çıkartan ve diğer kontrol birimlerine bu verileri ileten ünitedir. Motor kontrol sistemi, güç kontrol birimi, kullanıcı ekranı, iklimlendirme sistemi, batarya yönetim sistemi ve gövde yönetim sistemi gibi sistemler araç kontrol sisteminin haberleştiği başlıca sistemlerdir (Durgun, 2019).

Hibrit ve elektrikli araç teknolojisinde kurulan topolojinin başlıca fonksiyonları; yakıt verimini arttırmak, egzoz emisyonunu minimuma indirmek, araç donanımlarını gözlemlemek ve korunmaktır. Bu donanımlar batarya şarj durumunu, batarya, elektrik motoru ve içten yanmalı motorun sıcaklığını kontrol ederek aşırı ısınmalarının engellenmesi için çalışmaktadır. Böylece bir bataryanın gerekli şarj ve ısı kontrolünü yapılmakla birlikte bataryanın hatasız ve uzun süre kullanımınıda garanti etmektedir (Sefik, 2017).

Araç kontrol sistemi yazılım ve donanım kısımlarından oluşur. Donanım kısmı mikrodenetleyici, besleme devresi, haberleşme devresi gibi kritik devrelerden meydana gelmektedir. Yazılım kısmında ise verilerin alınması, işlenmesi ve haberleşme hattına gönderilmesi gibi başlıca döngüler bulunmaktadır (Durgun, 2019).

Tez çalışmasında gerçekleştirilen sistemin proje aşamaları devre tasarımının oluşturulması, malzeme tedariği, sistem yazılımı ve sistemin bütün olarak bir araya getirilmesi şeklinde ifade edilebilir. Kurulumu gerçekleştirilen sistemin blok diyagramı Şekil 3.1'te görülebilmektedir.



Şekil 3.1. Kurulumu gerçekleştirilen sistemin blok diyagramı

Gerçekleştirilen sistemde bulunması gereken bileşenler aşağıda listelenmiştir.

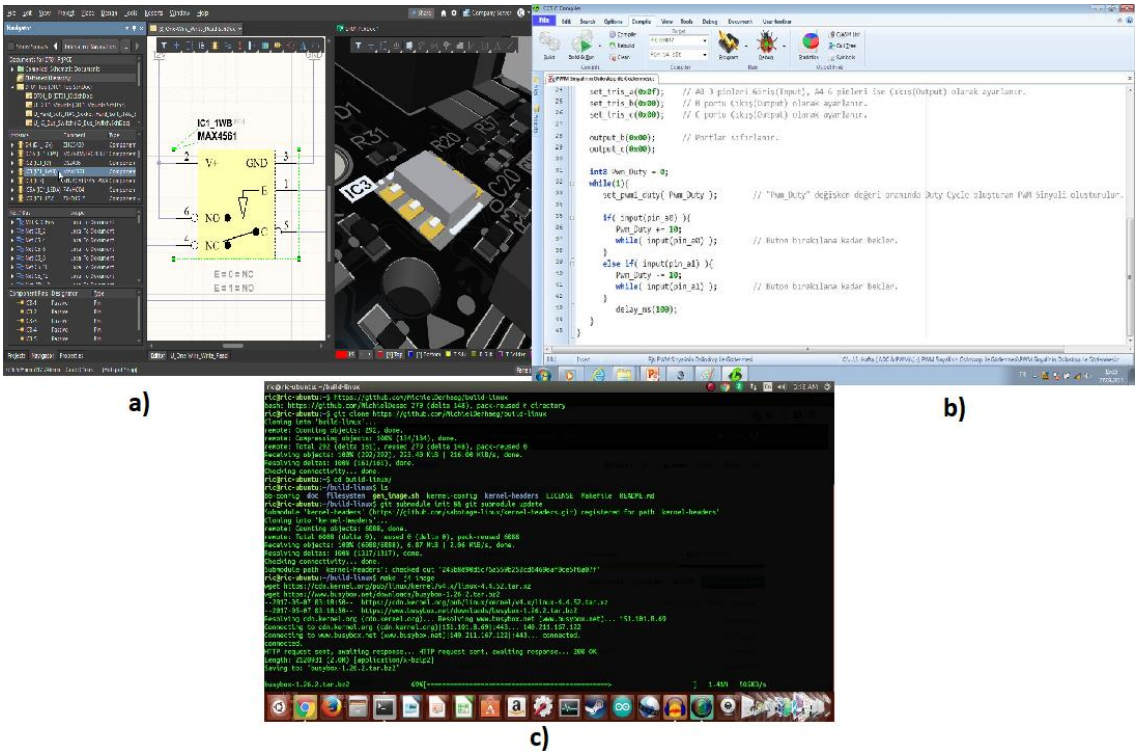
1. Modüllerden alınan bilgileri kullanıcıya aktarmak için bir ekrana ihtiyaç vardır.
2. Sistemin tüm verilerini işleyip ekrana görüntü aktaracak ve gerektiğinde ağa bağlanarak bilgi alış-verişini yapacak bir Linux tabanlı PC gereksinimi vardır.
3. Modüllerden alınan bilgilerinin bir merkezde toplanıp gerekli bir biçimde kontrol edilmesi ve PC tarafına bilgilerin gönderilmesi için bir ana kontrol ünitesine ihtiyaç vardır.
4. Tüm modüllerin ve ana kontrol ünitesinin haberleşmesi için bir CAN Bus haberleşme hattı oluşturulmasına ihtiyaç vardır.
5. Farlar, aydınlatmalar ve sinyallerin kontrolü için bir modüle ihtiyaç vardır.
6. Sıcaklık ölçüm ve kontrolü için bir modüle ihtiyaç vardır.
7. Gaz pedalı, fren pedalı ve bunları simüle edecek bir cihaz modülüne ihtiyaç vardır.
8. Batarya yönetimi için bir modüle ihtiyaç vardır.

Burada amaç, araç için tüm gereksinimlerin oluşturulması değil sistemin mantığının kavranması adına bir kontrol panosunun oluşturulmasıdır. Böylece bir araç kontrolü için en temel yapı taşı olan araç kontrol sisteminin sistem parçalarının özel olarak oluşturulup araca özgü ve maliyeti daha uygun şekilde sistemin kurulması sağlanmaktadır. Geliştirilecek sistemin, bir deney düzeneği veya eğitim seti olacak nitelikte yapılması kurgulanmıştır. Bu sistem, araç kontrol sistemini simüle edecek şekilde oluşturulmuştur. Deney düzeneği üzerinde bir Linux işletim sistemine sahip gömülü bilgisayar, bu bilgisayara bağlı LCD ekran, Altium Designer üzerinde tasarlanmış kontrol modülleri, modüller arası haberleşmeyi sağlayacak olan CAN Bus haberleşme hattı ve modüllere bağlı olan sensör birimleri bulunmaktadır. Batarya yönetim sistemi, aydınlatma yönetim sistemi, gaz, fren ve motor kontrol birimleri modül olarak deney düzeneğinde bulunmaktadır.

Tezin tamamı elektronik devre tasarımı, elektronik devre yazılımları ve Linux bilgisayar yazılımı olmak üzere 3 ana parçadan oluşmaktadır.

Elektronik devre tasarımları Altium Designer üzerinde profesyonel bir şekilde yapılmıştır. Tasarlanan bu devreler araçlarda bulunan sensörlerden alınan bilgileri

eksiksiz bir biçimde ana kontrol ünitesine CAN Bus aracılığı ile iletilmesine yardımcı olmaktadır. Bu devreler veya sistemler birden fazla olacak şekilde tasarlanmıştır. Kimi sistem far kontrolü için kullanılırken kimi sistemler batarya yönetimi için çalışma sağlamaktadır. Şekil 3.2'de Altium Designer devre tasarımı görünümü, CCS PIC işlemci programlama arayüzü ve linux sistem komut satırı ekranı görülmektedir.



Şekil 3.2. a) Altium Designer devre tasarımı görünümü **b**) CCS PIC işlemci programlama arayüzü **c**) Linux sistem komut satırı ekranı

Tasarlanan devrelerde bulunan işlemcilerin yazılımının, devrenin amacına göre yapılması gerekmektedir. Yazılım geliştirme ortamı olarak CCS C derleyici kullanılmıştır. Yazılım dili olarak C programlama dili tercih edilmiştir. Bu geliştirme ortamında CAN Bus yazılımı, derleyicinin kendi kütüphanelerinden yararlanılarak geliştirilmiştir.

Linux işletim sisteminin çalıştırılması için Raspberry Pi, bilgi gösterimi için bir ekran kullanılmıştır. Bu ekran kullanıcıya araç ile ilgili bilgileri, işlenen yazılım vasıtasıyla görsel olarak aktarmaktadır. Geliştirilen sistemde Python, QT ve QML yazılım dilleri kullanılarak hem GUI tasarımları hem de sistemin çalıştırılmasında gereksinim duyulan yazılım gerçekleştirilmiştir.

3.2. Ekran

İnternet araçılığıyla ihtiyaca yönelik bir LCD ekran temin edilmiştir. Ekranın fotoğrafı Şekil 3.3'te görülebilmektedir. Fiyat-performans kıyası yapıldığında bu çalışma için ideal bir ürün olduğu söylenelebilir.

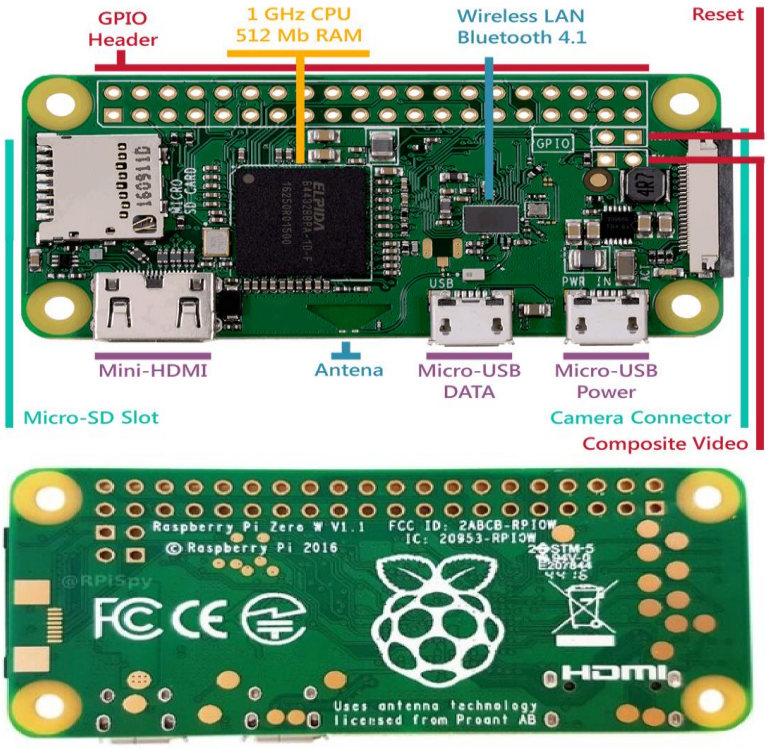
Şekil 3.3. Kullanılan ekranın görseli

8.8 inç, 1920 x 480 piksel IPS çözünürlüklü ürün HDMI girişi sayesinde projede kullanışlı bir yapı ortaya koymaktadır. Ekran tedariği için gerekli link EK-1'de belirtilmiştir.

3.3. Raspberry Pi

Kredi kartı büyüklüğünde olan Raspberry Pi, hobi, eğitim veya endüstriyel kullanım için tasarlanmış düşük maliyetli ve genellikle Linux işletim sistemi desteği ile kullanılan mini bilgisayarların adıdır (Özcan, 2019). Raspberry Pi, bir bilgisayar olmasıyla birlikte üzerinde bulunan 40 adet pin kullanıcı giriş, çıkışları sayesinde fiziksel donanımların bağlantısı ile kontrol için kullanılabilecek bir tümleşik yapıdır. Herhangi bir bilgisayarda yapılabilecek hemen hemen herşeyi Raspberry Pi bilgisayarı ile yapmak mümkün olabilir. Raspberry Pi'nin bugüne kadar çıkartılmış birçok türü bulunmaktadır (Adafruit, 2018).

Bu tez çalışmasında Şekil 3.4'te görülen Raspberry Pi Zero W (Wifi destekli) modeli kullanılmıştır. Ürün tedariği için gerekli link EK-1'de belirtilmiştir. Tercih edilme sebebi olarak ürünün fiyatı, kablosuz bağlantı ve bluetooth desteği olması, 512 MB RAM ve Mini HDMI girişinin olması ön plana çıkmaktadır.



Şekil 3.4. Raspberry Pi Zero W PC görseli

Gerçekleştirilen çalışmada kablosuz ağ desteği sayesinde kolaylıkla internete bağlanılıp hedeflenen verilerin alınması sağlanmıştır. Hava durumu, trafik verisi ve çevrimiçi uygulamalar bunların en önemlileridir. Raspberry Pi üzerinde uygulama geliştirmek için bazı yazılım dillerine ihtiyaç vardır. Bu tez çalışması için Python, QT ve QML yazılım dillerinden faydalanılmıştır.

3.4. Python, QT ve QML ile GUI Tasarımı

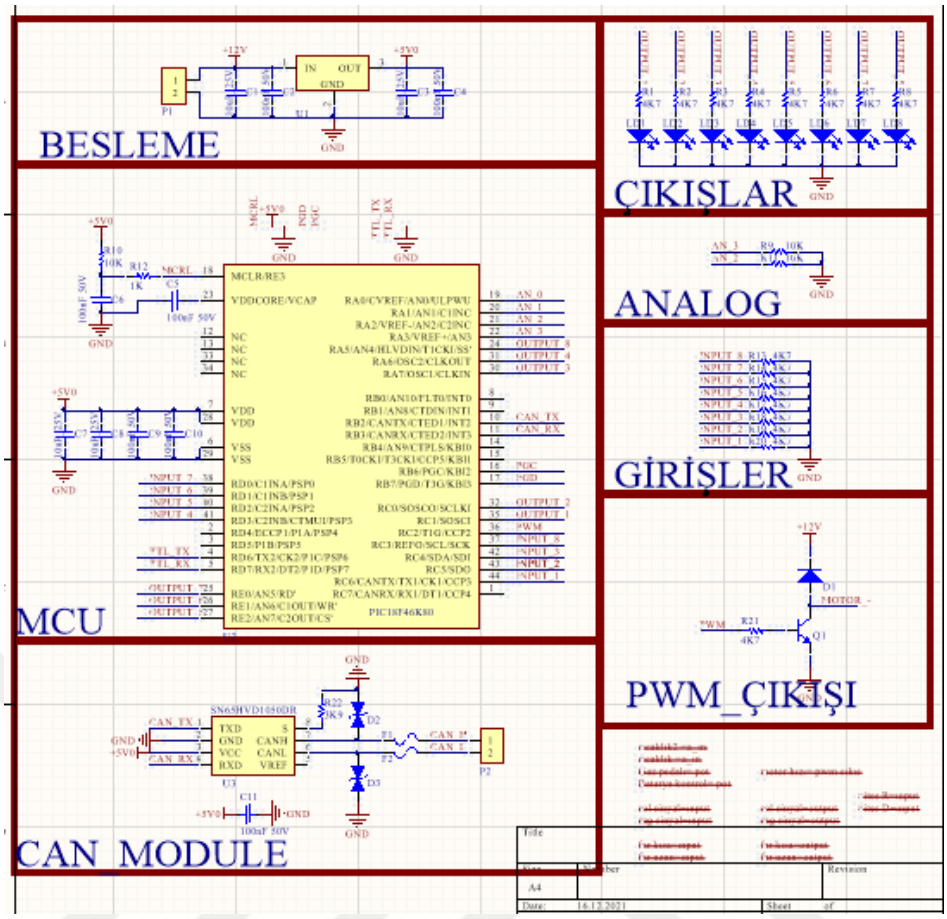
Raspberry Pi üzerinde gerekli kütüphanelerin(Bölüm 4.1.2) kurulmasıyla birlikte araç kontrol sisteminin görsel ekran tasarımları ve yazılımları Python, QT ve QML yardımıyla yapılmaktadır. Python programlama dili ile tasarlanan sensörlerin ana kontrol modülünden verilerin alınması adına bir köprü kurularak veriler QT yazılım diline aktarılmaktadır. QT platformunda işlenen veriler QML tarafından görsel olarak ekrana basılmaktadır. Şekil 3.5'de örnek bir gösterge paneli taslak çalışması görülmektedir. Sistem bir xx.py ana dosyası ile çalışmaktadır. Bu dosyaya yardımcı olarak ara yüz tasarımı için xx.qml dosyası çağırılmaktadır. Detaylı anlatım ise bölüm 4'de yapılmıştır.

Şekil 3.5. Oluşturulacak örnek gösterge paneli ilk taslak çalışması

3.5. Devre Tasarımı, Komponent Seçimi ve Tedariği

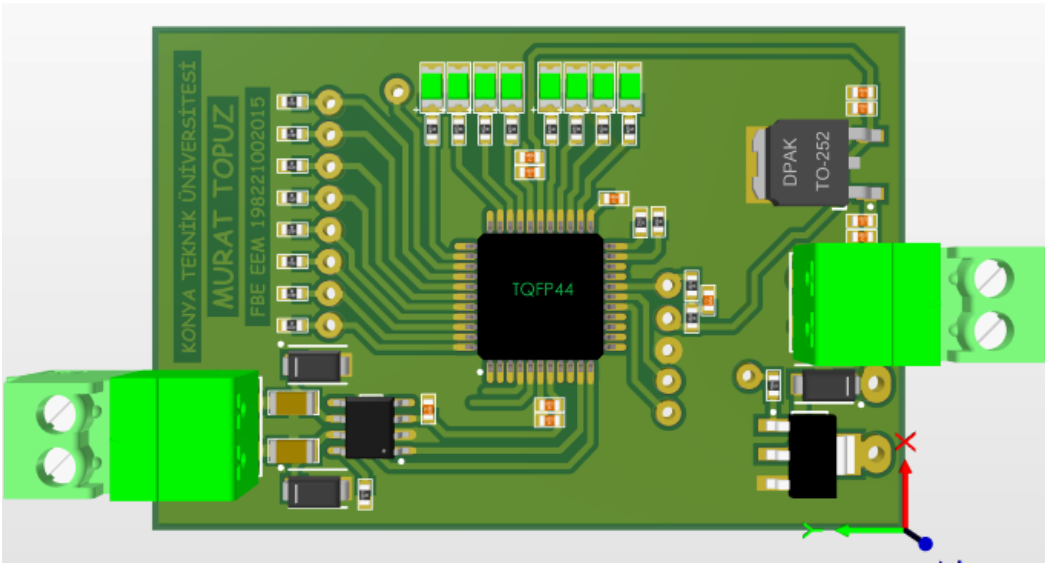
Sistem gereksinimlerine göre Altium Designer üzerinde devre tasarımı oluşturabilmek adına bir proje oluşturulmuştur. Proje içerisine şematik ve PCB dosyaları oluşturularak ilk adım atılmıştır. Şematik dosyasına devre tasarımı için gerekli komponentlerin kütüphanesi oluşturulmuştur. Gerekli komponentler ile tasarım tamamlanmıştır.

Devrede 12V besleme girişi olacağından dolayı uygun gerilim için bir voltaj regülatör devresi tasarlanmıştır. CAN Bus entegresi, ana işlemci ve diğer komponent beslemeleri için uygun olan 5V regülesi tamamlanmıştır. DC motor kontrolü, analog girişler, dijital girişler ve çıkışlar için devre blokları ve CAN Bus haberleşmesi için bir CAN Bus modülü ile devre bloğu tasarımı yapılmıştır. Yapılan şematik tasarımı Şekil 3.6'da detaylı olarak görülmektedir. Şematik ve devre tasarımı tecrübe gerektirmektedir. Komponentlerin veri dosyalarını tasarıma göre detaylı incelemek gerekebilir. Tasarımda kullanılan tüm komponentler EK-1'de gösterilen internet sitesinden tedarik edilmiştir.



Şekil 3.6. Altium Designer üzerinde yapılmış şematik tasarımı

Şematik sonrası PCB tasarımı yapılmıştır. Proje öncesi şematik tasarımı yapmak PCB tasarımı tarafında ve tasarlanan proje ürüne dönüştüğü zaman hata arıza durumunda kolaylık sağlayacaktır.

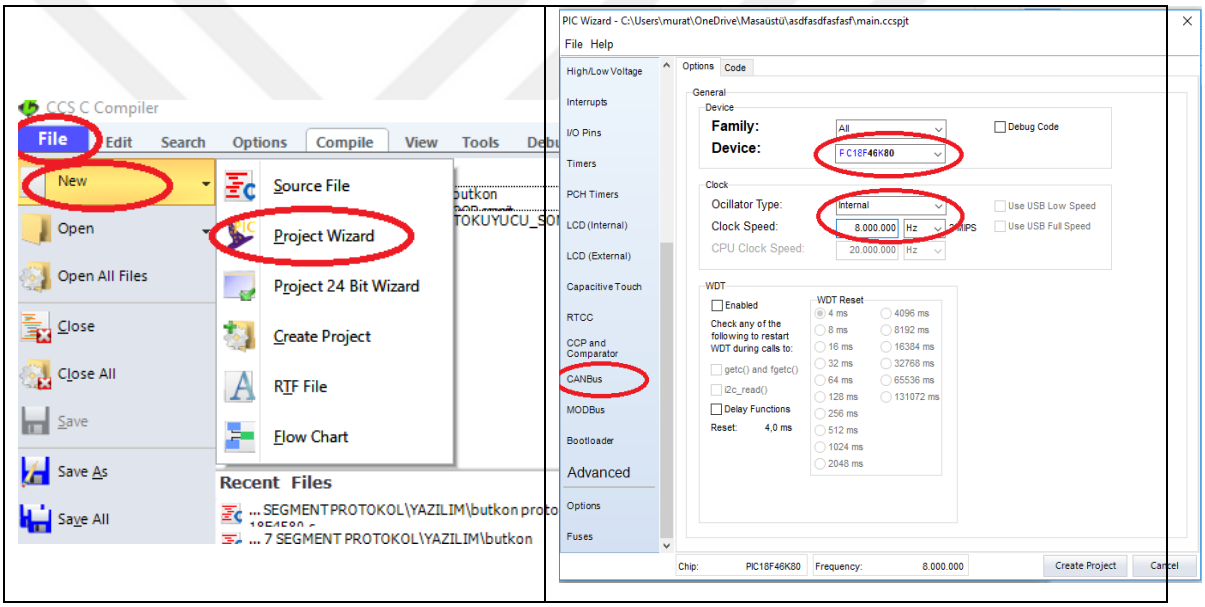


Şekil 3.7. Altium Designer üzerinde yapılmış PCB tasarımı

3.6. Tasarlanan Kontrol Sisteminin ve Sensörlerin CCS C ile CAN Bus Yazılımı

Yazılımı gerçekleştirebilmek için hem C programlama bilgisine, hem de CAN Bus bilgisine ihtiyaç vardır. Gerekli olan algoritmanın kodlanabilmesi ve CAN Bus haberleşme yazılımının derlenebilmesi adına tasarımda kullanılan PIC işlemci için yazılım derleme platformu olarak CCS C derleyicisi seçilmiştir. Kullanımı kolay, CAN Bus kütüphane desteği ve örnek desteğinin fazla olması CCS C derleyicisini ön plana çıkarmıştır.

Öncelikle şekil 3.8'de görüldüğü üzere CCS C compiler derleyicisinde proje başlatılır. File, New ve Project Wizard menülerini sırasıyla tıklayarak proje kaydetme sayfası açılır.



Şekil 3.8. CCS C Compiler proje oluşturulması

Açılan pencerede projenin kaydedileceği yer seçilerek kaydet butonuna tıklandıktan sonra açılan proje sayfasında ilk olarak kullanılacak olan işlemci türü seçilmelidir. Kullanılacak işlemci seçimi sonrası sistem kristal seçimi gerçekleştirilir ve sol tarafta bulunan CAN Bus kütüphane seçimi sonrası gerekli olan zamanlayıcı, seriport ve giriş çıkış ayarları aynı şekilde oluşturularak proje oluşturulur. Projenin oluşturulmasının ardından Şekil 3.9'da görüldüğü üzere proje içerisinde can_init() ile CAN Bus kütüphanesi çalışmaya uygun ayarlanmış olacaktır. CAN Bus veri alma (can_getd()) ve veri gönderme (can_putd()) komutları ile algoritma oluşturulmaya başlanır.

```
52  void CANRX0_isr(void) {
53      Gelen_ID_ = 0;
54      can_getd(Gelen_ID_, data_CevapGelen, len, stat);
55     // if (!(COMSTAT.txbo)) can_putd(Gelen_ID_, data_CevapGelen, len, 2, 1, 0);
56      // retrieves a message from the CAN bus storing the
57      // ID in the ID variable, the data at the array pointed to by
58      // "data', the number of data bytes in len, and statistics
59  }
60
61  void main() {
62      restart_wdt();
63      //----------------- CAN AYARLARI --------------------------------
64      can_init();
65      can_set_mode(CAN_OP_CONFIG);
66      can_set_mode(CAN_OP_NORMAL);
67      can_disable_filter(0x1fffffff);
68      //-----------------END-----------------------------------------------
69      setup_adc(adc_clock_internal);
70      setup_adc_ports(2|3);
71
72      if (COMSTAT.rx0ovfl)COMSTAT.rx0ovfl = 0; //RX DE TAŞMA VARSA REGISTERI SIFIRLIYOR
73
74      setup_timer_2(T2_DIV_BY_16, 31, 10);
75
76      enable_interrupts(INT_TIMER2);
77      enable_interrupts(INT_CANRX0);
78      enable_interrupts(INT_CANERR);
79      enable_interrupts(GLOBAL);
80
81      while (TRUE) {
82          restart_wdt();
83          if (!(COMSTAT.txbo))can_putd(data_batarya + 200, data_Gonderilen, 1, 3, 1, 0);
```

Şekil 3.9. CCS C Compiler ile CAN Bus init ve veri alma komutları

Sistemde sensörlerden bilgilerin gerekli şekilde okunarak bir adres ile eşleştirilmesi sağlanır. Mesaj adresi ile eşleşen veriler mesaj uzunluğu ile beraber CAN Bus mesajı için hazır hale gelmiş olmaktadır. Burada unutulmaması gereken husus mesaj adres uzunluğunun standart CAN için 2^{11} bit, geliştirilmiş (extended) CAN için 2^{29} bit olmasıdır. Bu çalışmada standart CAN kullanılmaktadır. Ayrıca mesaj uzunluğunun 8 bayt'ı geçmemesine dikkat etmek gerekmektedir. Hazırlanan veriler CAN Bus hattından basmak için hazırlanmış olmaktadır. Her mesaj can_getd (ID, data, len, FALSE) fonksiyonu ile CAN Bus hattından basılır. Basılan mesajlar diğer modüllerden can_putd(ID,data,len,priority,TRUE,FALSE) komutu ile okunarak gelen mesajın okunan modüle ait olup olmadığını yazılımsal olarak filtre etmek gerekmektedir. Eğer gelen mesaj modüle ait ise gerekli işlem yapılmaktadır.

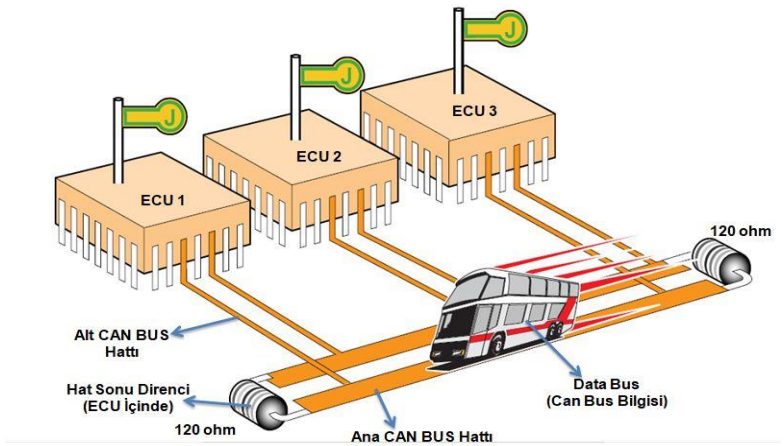
Örneğin bir sinyal modülünde işlem şu şekilde gerçekleşmektedir. Sinyal modülünde bulunan sinyal kolu bir yöne aktif edilmektedir. Aktif edilen anahtar, modül tarafından algılanır. Algılanan sinyal işlenerek haberleşme hattından basılır. CAN hattında oluşan mesaj ilgili ID'de ise gerekli modül tarafından algılanır ve yapılması gereken işlem için CAN hattından yeni bir cevap mesajı basılır. İlgili modüle gelen cevap mesajı işlenerek gerekli olan sinyalin aktif edilmesi sağlanır. CCS C ile ilgili tüm kodlar EK-2'de verilmiştir.

3.7. CAN Bus nedir?

CAN Bus, otomotiv sektöründe kullanılmak üzere geliştirilmiş, kullanışlı bir iletişim protokolüdür. CAN Bus, bu sektörde en çok bilinen haberleşme sistemidir. Otomotiv tasarımında kablo karmaşasını azaltmak için Robert Bosch tarafından 1980'lerde geliştirilmiştir. Böylece tek kablodan yazılım kontrollü veri transferi sağlanabilmiştir (Ünal, 2006).

CAN başlangıçta sadece otomotiv sistemi için geliştirilmiş olsa da üstün özellikleri ve yüksek performansı sayesinde günümüzde endüstride de uygulamaları yaygın olarak görülmektedir (Dinçer, 2010).

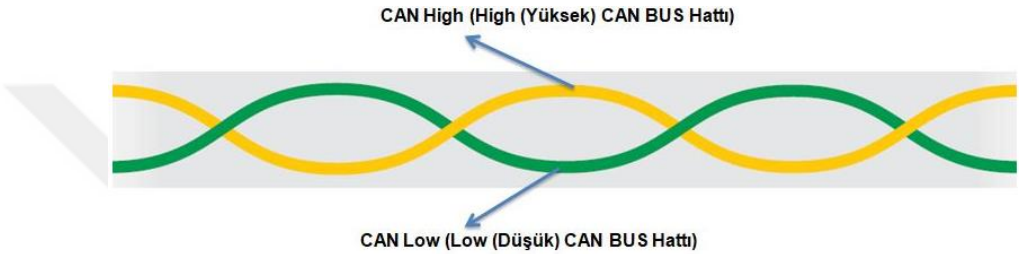
Veri iletim hızı 5 Kbit/s ile 1 Mbit/s arasında değişmektedir. 500 Kbit/s veri hızı neredeyse gerçek zamanlı bir iletim sağlamaktadır (Kayan, 2016). CAN protokolü, iletişim ortamına bit öncelikli yapı ile CSMA/CD kullanarak erişmektedir. Böylece hem mesajların çarpışmasının önüne geçilmekte hem de iletişim hattının uzunluğu sınırlandırmaktadır. Dolayısıyla veri hızı 1 Mbit/s olduğunda hat (bus) uzunluğu en fazla 40 Metre iken 40 Kbit/s olduğunda 1000 Metre olur. Veri aktarım hızına göre yüksek hızlı ve düşük hızlı olmak üzere 2 çeşit CAN iletişim ağından bahsetmek mümkündür (Kayabağı, 2008). Yüksek Hızlı CAN (CAN-C) protokolü; daha çok otomobillerin güç bölümündeki bağlantılar, motor kontrol, vites kutusu ve fren gibi önem arz eden yerlerde kullanılır. Bu birimler aracın en önemli ve kritik donanımları olduğundan 125 kBit/s ile 1 Mbit/s hızları arasında kontrol edilir. Düşük hızlı CAN (CAN-B) katmanı ara katmanlardan biridir. Veri aktarım hızı 5 Kbit/s ile 125 Kbit/s arasındadır. Bu hızlar arabaların kapılarını ve tavanını açmak için yeterlidir. Bu ağ yapısı hatalı mesajların pek fazla önemsenmediği durumlarda kullanılmaktadır. Şekil 3.10'de CAN Bus fiziksel yapısı görülmektedir.



Şekil 3.10. CAN Bus hattının fiziksel yapısı

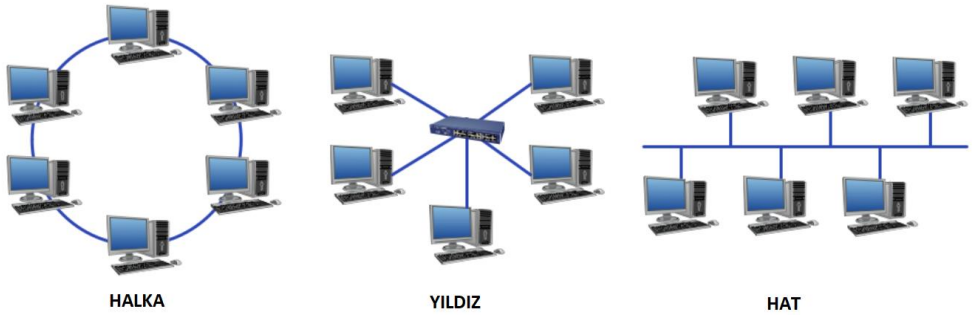
CAN, yapı itibariyle mesaj tabanlı bir protokoldür. Gönderilen mesajın içerisinde içerik ve öncelik bilgisi vardır. Fakat, gideceği modülün adresi yoktur. Yani, mesaj gönderisi modül adreslerine yöneltilmemektedir. Sistemde bulunan modüller, hat üzerinden gelen her mesajı alacaktır. Ardından, modüller gelen mesajın içeriğine bakarak kabul veya reddederler. Gönderilen mesajlar, sistemin tasarım şekline bağlı olarak bazen bir modüle bazen ise birden çok modüle ait olabilir (Dinçer, 2010).

Paralel ve çift sarmal kablo, koaksiyel (eş-eksenli) veya fiber optik kablolar bu sistemlerde iletim için kullanılan en yaygın kablolardır (Dinçer, 2010). Şekil 3.11'de CAN Bus çiftli sarmal kablo yapısı görülmektedir.



Şekil 3.11. CAN Bus çiftli sarmal kablo yapısı

CAN protokolünde Şekil 3.12'de görülebilen hat, halka veya yıldız topolojileri kullanılabilir. Fakat en yaygın kullanılan topoloji tipi, hat topolojisidir (Dinçer, 2010).

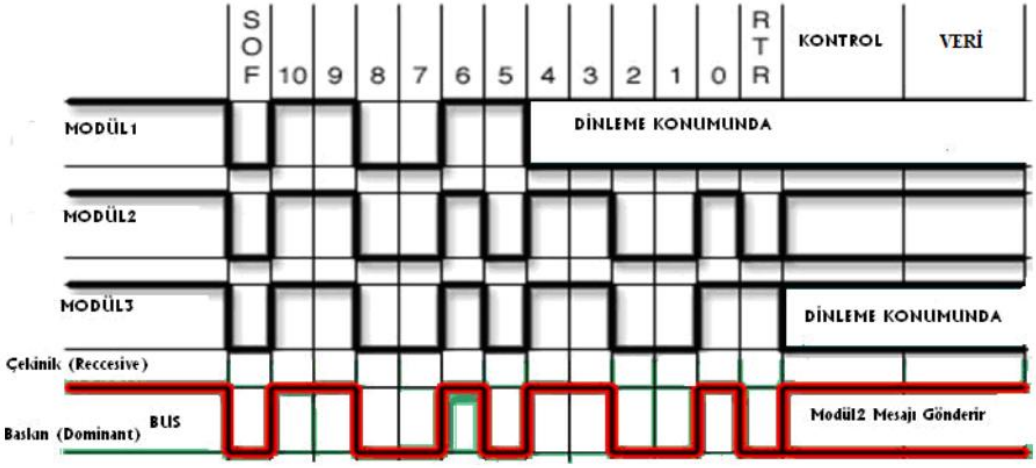


Şekil 3.12. CAN Bus protokolünde kullanılan topolojiler

3.7.1. Çoklu taşıyıcı giriş duyusu (Carrier Sense Multiple Access - CSMA)

CAN protokolü, CSMA/CD+CR (Carrier Sense Multiple Access with Collision Detection And Collision Resolution) yapısında bir protokoldür. CSMA, ağda bulunan tüm modüllerin, hat (bus) üzerinden mesaj gönderimi öncesinde bir periyotluk boş işlem zamanını takip etmesini gerektirir (Fil, 2019). Boş işlem zamanı sonrasında, bus üzerinde bulunan tüm modüller, eşit önceliğe sahip olurlar (Kalaycı, 2015). Çarpışma

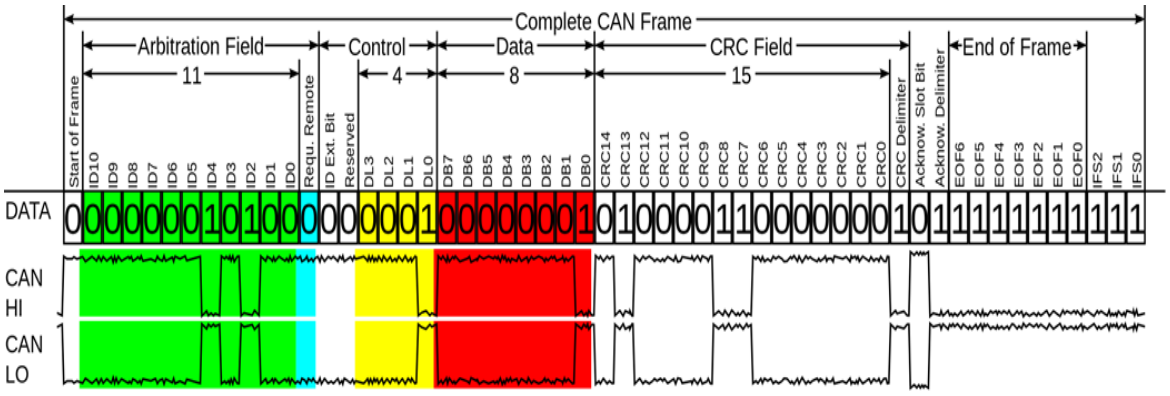
algılama (Collision Detection CD), ağda aynı anda iki modülün mesaj göndermesi durumunda oluşacak çakışmanın ortaya çıkarılması işlemidir (Kayabağı, 2008). Çarpışma Çözünürlüğü (Collision Resolution CR) ise, tespit edilen çakışmanın giderilmesi işlemidir. Çakışmanın çözümlenmesinde uzlaştırma (arbitration) denilen yöntem kullanılmaktadır (Kayan, 2016). Şekil 3.13'te görüldüğü gibi modül1, modül2 ve modül3'ten aynı anda veri gelmektedir. Baskın olan veri öncelikli olarak devam ederken çekinik olanlar CD sayesinde bekleme konumuna alınmaktadır. Öncelikli veri gönderim işlemini tamamlanması ardından CR sayesinde sıradaki veri gönderim işlemine devam etmektedir. Şekil 3.13'de uzlaştırma işlemi görülebilmektedir.



Şekil 3.13. CAN Bus protokolünde uzlaştırma (arbitration) işlemi

3.7.2. CAN Bus veri paketleme yapısı ve mantığı

CAN Bus protokolünde kullanılan veri paketi Şekil 3.14'de görülmektedir. Veri paketinde yer alan bölümler ise alt başlık olarak bu bölümde sunulmuştur.



Şekil 3.14. CAN Bus veri paketi

3.7.2.1. Mesaj ID alanı (Arbitration Field)

CAN Bus protokolünde bilgi veri paketleri ile iletilir. Haberleşme, SOF (Start Of Frame) sinyali ile başlar. SOF sinyali, 1 bitlik bir sinyaldir ve baskındır (dominant). Lojik0 baskın, lojik1 çekingen (recessive) olarak adlandırılmaktadır. Mesaj ID alanında iki tip paketleme yapılır. Birincisi, 11 bit tanımlayıcıya sahip olanlardır ve CAN 2.0A veya standart CAN diye isimlendirilirken ikincisi ise 29 bit tanımlayıcıya sahip olanlardır ve CAN 2.0B veya geliştirilmiş (extended) CAN diye isimlendirilir. Aralarındaki temel fark, tanımlanabilecek mesaj sayısıdır. Standart CAN sisteminde 2^{11}=2048 adet mesaj tanımlanabiliyorken geliştirilmiş CAN sisteminde 2^{29}=536870912 adet mesaj tanımlanabilmektedir (Ünal, 2006).

Mesaj önceliğini mesaj ID belirler. Ayrıca mesaj ID alanı RTR biti de içerir. RTR biti mesajın istek mesajı mı yoksa veri mesajı mı olduğunu belirtmektedir. Eğer çekingen ise gönderilecek pakete istek çerçevesi (remote frame), baskın ise veri çerçevesi (data frame) denir. Bu alandaki ilk 7 bit ardışık olarak çekingen olamaz.

3.7.2.2. Kontrol alanı (Control Field)

Kontrol alanı, 6 bitten oluşur. İlk biti standart veri paketlemesi mi yoksa geliştirilmiş veri paketlemesi mi olduğunu gösterir. Sonraki 1 bit rezervedir. Son 4 bit ise veri alanının kaç byte değerinden oluştuğunu belirten bitlerdir.

3.7.2.3. Veri alanı (Data Field)

CAN-Bus ile veri iletilirken en çok 8 byte bilgi gönderilebilir. Bu değer 8 byte'dan daha az da olabilir.

3.7.2.4. Dönüşsel artıklık kontrol alanı (CRC Field)

Dönüşsel artıklık kontrol alanı, 15 bitlik CRC sırası (sequence) ve CRC sınrılayıcı'dan (delimiter) oluşur. Dönüşsel artıklık kontrol alanının görevi pakete ait CRC (veri doğruluğunu) kodunu tutmaktır. Alınan paketin CRC değeri hesaplanır ve alıcı ünite paket ile birlikte gelen CRC değeri karşılaştırılır. Değerler birbirine eşit ise alınan paket geçerlidir. Aksi durumda CRC hatasından dolayı hata oluştuğunu belirten

hata çerçevesi yollar. Bu bilgiyi alan gönderici, veriyi tekrar göndermeye çalışır. Bu bilgi ünitelerden sadece birinden gelse dahi veri tekrar yollanmalıdır.

3.7.2.5. Alındı bilgisi alanı (ACK Field)

ACK alanı 2 bitten oluşur. İlk biti göndericiden çekingen olarak gelir. Veri, alıcı tarafından doğru alınmışsa bit baskın olur. Bu alanın 2.biti ise ACK delimiter olarak adlandırılır ve çekingendir.

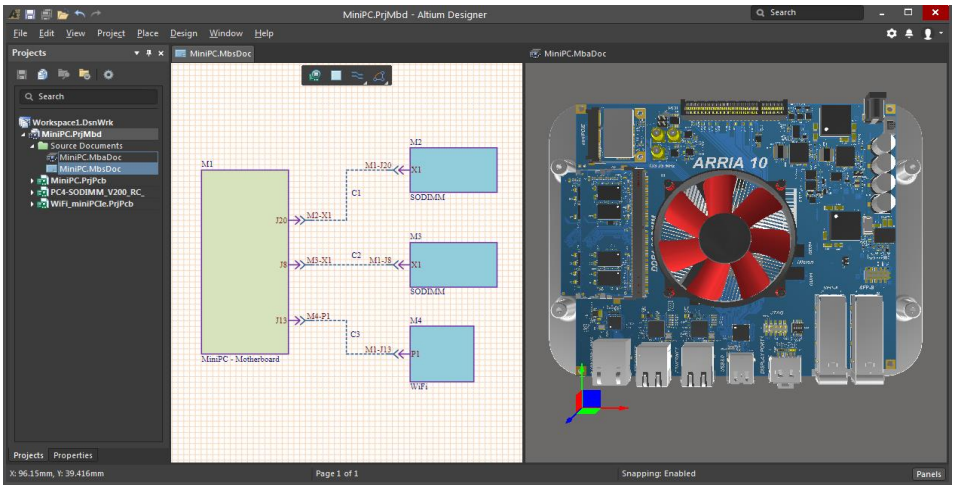
3.7.2.6. Çerçeve sonu alanı (End of Frame)

Bu alandaki bitler çekingendir ve 7 bitten oluşur. Daha sonrasında ise mesajlar arası boşluk bırakmak amacıyla 3 bitlik INT alanı bulunur ve bu bitler çekingendir.

3.8. Altium Designer ile CAN Bus Kontrol Modüllerinin Devre Tasarımı

Altium Designer, baskı devre kartları için bir PCB ve elektronik tasarım otomasyon yazılımıdır. Avustralyalı yazılım şirketi Altium Limited tarafından geliştirilmiştir (Wikipedia, Altium Designer, 2021).

Şekil 3.15'te görüldüğü üzere program üzerinde ilk önce bir proje oluşturarak işleme başlanmaktadır. Proje içerisine şematik ve PCB dosyaları eklenerek proje devam ettirilir. Şematik dosyası içerisinde tasarımı yapılacak projenin malzemeleri ve yolları oluşturulur. Oluşturulan şematik, PCB dosyasına aktarılır ve tasarım gerçekleştirilir. Yapılan tasarım, üretim sonrası elimize alacağımız nihai ürün olacaktır.

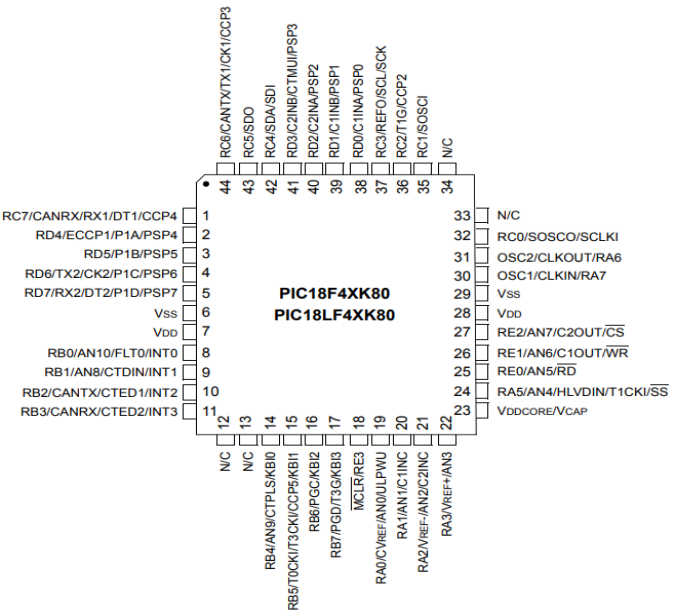


Şekil 3.15. Altium Designer programı şematik ve PCB görseli

Yapılan bu çalışmada Altium Designer üzerinde CAN Bus modüllerinin tasarımı gerçekleştirilmiş olup kullanılan malzemeler ve komponentler özenle seçilmiştir. Komponentlerin seçiminde endüstriyel ve otomotiv kullanımı için uygun olmasına dikkat edilmiştir. PCB çiziminde ise komponentlerin fiziksel yerleşimi, bunların birbiriyle olan bağlantısını sağlayan yolların kalınlığı ve uzunluğu, komponentlerin çalışma sıcaklığı ve gerçekleştireceği fonksiyonlar otomotiv ve endüstriyel kullanımlar için uygun olarak çizilmiştir (Exin Tasarım, 2021). Şematik oluşturma ve PCB tasarımı yapılması özellikle endüstriyel ve otomotiv kullanımı için tecrübe gerektirmektedir.

3.8.1. CAN destekli Microchip PIC MCU (PIC18F46K80)

Gerçekleştirilen çalışmada CAN Bus haberleşme protokolü desteği bulunması, yeterli sayıda giriş/çıkış portuna sahip olması ve yeterli miktarda hafızasının olması sebebi ile işlemci olarak Microchip firmasının PIC18F46K80 mikrodenetleyicisi kullanılmıştır (Derse, 2017). Pin diyagramı Şekil 3.16'da verilen PIC18F46K80 mikrodenetleyici, dışarıdan veri girişini sağlamak, sensörlerden bilgileri gönderip almak, motor kontrolünü sağlamak ve CAN Bus hattı vasıtasıyla ana kontrol kartıyla haberleşip verileri işlemek amacıyla kullanılmıştır.

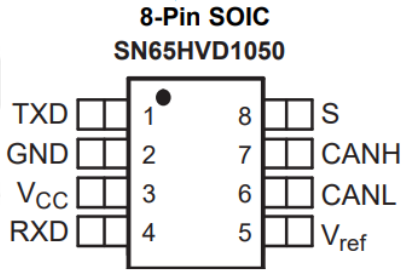


Şekil 3.16. Tasarımda kullanılan PIC18F46K80 MCU pin diyagramı

Bu işlemcide 1 ve 44 nolu veya 10 ve 11 nolu pinlerde bulunan CANRX ve CANTX ön plana çıkmaktadır. Haberleşme için kullanılacak CAN entegresi ile ana işlemci, bu donanım ile haberleşme gerçekleştirecektir. Bu pinler kullanılacak işlemci ve kılıfta değişiklik gösterebilir. Tasarım yapılırken kullanılacak donanım özelliklerine ve donanım gereksinimlerine dikkat edilmesi gerekmektedir.

3.8.2. CAN Bus entegresi (SN65HVD1050)

CAN Bus entegresi, mikroişlemcinin CANRX ve CANTX hattından gelen dijital sinyalleri iletim için uygun bir seviyeye çekerek CANL ve CANH hattından göndermek veya almak için kullanılan CAN hattını oluşturup haberleşmenin sağlandığı entegredir. Pin diyagramı Şekil 3.17'de görülen CAN Bus entegresi, CAN hattı üzerinde dış etmenlerden dolayı oluşan parazitleri engellemek ve bastırma özelliğine de sahiptir (Öztürk, 2010).



Şekil 3.17. Tasarımda kullanılan CAN Bus entegresi pin diyagramı

3.9. C Dili ile PIC MCU CAN Bus Yazılımı

PIC, Microchip firmasının üretimi olan mikro denetleyicilerin genel ismidir. Programlanmamış bir PIC, motoru takılmamış bir arabadan farksızdır. PIC programlamak için Assembly, C, Basic yazılım dilleri programlamada en çok kullanılanlardır. Endüstride PIC programlama en yaygın C dili ile yapılmaktadır. Kod yazımından sonra, yazılımın işlemcinin anlayacağı HEX koduna çevirilip PIC'e yüklenmesi gerekmektedir. HEX uzantlı dosya ICPROG gibi bir yazılımla oluşturulabilir. Yazılım yükleme sonrasında belirlenen giriş ve çıkışlar bağlantı sonrası çalışmaya hazır halde olacaktır (Tesla Akdemi, 2021).

CAN Bus ve sensörlerin yazılımı CCS C derleyicisi üzerinde yapılmıştır. CCS C derleyicisinin CAN Bus kütüphanesinden yararlanılmıştır. Yazılıma başlarken derleyici

üzerinden kullanılacak olan işlemci, kristal ayarları ve gerekli olan özellikler seçilmelidir.

CAN Bus'ın bilgiyi hem alan hem de aktaran bir sistem olduğu unutulmamalıdır. Burada önemli olan hangi bilginin öncelikli olduğuna karar verilmesi ve bir algoritma kurularak bilgilerin gönderiminin sağlanmasıdır (Öztürk, 2010).

3.10. Linux İşletim Sistemi ve Gömülü Yazılım

Bilgisayarda programların çalıştırılmasını ve istenilen temel işlevlerin yerine getirilmesini işletim sistemi sağlamaktadır. 1991 yılında Linus Torvalds tarafından geliştirilen Linux, işletim sistemlerinin çekirdeğidir. Aynı zamanda genel bir adıdır. Günümüzde ise açık kaynak kod felsefesi olan yapıda geliştirilmesine devam edilmektedir. Ubuntu, Debian, Fedora, Pardus, KNOPPIX ve OpenSuse başlıca sayılabilecek Linux tabanlı işletim sistemleridir (Coşar, 2010).

Gömülü sistem ise içinde bulunmuş olduğu sistemin yönetim görevini üstlenerek akıllı hale getiren; donanım ve yazılım alt birimleri ile etkileşimli çalışan entegre sistemin tümüdür.

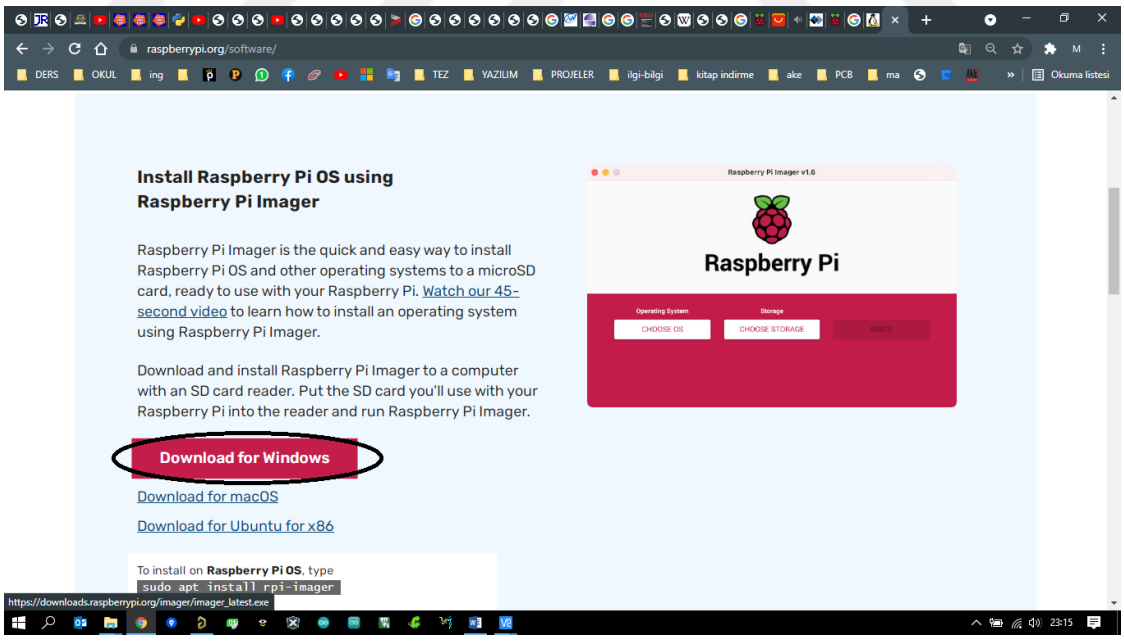
4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA

4.1. Sensör Modüllerinden Alınan Bilgilerin Tasarlanan Ekranda Gösterilmesi

Bilgisayar tarafında ilk yapılacak işlem bir işletim sistemi kurulumu olacaktır. Proje kapsamında kullanılacak bilgisayar, Raspberry Pi ZERO W olduğundan Linux temelli Debian'a dayalı Raspbian işletim sistemi kurulumu yapılmıştır. Kurulum için öncelikle 8GB hafızaya sahip bir hafıza kartına ihtiyaç vardır. Hafıza kartının yüksek hafızalı olarak seçilmesi proje dosyalarında oluşacak büyüklüklerde sıkıntı yaratmaması açısından önemlidir.

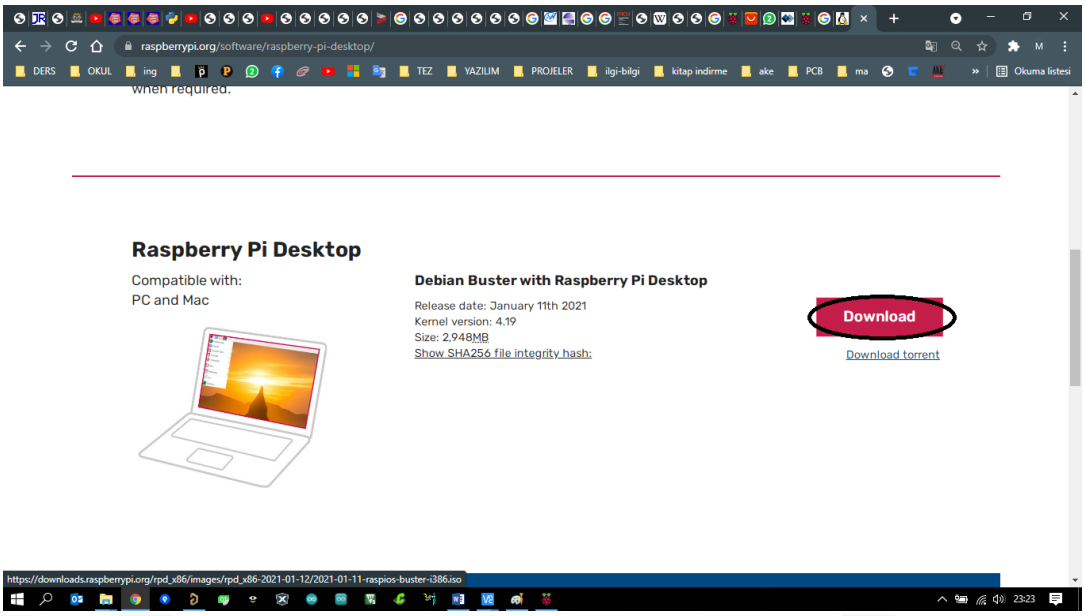
4.1.1. Raspberry Pi'ye işletim sisteminin kurulması

Hafıza kartına işletim sistemini uygun biçimde kurmak için işletim sistemi yazdırma yazılımlara ihtiyaç vardır. Bu tez çalışmasında Raspberry Pi Imager yazılımı tercih edilmiştir. Şekil 4.1'de görülen sayfadan indirilen program kurulum gerektirmeksizin direkt olarak çalışmaktadır. İndirme linki EK-1'de verilmiştir.



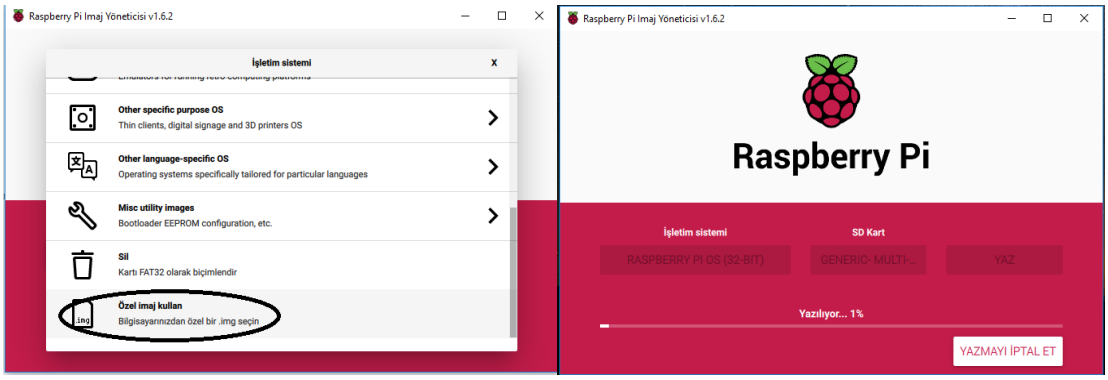
Şekil 4.1. Raspberry Pi Imager indirme sayfası

Gerekli olan Raspbian (Raspberry Pi Debian) işletim sistemi Şekil 4.2'de görülen sayfadan indirilmelidir. İndirme linki EK-1'de verilmiştir.



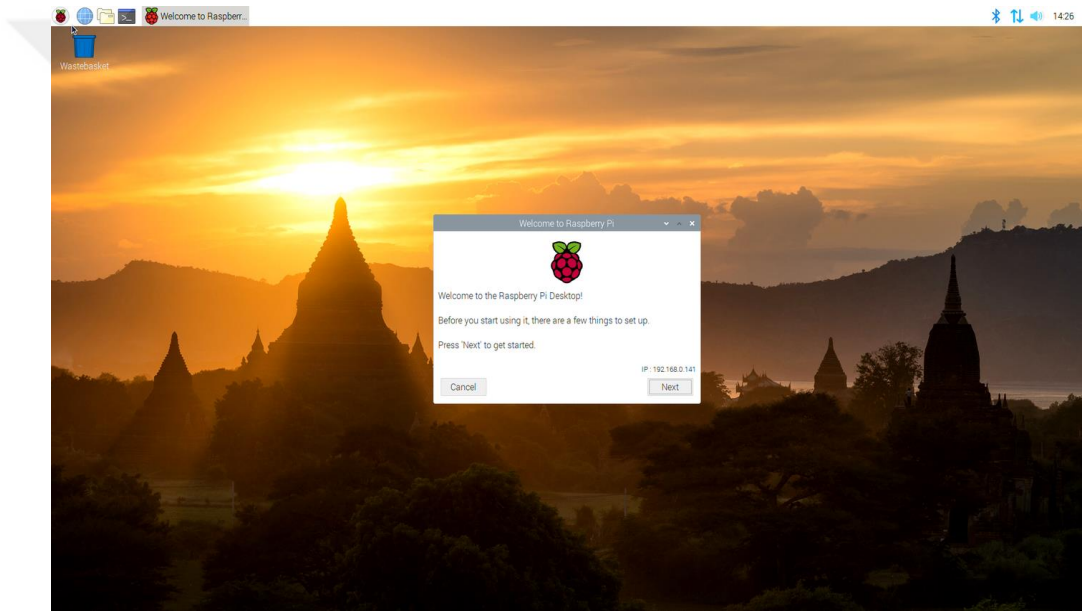
Şekil 4.2. Raspberry Pi Debian işletim sistemi indirme sayfası

İndirilen işletim sisteminin hafıza kartına kurulumunu yapmak için Şekil 4.3'te görüldüğü gibi imager yazılımı çalıştırılır ve acılan pencerede işletim sistemi seçme ikonu tıklatılır. Gelen işletim sistemi penceresinden özel imaj kullan sekmesi tıklanarak önceden indirilmiş olan işletim sisteminin konumu seçilir. Önceden bir işletim sistemi indirmeden işletim sistemi penceresinden de kullanılmak istenilen işletim sistemi kurulumu yapılabilmektedir. Bunun dezavantajı kurulum yaparken biraz daha uzun sürmesidir. Tüm adımlar tamamlandıktan sonra bilgisayara takılı olan SD kart secimi yapılır ve "yaz" butonu imaj yönetici sayfasında tıklanarak kurulumun tamamlanması beklenmelidir.



Şekil 4.3. Raspberry Pi Imager işletim sistemi kurulum sayfası

Raspberry Pi, uygun bir HDMI kablosu yardımıyla monitöre bağlanmalıdır. İşletim sistemi kurulmuş olan hafıza kartı Raspberry pi'nin sd kart slotuna uygun biçimde takılarak Raspberry Pi'ye power portundan güç verilmelidir. Tez çalışmasında kullanılacak olan ekran, standart boyutlarda bir ekran olmadığından dolayı ilk etapta çalışmayabilir. Çalışması için config.txt dosyasında gerekli değişiklikler yapılmalıdır. Değişiklik adımları bölüm 4.1.4'de ayrıntılı olarak anlatılmaktadır. İlk çalıştırmada ve proje yazılımı sürecinde normal bir bilgisayar monitörü kullanılması tasarımda kolaylık acısından daha iyi olacaktır. Sistemin ilk çalışmasında Şekil 4.4'de görülen ekran görüntüsü ile karşılaşılacaktır. USB portuna klavye-fare bağlantısı yaparak gerekli olan kütüphanelerin kurulmasıyla sistem için ilk adım atılmış olmaktadır.



Şekil 4.4. İlk enerjide gelen masaüstü görüntüsü

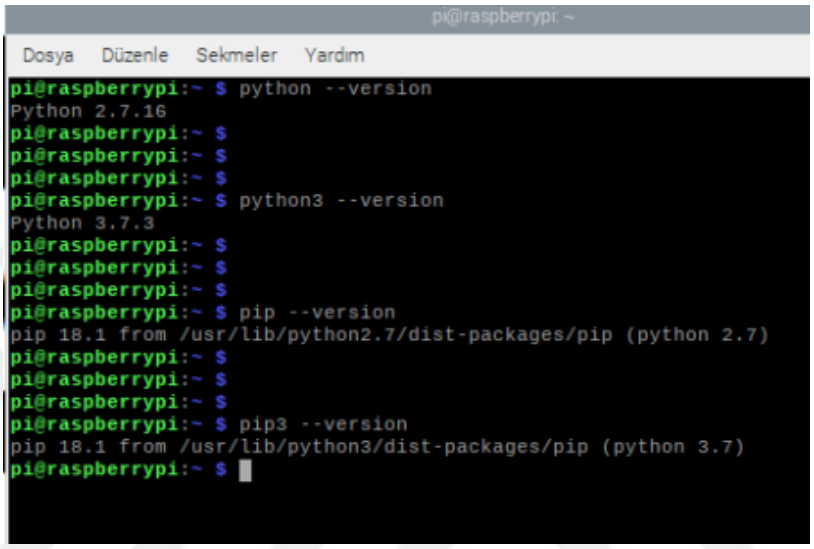
4.1.2. Raspberry Pi'ye gerekli kütüphanelerin kurulması

Raspberry Pi üzerinde Python, QT ve QML araçlarına ve kütüphanelerine ihtiyaç vardır. Linux bir sistem için bu araçların kurulması komut satırı ekranından yapılmaktadır. Bu işlemler sırasında Raspberry Pi internete bağlı olmalıdır.

Çoğu Linux sistemde Python kurulu olarak gelmektedir. Python'ın kurulu olup olmadığını anlamak için işletim sistemi komut satırını açarak "python --version" yazılır. Dönen cevap eğer "Python 2.x.x" şeklinde ise Python 2 veya Python 3, ya da "Python 3.x.x" şeklinde bir sonuç dönmüşse sistemde sadece Python 3 kurulu olduğu anlaşılır.

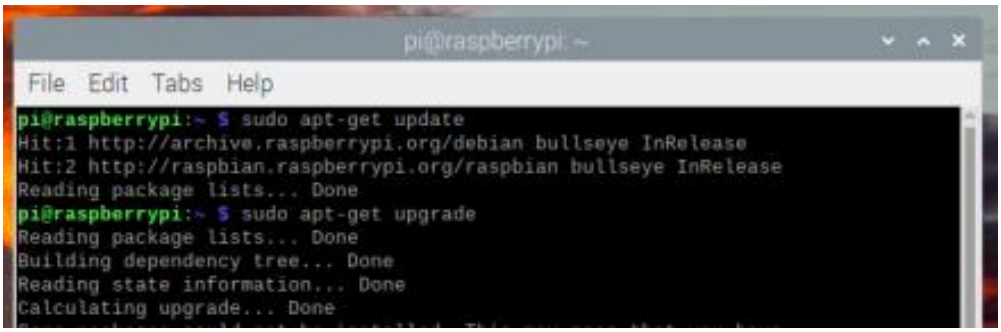
Yalnızca Python 3 versiyonu sorgulamak için komut satırına “python3 –version” yazmak yeterli olacaktır.

Python işletim sisteminin kurulu olmadığına dair bilgiler gelmişse Python kurulumu yapılmalıdır. Python kurulumu yapabilmek için komut satırına “sudo apt-get install python3.8” yazılarak python 3.8 yüklemesi yapılır. Versiyon sorgusu işlemi ardından gelecek olan ekran Şekil 4.5’de görüldüğü gibidir.



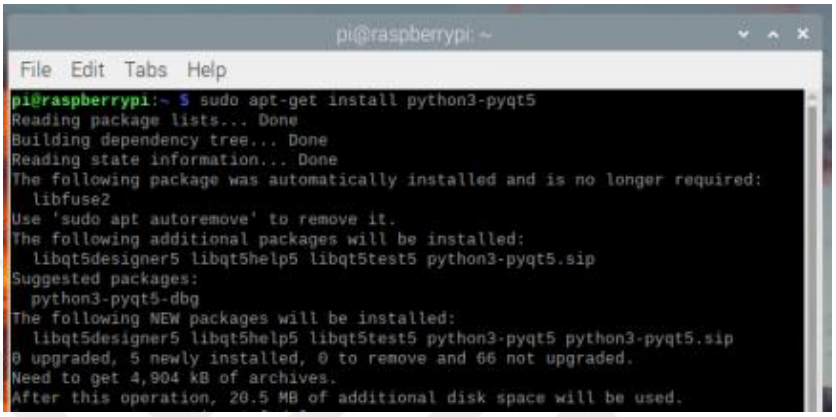
Şekil 4.5. Python ve pip versiyon sorgulama

Yükleme tamamlanınca ileride herhangi bir sorun yaşamamak için sistemin güncellenmesi gerekmektedir. Komut satırı ekranına güncel paket listelerini indirmek için Şekil 4.6’da görüldüğü gibi “sudo apt-get update” yazılarak işlemin bitmesi beklenir. Tamamlanan işlemin ardından kurulu paketlerin daha yeni versiyonlarını yüklemek için “sudo apt-get upgrade” komutunu yazarak sistem güncellemeleri tamamıyla yüklenmelidir (Yapayzekalabs, 2019).



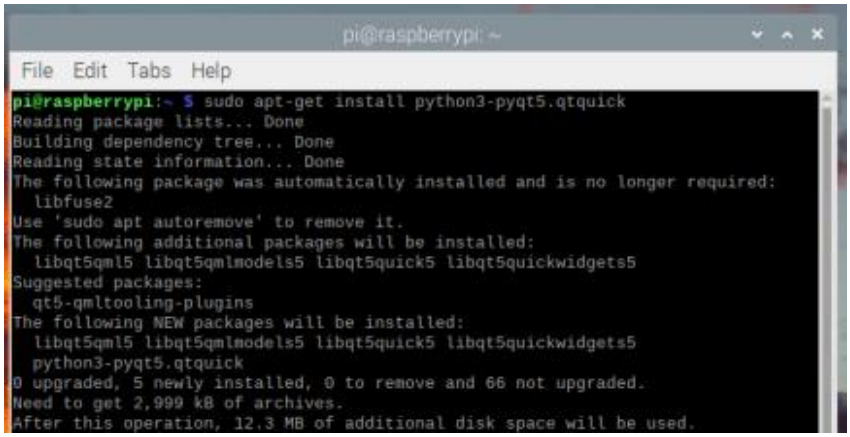
Şekil 4.6. Paketlerin indirilmesi ve versiyonlarının güncellenmesi ekranı

Python yüklemesi ve sistem güncellenmesinden sonraki adım pyqt ve qt araçlarının yüklenmesidir. Python 3 için pyqt5 yüklemek için Şekil 4.7'de görüldüğü üzere komut satırına "sudo apt-get install python3-pyqt5" komutu yazılmalıdır. Tamamlanan işlem ardından pyqt5 araçlarını yüklemek için sırasıyla "sudo apt-get install pyqt5-dev-tools" ve "sudo apt-get install qttools5-dev-tools" komutları çağırılmalıdır.



Şekil 4.7. pyqt5 yüklenmesi ekranı

Son adım olarak Qml modülleri yüklenmelidir. Pyqt5 qml'i python modunda kullanmak için "sudo apt-get install python3-pyqt5.qtquick", pyqt5 qml kontrol modülleri için "sudo apt-get install qml-module-qtquick-controls" ve son olarak pyqt5 qml map modullerini yüklemek için "sudo apt-get install qml-module-qtlocation qml-module-qtpositioning" kodları komut satırına yazılmalıdır. Yükleme ekranı Şekil 4.8'de görülmektedir.



Şekil 4.8. pyqt5'in kullanacağımız kütüphanelerinin yükleme ekranı

Artık tüm kütüphane ve modüller ihtiyaca göre yüklenmiş olup gerekli yazılıma başlamak mümkün hale gelmiştir.

4.1.3. Linux üzerinde GUI yazılımları

İlk adım olarak istenilen ekranın tasarımı yapılmalıdır. Bir aracın ekranında ne olması isteniyorsa tasarımda bunlar dikkate alınmalıdır. Ekran tasarımı için hazırlanan fotoğraflar, masaüstünde bir image klasörü açılarak içerisine kaydedilmelidir. İmage klasörünün bulunduğu konuma "main.qml", "main.py" ve "run.sh" adında üç text dosyası açılmalıdır. Main.qml dosyası içerisinde arayüz Qt Qml kodları, main.py dosyasında ise Python kodları bulunmaktadır. Run.sh dosyasında ise Python kodunu çalıştırmak için üç satırlık bir script bulunmaktadır. Bu dosyaya çift tıklatıp gelen pencerede çalıştır tıklatıldığında oluşturulan kodlar çalışmaya başlayacaktır.

Yazılımda esas çalışan Python kodlarıdır. Python yazılımı içerisinde Qt Qml kodları yani arayüz kodları çağırılmaktadır. İlk olarak main.py dosyası içerisine Python kodlarını yazarak başlanmalıdır.

Dosyanın en tepesine kullanılacak kütüphaneler eklenmelidir. Kullanılacak olan seriport, timerlar, pyqt dosyaları ve sisteme ait kütüphanelerin yazılım içerisine eklenmesi durumunda rahatlıkla erişilebilir hale gelmektedir. Bunun için gerekli kodlar aşağıda verilmiştir.

```
import sys
import threading
import serial
import time
from datetime import datetime
from PyQt5.QtWidgets import QApplication
from PyQt5.QtQml import QQmlApplicationEngine
from PyQt5.QtCore import QTimer
```

Qt application nesnesi bu program için olayların gerçeklemesini ve kontrolünü sağlarken Qmt applicationengine ise qml dosyasını parse edip arayüzü oluşturacak nesneyi oluştururken Javascript engine de bu fonksiyon içerisinde başlatılır. Son olarak arayüzde bulunan root nesne, rootcontext fonksiyonuyla oluşturulurken bu nesne üzerinde değişkenlerin ve görsel bileşenlerin oluşturulması sağlanmaktadır. Bunları gerçekleştirmek için aşağıdaki kod satırı eklenmelidir.

```
app = QApplication(sys.argv)
engine = QQmlApplicationEngine()
ctx = engine.rootContext()
```

Yazılım içerisinde oluşturulacak algoritma haricinde sistemin yürütülmesiyle ilgili olan kodlar devam etmektedir.

```
engine.load('main.qml')
timer=QTimer()
timer.timeout.connect(handleSerialData)
```

main.qml dosyasında oluşturulan ara yüz kodlarının yüklemesi engine.load fonksiyonu ile gerçekleştirilmektedir. Ara yüzün yenilenmesi için kullanılacak olan timer değeri bir değişkene aktarılmaktadır. Timer tetiklendiğinde oluşturulan bir fonksiyonun çağrılması, connect ile başarılı şekilde tamamlanabilir.

Python kodlarının sonuna gelindiğinde seri porttan değer okunacak thread nesnesi yazılacak fonksiyon ile oluşturulmaktadır. Oluşturulan bu thread, start ile başlatılmaktadır. Daha önce oluşturulan timer'ın yenilenme zamanı start ile başlatılır. Bu yazılım için bir event loop çeviri, gelen işletim sistemi ve kullanıcı girdileri ara yüze ve ilgili elemanlara exec ile yollanır. Bu kısımda program durana kadar sonsuz döngü dönmektedir.

```
thread = threading.Thread(target=openSerialPort)
thread.start()
timer.start(100)
app.exec()
```

Oluşturulan bu kod blokları yapılacak uygulama için gerekli olan sistem kodlarıdır. Bu kodların haricinde uygulama için oluşturulan algoritma bulunmaktadır. Kodun tamamı EK-3'de sunulmuştur.

Oluşturulan Python kodları sonrası ara yüz için oluşturulan main.qml dosyası içinde Qt QML kodları yazılır. Her yazılıma başlamadan önce olduğu gibi Qt QML tarafında da kütüphanelerin eklenmesi ile başlangıç yapılmaktadır.

```
import QtQuick 2.11
import QtQuick.Controls 1.4
import QtQuick.Window 2.4
import QtLocation 5.6
import QtPositioning 5.6
```

Kütüphanelerin eklenmesinin ardından Python tarafında oluşturulan ApplicationWindow fonksiyonu artık Qt QML tarafında oluşturulur. Bu fonksiyon içerisinde artık arayüzle ilgili daha önce oluşturulan görseller kullanılarak bir panel

oluşturmaya başlanır. visibility: "FullScreen" ile ekranımızın tam ekran olarak çalışacağını bildirerek height: 480 width: 1920 ile ekran boyutu 1920x480 olarak ayarlanır.

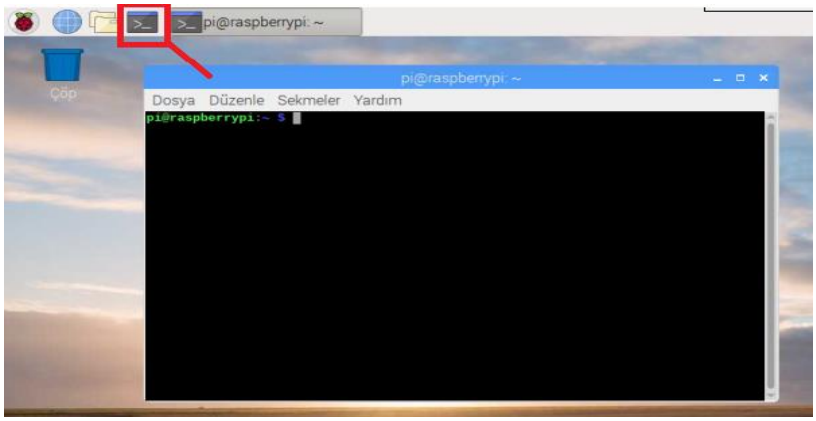
```
Image {
   x:436;y:222
   source: "image/buyukibre.png"
   transform: Rotation
   {
      origin.x: 162; origin.y: 22; angle:(-33+hiz)
   }
}
```

Uygulamada çoğunlukla kullanılacak kod bloğu bu şekildedir. Bir resim kullanılacağını bildirerek resmin x ve y konumu ifade edilir. Kullanılacak resim sistem üzerinde bulunan klasör konumu belirtilerek transform ile bir dönme eylemi gerçekleştirilir (QT, 2021). Ara yüz Qt QML kodlarının tamamı EK-4'te verilmiştir. Bahsedilen tasarım ve yazılımın gerçekleştirilmesi sonrasında görünen ekran Şekil 4.9'da görülmektedir.

Şekil 4.9. Tasarım ve yazılım sonrası ekran görseli

4.1.4. Açılış logosunun değiştirilmesi ve ekran çözünürlüğünün ayarlanması

Yazılım işleminin tamamlanmasıyla birlikte kullanılacak ekran için ekran çözünürlüğü ve açılış logosunu config.txt, pix.script ve cmdline.txt dosyalarında yapılacak değişiklik ile ayarlanabilmektedir. İlk olarak açılış logosu değişimi anlatılacaktır. Ayrıca açılış logosu değişimi sırasında açılırken gelen metinler de kapatılacaktır.



Şekil 4.10. Komut satırının açılışı

İlk olarak Şekil 4.10'da görüldüğü üzere Raspberry Pi bilgisayarında komut satırı ekranı açılır.

Rainbow ekranını kaldırmak için komut satırına "sudo nano /boot/config.txt" yazarak config.txt dosyası root olarak açılır. Satırın en sonuna "disable_splash=1" ekledikten sonra Ctrl+X ile dosyayı kaydedip komut satırına tekrar dönür (Smart Tech, 2020).

Giriş resminin altındaki metin mesajını kaldırmak için tekrar komut satırına "sudo nano /usr/share/plymouth/themes/pix/pix.script" yazarak root olarak "pix.script" sayfası açıldığında aşağıdaki komutlar silinir veya yorum yapılır (Smart Tech, 2020).

```
message_sprite = Sprite();
message_sprite.SetPosition(screen_width * 0.1, screen_height * 0.9, 10000);
my_image = Image.Text(text, 1, 1, 1);
message_sprite.SetImage(my_image);
```

Satırlarında yorum yapmak istenirse satırların başına "#" işareti koymak yeterli olacaktır. İşlem bittikten sonra Ctrl+X ile dosyayı kaydedip komut satırına tekrar dönülür.

Önyükleme mesajlarını kaldırmak için komut satırına "sudo nano /boot/cmdline.txt" yazarak gelen ekranda bulunan "console=tty1"i "console=tty3" ile değiştirilir. Bulunan satırın en sonuna giderek "splash quiet plymouth.ignore-serial-consoles logo.nologo vt.global_cursor_default=0" kodlarını eklenir. Tekrar Ctrl+X ile dosyayı kaydedip komut satırına tekrar dönülür (Smart Tech, 2020).

Son olarak açılış logosu değişimi için öncelikle tasarlanan logo masaüstüne kopyalanmalıdır. Komut satırına "sudo cp New_Logo.png

/usr/share/plymouth/themes/pix/splash.png” komutlarını yazarak “splash.png” adındaki varsayılan açılış logosu “New_Logo.png” olarak kaydedilen yeni logo ile değiştirilir (Smart Tech, 2020). Artık açılışta istenmeyen resimler ve yazılar görülmeyecektir. Şekil 4.11’de görüldüğü gibi açılış yazıları yerine özel tasarlanmış olan logo ile sistem açılmaktadır.

Şekil 4.11. Raspberry Pi yeni açılış logosu görünümü

Bu çalışmada kullanılmış olan ekranın çözünürlüğü standart bir çözünürlük olmadığından HDMI kablosu ile bilgisayara bağlandığında görüntü gelmeyebilir. Görüntü ayarı için “config.txt” dosyasında değişiklik yapılması gerekmektedir. Komut satırı ekranında iken tekrar “sudo nano /boot/config.txt” yazarak ilgili ekrana giriş sağlanır. Gelen sayfanın en altına aşağıdaki kodlar yapıştırılır (Forums raspberrypi, 2020).

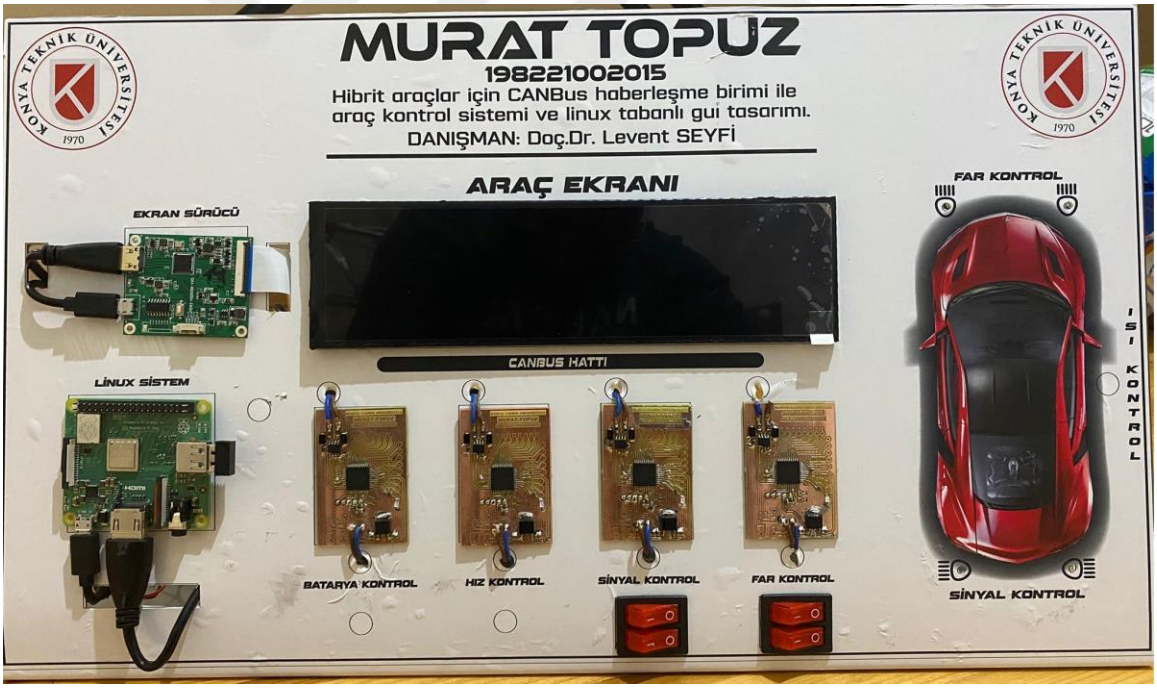
```
hdmi_timings=480 0 30 30 30 1920 0 6 6 6 0 0 0 60 0 55296000 8
hdmi_group=2
hdmi_mode=87
hdmi_drive=2
display_rotate=1
hdmi_force_mode=1
framebuffer_width=1920
framebuffer_height=480
max_framebuffer_width=1920
max_framebuffer_height=1920
```

İşlem bittikten sonra Ctrl+X ile dosya kaydedilerek komut satırına tekrar dönülür. 1920x480 çözünürlükte olan ekran bilgisayara HDMI yardımıyla bağlanıp komut satırına “sudo reboot” yazarak sistem yeniden başlatılır. Sistem artık yeni açılış logosuyla sorunsuz çalışacaktır. Böylece, Linux sistem için python kurulumu, pyqt ve qt araçlarının yüklenmesi, qml modüllerinin indirilip yüklenmesi ve son olarak açılış logo ve yazıların değiştirilmesi işlemleri tamamlanmış olup sistem uygun biçimde çalışmaya hazır hale getirilmiştir.

5. SONUÇLAR VE ÖNERİLER

Gerek otomotiv, gerekse endüstriyel alanda oluşturulmak istenen elektronik kontrol ünitelerinin adım adım tasarlanıp hayata geçirilmesi konusunda büyük katkı sağlayacağı düşüncesiyle bu uygulama çalışması gerçekleştirilmiştir. Ayrıca otomotivde ve endüstride sıkça kullanılan CAN Bus haberleşmesi, gömülü sistemlerde adını sıklıkla duyduğumuz QT, QML programlama dili, hemen hemen yazılımın her alanında kullanılabilen Python programlama dili ve işlemci programlamada adından sıkça söz ettiren C programlama dili de çalışma içerisinde anlatılarak bu konularda çalışacaklara önemli bir kaynak ortaya konmuştur. Elektronik kontrol kartı tasarımı, kontrol kartı yazılımları, gömülü sistemler ve yazılımlar, hatta Linux sistem üzerinde yazılım geliştirilerek bir ekran çalışması başarıyla gerçekleştirilmiştir.

Tasarlanan sensör, kontrol sistemi ve ekran sistemin çalışmasını daha iyi kavramak adına bir deney düzeneği de Şekil 5.1'de görüldüğü gibi sergilenmiştir.



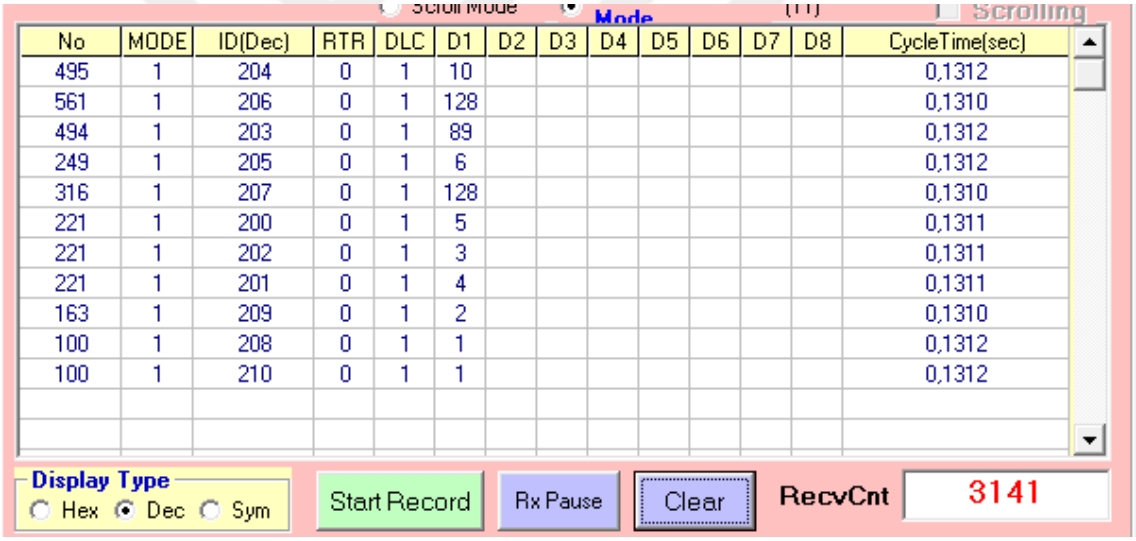
Şekil 5.1. Gerçekleştirilen deney düzeneğinin son hali

Sergilenen düzenekte kullanılan sensörleri simüle eden anahtarlar, potlar ve görsel olarak ledler kullanılmıştır. Belirlenen sensörün anahtarından veya potundan gelen bilgiyi CAN Bus haberleşmesi vasıtasıyla sorunsuz biçimde kontrol birimine göndererek kontrol biriminin veriyi işlemesiyle bir geri bildirim mesajı gönderilmiştir ve yapılması gereken işlemin sorunsuz biçimde tamamlandığı gözlemlenmiştir.

Gönderilen bu geri bildirim mesajı sistemin önceliğine göre hareket ederek belirlenen CAN Bus filtresi sağlıklı olarak çalışmıştır.

Kullanılan düzenekte anahtarlar ya da pot yerine araçlar için satılan sinyal kolu, far anahtarı veya gaz pedalı satın alınarak sistemin daha da geliştirilmesi mümkündür. Ancak maliyetleri oldukça fazla olacaktır. Deney düzeneği yerine gerçek bir araç tasarımı yapılarak araç üzerinde de bu sistem çalıştırılabilir. Bu sayede deney ile gerçek çalışma arasında farkların olacağı da gözlemlenmiş olacaktır. Ayarlanan değerler test düzeneğine göre olduğundan gerçek bir araç çalışmasında yazılımda bulunan aralıklar ve değerler kalibre edilmelidir.

Sistem bir bütün olarak çalışırken deney düzeneği üzerinde Şekil 5.2'de görüldüğü gib CAN Bus hattı dinlenip mesajların incelemesi yapılmıştır. Mesajların alınması, bilgisayar yazılımı ve CAN Bus okuyucu modül yardımıyla gerçekleştirilmiştir.



Şekil 5.2. CAN Bus hattından dinlenen mesajlar

Sonuç olarak geliştirilen bu sistem bilgi verici ve eğitici bir eğitim materyali olarak laboratuvar uygulamalarında da kullanılabileceği gibi hibrit veya elektrikli araçlarda kullanılabilecek bir kontrol sisteminin altyapısını oluşturmaktadır.

Gelecek çalışmalarda ise Linux sistemi GSM-GPRS vasıtasıyla internete ve mobil şebekeye bağlayarak araç konumu ve internet üzerinden anlık bilgiler alınarak çevrimiçi bilgiler ekranda kullanıcıya gösterilebilir. Böylece tasarlanan ekrandaki harita aracılığıyla sistem, navigasyon amaçlı da kullanılabilecektir.

KAYNAKLAR

Acar C., 2019 https://www.autonom.com.tr/can-bus-ve-arac-ici-ag-iletisimi/ [Son erişim 26.10.2021]

Adafruit, 2018, Learn Raspberry Pi, https://learn.adafruit.com/category/learn-raspberry-pi [Son erişim 26.10.2021]

Arslan, S., Gündüzalp, M., Türk, E., 2016, Gömülü Bir Sistem İçin Bir Çoklu Ortam Uygulaması, Electrical, Electronics and Biomedical Engineering Conference (ELECO 2016), 189 – 193.

Bulgu, A.E., 2010, Tekerlek Motorlu Seri Hibrit Elektrikli Araçlar İçin Kontrol Algoritmalarının Geliştirilmesi, Yüksek Lisans Tezi, *İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, Kontrol ve Otomasyon Mühendisliği Anabilim Dalı.*

Çağışlar, A.S., 2018, Elektrikli Araçlar İçin Fırçasız Doğru Akım Motoru Tasarımı, Yüksek Lisans Tezi, *İstanbul Üniversitesi Fen Bilimleri Enstitüsü Elektrik Elektronik Mühendisliği Anabilim Dalı.*

Çoşar, M., 2010, İşletim Sistemi, Hitit Üniversitesi Ders Notu.

Derse, C., 2017, Yeni Bir Can Bus İletişim Tabanlı Otomotiv Güç Dağıtım Modülü Uygulaması Ve Sistemin Güvenilirlik Analizi : "eaPDM" , Yüksek Lisans Tezi, *Celal Bayar Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı*

Dinçer E., 2010, Can Bus İle Dağıtık Kontrol Uygulaması, Yüksek Lisans Tezi, *Niğde Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik Elektronik Mühendisliği Anabilim Dalı.*

Durgun, Y.E., 2019, Elektromobil Araç Kontrol Sistemi Tasarımı Ve Entegrasyonu, Yüksek Lisans Tezi, *Sakarya Uygulamalı Bilimler Üniversitesi, Lisansüstü Eğitim Enstitüsü, Mekatronik Mühendisliği Anabilim Dalı.*

Erçin, M.E., 2020, Elektrikli Ve Hibrit Araçlar İçin Motor Sürücü İnvertörü Geliştirilmesi, Yüksek Lisans Tezi, *Gebze Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik Mühendisliği Anabilim Dalı.*

Exin tasarım, 2021, PCB Kart Tasarımı, http://www.exintasarim.com/pcb-kart-tasarimi/ [Son erişim 26.10.2021]

Fil, C., 2019, İçten yanmalı motorların j1939 CAN Bus'tan alınan bilgilerinin ve hata mesajlarının işlenmesi ve mobil uygulamada görüntülenmesi, Yüksek Lisans Tezi, *İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, Mekatronik Mühendisliği Anabilim Dalı.*

Forums raspberrypi, 2020, 1440x1440 HDMI timings, https://forums.raspberrypi.com/viewtopic.php?t=263639 [Son erişim 26.10.2021]

Harrison, I., 2006, Büyük Buluşlar, Doğuş Grubu İletişim ve Yayıncılık.

Kalaycı, O., 2015, PLC ve CAN Bus Haberleşme Protokolü İle Bina Enerji Yönetimi Uygulaması, Yüksek Lisans Tezi, *Sakarya Üniversitesi Fen Bilimleri Enstitüsü, Mekatronik Mühendisliği Anabilim Dalı.*

Kayabağı, H., 2008, CAN (Controller Area Network) Temelli Alarm Sistemi Tasarımı, Yüksek Lisans Tezi, *İnönü Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı*.

Kayan Ö., 2016, CAN-Bus Protokolü, http://omerkayan.blogspot.com/2016/07/can-bus-protokolu.html [Son erişim 26.10.2021]

Kerem, A., 2014, Elektrikli Araç Teknolojisinin Gelişimi ve Gelecek Beklentileri, *Mehmet Akif Ersoy Üniversitesi, Fen Bilimleri Enstitüsü Dergisi*, 5(1), 1 – 13.

Özbey, B., 2020, Elektrikli araçlar için kablosuz şarj sistemi tasarımı ve optimizasyonu, Yüksek Lisans Tezi, *İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Kontrol ve Otomasyon Mühendisliği Anabilim Dalı*.

Özcan, F.Ö., 2019, Labview ve Raspberry Pi 3 Kullanarak Sıcaklık Kontrolü, Yüksek Lisans Tezi, *İnönü Üniversitesi, Fen bilimleri Enstitüsü, Elektrik Elektronik Mühendisliği Anabilim Dalı*.

Öztürk, M., 2010, Hidrojen Hibrit Otobüslerde CAN Bus Sistemleri ve Uygulaması, Yüksek Lisans Tezi, *Gebze Yüksek Teknoloji Enstitüsü, Mühendislik ve Fen Bilimleri Enstitüsü, Elektronik Mühendisliği Anabilim Dalı*.

QT, 2021, Qt QML, https://doc.qt.io/qt-5/qtqml-index.html [Son erişim 26.10.2021]

Sefik, İ., 2017, Elektrikli ve Hibrit Araçlar: Kontrol Sistemleri, https://tr.linkedin.com/pulse/elektrikli-ve-hibrit-ara%C3%A7lar-kontrol-sistemleri-i%CC%87brahim-%C5%9Fefik [Son erişim 26.10.2021]

Smart Tech, 2020, Raspberry Pi Boot Process, https://www.youtube.com/watch?v=kdugp7DrODY [Son erişim 26.10.2021]

Tesla Akademi, 2021, PIC Programlama, https://teslaakademi.com/pic-programlama [Son erişim 26.10.2021]

Tuncay, R., Üstün, Ö., 2004, Otomotiv Elektroniğindeki Gelişmeler, *Elektrik Elektronik ve Bilgisayar Mühendisliği Sempozyumu*, Bursa.

Urkun, Ş.F., 2020, Elektrikli araçlar için kontrol sistemi geliştirilmesi, Yüksek Lisans Tezi, Ege Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik Elektronik Mühendisliği Anabilim Dalı.

Ünal, İ., 2006, CAN (Control Area Network) Üzerinden PIC Programlama, Yüksek Lisans Tezi, *Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, Isparta Elektronik Bilgisayar Eğitimi Anabilim Dalı*.

Ünlü, N., Karahan, Ş., Tür, O., Uçarol, H., Özsu, E., Yazar, A., Turhan, L., Akgün, F., Tırıs, M., 2003, Elektrikli Araçlar, TÜBİTAK Marmara Araştırma Merkezi, Enerji Sistemleri ve Çevre Araştırma Enstitüsü, Gebze.

Wikipedia, 2021, Altium Designer,https://en.wikipedia.org/wiki/Altium_Designer [Son erişim 26.10.2021]

Yapayzekalabs, 2019, Raspberry Pi Qt Kurulumu, https://yapayzekalabs.blogspot.com/2019/12/qt-raspberry-pi-kurulumu.html [Son erişim 26.10.2021]

Yıldırım, C., 2020, Gömülü sistemler için bilgisayar tabanlı grafiksel kullanıcı arayüzü (GUI) tasarım aracı geliştirilmesi, Yüksek Lisans Tezi, Marmara Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Ana Bilim Dalı.

Yörükoğulları, E., Topdemir, H.G., İhsanoğlu, E., 2013, Bilim ve Teknoloji Tarihi, Anadolu Üniversitesi, Eskişehir

EKLER

EK-1
Ürün satın almak için internet sitesi linkleri

- Ekran satın alma linki.

https://tr.aliexpress.com/item/1005001453148082.html?spm=a2g0s.9042311.0.0.27424c4dLzqO1x

- Linux PC Raspberry Pi satın alma linki.

https://market.samm.com/raspberry-pi-zero-w

- Komponentlerin tedarik linki.

https://ozdisan.com/

- Raspberry Pi Imager indirme linki.(Hafıza kartına işletim sistemi kurmak için)

https://www.raspberrypi.org/software/

- İşletim sistemi indirme linki.

https://www.raspberrypi.org/software/raspberry-pi-desktop/

EK-2
CCS C kodları

```
#include <18F46K80.h>
#include <stdio.h>

//#FUSES PLLEN
#FUSES WDT
#FUSES WDT4096
#FUSES NOXINST
//#FUSES NOBROWNOUT
#FUSES BORV30
#FUSES NODEBUG
#FUSES BORM_LOW
#FUSES PROTECT

#use delay(internal=8MHz)
#use
rs232(baud=9600,parity=N,xmit=PIN_D6,rcv=PIN_D7,bits=8,stream=PORT2)

#define CAN_USE_EXTENDED_ID FALSE
#define CAN_BRG_SEG_2_PHASE_TS TRUE
#define CAN_BRG_PRESCALAR 4
#define CAN_BRG_SYNCH_JUMP_WIDTH 0
#define CAN_BRG_PROPAGATION_TIME 2
#define CAN_BRG_PHASE_SEGMENT_1 5
#define CAN_BRG_PHASE_SEGMENT_2 5
#define CAN_BRG_WAKE_FILTER TRUE
#define CAN_BRG_SAM FALSE

#include <can-18F4580.c>

long long Gelen_ID_ = 0;
char data_Gonderilen[8];
int data_CevapGelen[8];
int len = 12;
int32 stat = 0;
unsigned int32 Timer1Ms = 0, TimerKullanilan = 0, ToogleTimer = 0;
int led_i = 0;

int hiz = 0, devir = 0, devir2 = 0, hararet = 0, yakit = 0, batarya = 0, solKucuk =
0, ortaFps = 0, farlar = 0, sinyaller = 0, arizalar = 0;

enum {
  data_hiz = 0, data_devir, data_devir2, data_hararet, data_yakit, data_batarya,
data_solKucuk, data_ortaFps, data_farlar, data_sinyaller, data_arizalar
};
```

```
//#define FarModulu
//#define SinyalModulu
//#define HizModulu
#define BataryaModulu

#INT_CANRX0////can rx den bilgi geldigi zaman bu kesmeye girer

void CANRX0_isr(void) {

  Gelen_ID_ = 0;
  can_getd(Gelen_ID_, data_CevapGelen, len, stat);
  //if (!(COMSTAT.txbo)) can_putd(Gelen_ID_, data_CevapGelen, len, 3, 1, 0);
  // retrieves a message from the CAN bus storing the
  // ID in the ID variable, the data at the array pointed to by
  // Ã¢â‚¬Å“data', the number of data bytes in len, and statistics
}


#INT_CANERR ////overflow olduysa

void CANERR_isr(void) {
  if (COMSTAT.rx0ovfl) {
    can_getd(Gelen_ID_, data_CevapGelen, len, stat);
    COMSTAT.rx0ovfl = 0;
  }

}

#INT_TIMER2

void TIMER2_isr(void) {
  Timer1Ms++;
  static int ledtoogle = 0;
  if (ledtoogle++ >= 30) {
    ledtoogle = 0;
    led_i = !led_i;
    if (led_i) output_high(PIN_A5);
    else output_low(PIN_A5);
  }
}

long map(long x, long in_min, long in_max, long out_min, long out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void main() {
  restart_wdt();
  //----------------- CAN AYARLARI ---------------------------------
  can_init();
```

```
          can_set_mode(CAN_OP_CONFIG);
          can_set_mode(CAN_OP_NORMAL);
          can_disable_filter(0x1fffffff);
          //-----------------END-----------------------------------------------
          setup_adc(adc_clock_internal);
          setup_adc_ports(3);

          if (COMSTAT.rx0ovfl)COMSTAT.rx0ovfl = 0; //RX DE TASÂ?MA VARSA
REGISTERI SIFIRLIYOR

          setup_timer_2(T2_DIV_BY_16, 31, 10);

          enable_interrupts(INT_TIMER2);
          enable_interrupts(INT_CANRX0);
          enable_interrupts(INT_CANERR);
          enable_interrupts(GLOBAL);

          while (TRUE) {
           restart_wdt();

           if (Timer1Ms - TimerKullanilan > 50) {
            TimerKullanilan = Timer1Ms;

        #ifdef FarModulu
            ////////////////////farlar modulu//////////////////////

            data_Gonderilen[0] = input(PIN_C6); // farlar acikmi?
         if (!(COMSTAT.txbo))can_putd(data_farlar + 200, data_Gonderilen, 1, 3, 1, 0);
            delay_us(20);

            data_Gonderilen[0] = input(PIN_C5); // ariza varmi?
   if (!(COMSTAT.txbo))can_putd(data_arizalar + 200, data_Gonderilen, 1, 3, 1, 0);
            delay_us(20);

            ////////////////////////////////////////////////////
        #endif

        #ifdef SinyalModulu
            ////////////////////Sinyal modulu//////////////////////
            data_Gonderilen[0] = (1 * input(PIN_C5)) + (2 * input(PIN_C6));
            // data_Gonderilen[data_Sinyaller]= (1*input(PIN_C5))+(2*input(PIN_C6));
  if (!(COMSTAT.txbo)) can_putd(data_Sinyaller + 200, data_Gonderilen, 1, 3, 1, 0);

            ////////////////////////////////////////////////////
        #endif

        #ifdef HizModulu
            ////////////////////Hiz modulu//////////////////////
            set_adc_channel(3);
            delay_us(20);
```

```
    data_Gonderilen[0] = map(read_adc(), 0, 255, 0, 248);
    static unsigned int val_hiz = 0;
    if (val_hiz < data_Gonderilen[0])val_hiz += 5;
    else if (val_hiz > data_Gonderilen[0] && val_hiz >= 6)val_hiz -= 6;
    if (val_hiz > 248)val_hiz = 248;
    else if (val_hiz < 3)val_hiz = 2;
    data_Gonderilen[0] = val_hiz;
  if (!(COMSTAT.txbo))can_putd(data_hiz + 200, data_Gonderilen, 1, 3, 1, 0);
    delay_us(20);

    static unsigned int val_devir = 0;
    data_Gonderilen[0] = map(read_adc(), 0, 255, 0, 200);
    if (val_devir < data_Gonderilen[0])val_devir += 4;
    else if (val_devir > data_Gonderilen[0] && val_devir >= 8)val_devir -= 8;
    if (val_devir > 200)val_devir = 200;
    else if (val_devir < 4)val_devir = 4;
    data_Gonderilen[0] = val_devir;
 if (!(COMSTAT.txbo))can_putd(data_devir + 200, data_Gonderilen, 1, 3, 1, 0);
    delay_us(20);

    static unsigned int val_devir2 = 0;
    data_Gonderilen[0] = map(read_adc(), 0, 255, 0, 86);
    if (val_devir2 < data_Gonderilen[0])val_devir2 += 2;
   else if (val_devir2 > data_Gonderilen[0] && val_devir2 >= 7)val_devir2 -= 7;
    if (val_devir2 > 86)val_devir2 = 86;
    else if (val_devir2 < 3)val_devir2 = 3;
    data_Gonderilen[0] = val_devir2;
if (!(COMSTAT.txbo))can_putd(data_devir2 + 200, data_Gonderilen, 1, 3, 1, 0);
    delay_us(20);
    ////////////////////////////////////////////////////

#endif

#ifdef BataryaModulu
    ////////////////////Batarya modulu////////////////////
    set_adc_channel(3);
    delay_us(20);

    data_Gonderilen[0] = map(read_adc(), 0, 255, 0, 92);
    static unsigned int val_bat = 0;
    if (val_bat < data_Gonderilen[0])val_bat += 3;
    else if (val_bat > data_Gonderilen[0] && val_bat >= 6)val_bat -= 6;
    if (val_bat > 92)val_bat = 92;
    else if (val_bat < 6)val_bat = 6;

    data_Gonderilen[0] = map(read_adc(), 0, 255, 0, 154);
    static unsigned int val_yakit = 0;
    if (val_yakit < data_Gonderilen[0])val_yakit += 5;
    else if (val_yakit > data_Gonderilen[0] && val_yakit >= 6)val_yakit -= 6;
    if (val_yakit > 154)val_yakit = 154;
```

```
            else if (val_yakit < 10)val_yakit = 10;



            data_Gonderilen[0] = map(read_adc(), 0, 255, 0, 89);
            static unsigned int val_hararet = 0;
            if (val_hararet < data_Gonderilen[0])val_hararet += 5;
          else if (val_hararet > data_Gonderilen[0] && val_hararet >= 6)val_hararet -= 6;
            if (val_hararet > 89)val_hararet = 89;
            else if (val_hararet < 5)val_hararet = 5;

            static  int16 solval = 0;
            if (solval < 263)solval += 2;
            else solval = 0;

            static  int16 fpsval = 0;
            if (fpsval < 265)fpsval += 2;
            else fpsval = 0;

            ////////////////////////////////////////////////////

            hararet = val_hararet;
            yakit = val_yakit;
            batarya = val_bat;
            solKucuk = solval;
            ortaFps = fpsval;
          #endif

           }

           if (Timer1Ms - ToogleTimer > 200) {
            ToogleTimer = Timer1Ms;

          #ifdef FarModulu
            if (Gelen_ID_ == 300 + data_farlar) {//FARLAR icin ANA
KONTROLCUDEN GELEN CEVAP
             if (data_CevapGelen[0] == 1) {
              output_high(PIN_C1);
              output_high(PIN_C0);
             }
             else {
              output_low(PIN_C1);
              output_low(PIN_C0);
             }
            }
          #endif

          #ifdef SinyalModulu
            static int toogle = 1;
```

```
        if (Gelen_ID_ == 300 + data_sinyaller) {//sinyaller icin ANA KONTROLCUDEN GELEN CEVAP
          if (data_CevapGelen[0] == 1 && data_CevapGelen[1] == 1) {
            if (toogle) {
              output_high(PIN_C1);
              output_high(PIN_C0);
              toogle = 0;
            } else {
              output_low(PIN_C1);
              output_low(PIN_C0);
              toogle = 1;
            }
          } else if (data_CevapGelen[0] == 1 && data_CevapGelen[1] == 0) {
            if (toogle) {
              output_high(PIN_C1);
              toogle = 0;
            } else {
              output_low(PIN_C1);
              toogle = 1;
            }
            output_low(PIN_C0);
          } else if (data_CevapGelen[0] == 0 && data_CevapGelen[1] == 1) {
            if (toogle) {
              output_high(PIN_C0);
              toogle = 0;
            } else {
              output_low(PIN_C0);
              toogle = 1;
            }
            output_low(PIN_C1);
          } else {
            output_low(PIN_C1);
            output_low(PIN_C0);
          }
        }

      #endif

      #ifdef HizModulu

      #endif

      #ifdef BataryaModulu

        if (Gelen_ID_ == 200 + data_farlar) {//FARLAR icin modulden gelen
          farlar = data_CevapGelen[0];
      if (!(COMSTAT.txbo))can_putd(300 + data_farlar, data_CevapGelen, 1, 3, 1, 0);
          delay_us(20);
        }
        if (Gelen_ID_ == data_arizalar + 200) {
```

```
            arizalar = data_CevapGelen[0];
          }
          if (Gelen_ID_ == 200 + data_sinyaller) {
            sinyaller = data_CevapGelen[0];
            switch (data_CevapGelen[0]) {
              case 1:
                data_Gonderilen[0] = 1;
                data_Gonderilen[1] = 0;
                break;
              case 2:
                data_Gonderilen[0] = 0;
                data_Gonderilen[1] = 1;
                break;
              case 3:
                data_Gonderilen[0] = 1;
                data_Gonderilen[1] = 1;
                break;
              default:
                data_Gonderilen[0] = 0;
                data_Gonderilen[1] = 0;
                break;
            }
      if (!(COMSTAT.txbo))can_putd(300 + data_sinyaller, data_Gonderilen, 2, 3, 1, 0);
            delay_us(20);

          }
          if (Gelen_ID_ == data_devir2 + 200) {
            devir2 = data_CevapGelen[0];
          }
          if (Gelen_ID_ == data_devir + 200) {
            devir = data_CevapGelen[0];
          }
          if (Gelen_ID_ == data_hiz + 200) {
            hiz = data_CevapGelen[0];
          }

          printf("%d %d %d %d %d %d %d %d %d %d %d\n", hiz, devir, devir2,
hararet, yakit, batarya, solKucuk, ortaFps, farlar, sinyaller, arizalar);
        #endif

        }
      }
}
```

EK-3
Python kodları

```
# kullanılan kütüphanelerin eklenmesi
import sys
import threading
import serial
import time
from datetime import datetime
from PyQt5.QtWidgets import QApplication
from PyQt5.QtQml import QQmlApplicationEngine
from PyQt5.QtCore import QTimer

app = QApplication(sys.argv) # qt application nesnesi bu program ici olaylarin gerceklemesini ve kontrolunu saglar
engine = QQmlApplicationEngine() # qml dosyasını parse edip arayuzu olusturacak nesnenin olusturulmasi. Javascript engine de bunu icinde baslatilir
ctx = engine.rootContext() # arayuz de bulunan root nesne. Bu nesne uzerinde degiskenler olusturalabilir veya gorsel bilesenler olusturulabilir

data = "0 0 0 0 0 0 0 0 0 0 0" # ekranda gosterilen verinin tanimlanmasi

now = datetime.now() # suanki sistem saatini alinmasi
tarih = now.strftime("%H:%M\n%d/%b/%y") # tarih ve saatin formatlanmasi

# arayuz degiskenlerinin varsayilan degerleri ile olusturulmasi
ctx.setContextProperty("hiz", 0)
ctx.setContextProperty("devir", 0)
ctx.setContextProperty("devir2", 0)
ctx.setContextProperty("hararet", 0)
ctx.setContextProperty("yakit", 0)
ctx.setContextProperty("batarya", 0)
ctx.setContextProperty("solKucuk", 0)
ctx.setContextProperty("ortaFps", 0)
ctx.setContextProperty("farlar", 0)
ctx.setContextProperty("sinyaller", 0)
ctx.setContextProperty("arizalar", 0)
ctx.setContextProperty("Tarih", tarih)

def handleSerialData(): # seri porttan gelen veriyi isleyen fonksiyon
    global ctx #global degiskenlerin tanimlanmasi
    global data

    # tarih ve saatin yeniden hesaplanmasi
    now = datetime.now()
    tarih = now.strftime("%H:%M\n%d/%b/%y")

    # arayuz degiskenlernin yeni degerlerine atanmasi
    ctx.setContextProperty("hiz", data[0])
```

```
        ctx.setContextProperty("devir", data[1])
        ctx.setContextProperty("devir2", data[2])
        ctx.setContextProperty("hararet", data[3])
        ctx.setContextProperty("yakit", data[4])
        ctx.setContextProperty("batarya", data[5])
        ctx.setContextProperty("solKucuk", data[6])
        ctx.setContextProperty("ortaFps", data[7])
        ctx.setContextProperty("farlar", data[8])
        ctx.setContextProperty("sinyaller", data[9])
        ctx.setContextProperty("arizalar", data[10])
        ctx.setContextProperty("Tarih", tarih)

    engine.load('main.qml') # arayuzu olusturcak qml kodlarinin yuklenmesi

    timer=QTimer() # arayuz guncellemeleri icin kullanilcak timer
    timer.timeout.connect(handleSerialData) # timer tetiklendigi zaman cagirilacak fonksiyonun baglanmasi

    runThread = True # threadin calisip calismadigini tutan degisken

    def openSerialPort(): # seri portu acan fonksiyon

        # global degiskenlerin tanimlanmasi
        global runThread
        global timer
        global data

        # seri portun acilmasi
        serialPort = serial.Serial(port="/dev/ttyUSB0", baudrate=9600, bytesize=8, timeout=2, stopbits=serial.STOPBITS_ONE )

        serialString = "" # seri porttan gelen verinin atandigi deigisken

        while runThread: # seri den surekli deger okumak icin dongu
            if serialPort.in_waiting > 0: # seri port da veri kontrolu
                serialString = serialPort.readline() # seriporttan bir satir verinin okunmasi
                values = serialString.decode("Ascii") # gelen veriyi ascii string'e cevirme
                data = list(map(int, values.split())) # gelen verinin parcalanmasi ve global degiskene atanmasi

    thread = threading.Thread(target=openSerialPort) # seri porttan deger okunacak thread nesnesinin olusturulmasi
    thread.start() # thred' in baslatilmasi

    timer.start(100) # arayuzu guncelleyen timer'in baslatilmasi

    app.exec() # uygulamanin start edilmesi. bu fonksiyon icin bir event loop ceviri ve gelen isletim sistemi ve kullanic girdilerini arayuze ve ilgili elemanlara yollar. Bu kimsimda sonsuz dongu vardir program durana kadar dongu doner.
    runThread = False # seri port thread' inin durdurulmasi
```

sys.exit(1) # isletim sistmeine programin durdugu bilgisinin verilmesi. 1 cikti kodu ile cikis yapılır

EK-4

Qt QML (arayüz) kodları

```
import QtQuick 2.11
import QtQuick.Controls 1.4
import QtQuick.Window 2.4
import QtLocation 5.6
import QtPositioning 5.6

ApplicationWindow {
   id: applicationWindow

   title: qsTr("MRT CAR DEMO")
   visible: true
   visibility: "FullScreen"
   Rectangle {
      height: 480
      width: 1920
      color: "black"
      anchors.fill:parent

      Shortcut {
         sequence: "Esc"
         onActivated: applicationWindow.close()
      }

      /********************anapanel**********************/
      Image {
         x:379;y:1
         source: "image/anapanel.png"
      }
      //////////////
      Image {
         x:436;y:222
         source: "image/buyukibre.png"
         transform: Rotation
         {
            origin.x: 162; origin.y: 22; angle:(-33+hiz)
         }


      }
      Image {
         x:540;y:188
         source: "image/buyukorta.png"
         Text {
            text: parseInt(hiz*200/248)
            font.family: "Arial"
```

```
        font.pointSize: 25
        color: "#BCBCBC"
        font.bold:true
        anchors.left: parent.left
        anchors.right: parent.right
        anchors.top: parent.top
        anchors.topMargin: 22
        horizontalAlignment: Text.AlignHCenter
        verticalAlignment: Text.AlignVCenter

        Image {
            anchors.top: parent.bottom
            anchors.topMargin: 4
            anchors.horizontalCenter: parent.horizontalCenter
            source: "image/hiz.png"
        }
    }
}

Image {
    x:563;y:340
    source: "image/vites.png"
}
//////////////
Image {
    x:738;y:81
    source: "image/kucukibre.png"
    transform: Rotation {
        origin.x: 67; origin.y: 16; angle:(-85+solKucuk)}//+263
}
Image {
    x:777;y:74
    source: "image/kucukorta.png"
}
//////////////
Image {
    x:887;y:77
    source: "image/kucukibre.png"
    transform: Rotation {
        origin.x: 67; origin.y: 16; angle:(-43+ortaFps)}//+265
}
Image {
    x:927;y:65
    source: "image/kucukorta.png"
}
//////////////
Image {
    x:1040;y:84
    source: "image/kucukibre.png"
    transform: Rotation {
```

```
        origin.x: 67; origin.y: 16; angle:(-77+yakit)}//+154
}
Image {
   x:1081;y:81
   source: "image/kucukibre.png"
   rotation:180
   transform: Rotation {
      origin.x: 21; origin.y: 15; angle:(45-batarya)}//-92
}
Image {
   x:1079;y:74
   source: "image/kucukorta.png"
}

//////////////
Image {
   x:850;y:89
   source: "image/miniibre.png"
   transform: Rotation {
      origin.x: 103; origin.y: 0; angle:(-1*hararet)} //-89
}

//////////////
Image {
   x:1155;y:222
   source: "image/buyukibre.png"
   transform: Rotation {
      origin.x: 162; origin.y: 22; angle:(-35+devir)}//+200
}
Image {
   x:1155;y:222
   source: "image/buyukibre.png"
   transform: Rotation {
      origin.x: 162; origin.y: 22; angle:(288-devir2)}//-86
}
Image {
   x:1261;y:188
   source: "image/buyukorta.png"
   Text {
      //x:1298;y:225
      text: "25"//derece
      font.family: "Arial";  font.pointSize: 25;  color: "#BCBCBC"
      font.bold:true; anchors.centerIn: parent
      horizontalAlignment: Text.AlignHCenter
      verticalAlignment: Text.AlignVCenter

      Image {
         anchors.left: parent.right
         anchors.leftMargin: 4
         anchors.verticalCenter: parent.verticalCenter
```

```
            anchors.verticalCenterOffset: -8
            source: "image/derece.png"
        }
    }
}

//////////////
/*****************************************/

/*******************solbolum**********************/
Image {
    x:0;y:0
    source: "image/sagarka.png"
}
Image {
    x:81;y:14
    source: "image/parksenoff.png"
    rotation:180
}
Image {
    x:81;y:402
    source: "image/parksenoff.png"
}
Image {
    x:68;y:84
    source: "image/araba.png"
}

Image {
    x:71;y:77
    source: farlar==1 ? "image/far.png"  : "image/faron.png"
}
Image {
    x:207;y:77
    source: farlar==1 ? "image/far.png"  : "image/faron.png"
}

Image {
    x:47;y:369
    source: sinyaller==1 ? "image/sinyaloff.png"  : "image/sinyalon.png"
}
Image {
    x:225;y:369
    source: sinyaller==1 ? "image/sinyaloff.png"  : "image/sinyalon.png"
    rotation:180
}

/*****************************************/

/*******************sagbolum**********************/
```

```
            Image {
                x:1582;y:0
                source: "image/solarka.png"
            }


            Plugin {
                id: mapPlugin
                name: "osm"
            }
            Map {
                x:1632;y:53
                width: 281
                height: 390

                plugin: mapPlugin
                center: QtPositioning.coordinate(38.358, 31.418)// medeniyetin baskenti
AKSEHIR:)
                zoomLevel: 14
            }


            /*****************************************/

            /*******************altbilgi**********************/
            Image {
                x:878;y:234
                source: "image/arababilgi.png"
            }
            Image {
                x:544;y:388
                source: "image/altpanel.png"
            }
            Image {
                x:748;y:429
                source: arizalar==1 ? "image/alticonlaron.png"  : "image/alticonlar.png"
            }
            Text {
                x:911;y:420
                text:Tarih;  font.family: "Arial"
                font.pointSize: 20
                color: "#BCBCBC"
                font.bold:true
                horizontalAlignment: Text.AlignHCenter
            }
        }
}
```

ÖZGEÇMİŞ