T.C.

GEBZE TEKNİK ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

GÖRÜNTÜ İŞLEME İLE OTONOM ROBOT ARAÇ TASARIMI

BERKAY TÜRK

BURAK ÖZŞAHİN

MEHMET ADANIR

LİSANS BİTİRME TEZİ

ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE

2022

T.C.

GEBZE TEKNİK ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

GÖRÜNTÜ İŞLEME İLE OTONOM ROBOT ARAÇ TASARIMI

BERKAY TÜRK

BURAK ÖZŞAHİN

MEHMET ADANIR

LİSANS BİTİRME TEZİ

ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI

PROF. DR. NECATİ ECEVİT

GEBZE

2022

T.C.

GEBZE TECHNICAL UNIVERSITY

ENGINEERING FACULTY

AUTONOMOUS WITH IMAGE PROCESSING ROBOT VEHICLE DESIGN

BERKAY TÜRK

BURAK ÖZŞAHİN

MEHMET ADANIR

LICENSE THESIS

DEPARTMENT OF ELECTRONIC ENGINEERING

THESIS SUPERVISOR

PROF. DR. NECATİ ECEVİT

GEBZE

2022

Bu çalışma ..../..../2022 tarihinde aşağıdaki jüri tarafından Elektronik Mühendisliği Bölümü'nde Lisans Bitirme Projesi olarak kabul edilmiştir.

Bitirme Projesi Jürisi

Danışman Adı		
Üniversite		
Fakülte		

Jüri Adı		
Üniversite		
Fakülte		

Jüri Adı		
Üniversite		
Fakülte		

Jüri Adı		
Üniversite		
Fakülte		

ÖZET

Bu çalışmada görüntü işleme algoritmaları ve sensör yazılımları kullanılarak otonom bir araç tasarımı üzerinde çalışmalar gerçekleştirilmiştir. Sensör ve otomasyon sistemi ardunio üzerinden kontrol edilmiştir. Sensör olarak ultrasonik mesafe sensörü ses aracılığıyla aracın önüne çıkan herhangi bir engeli ses dalgalarını yollayıp eğer bir cisim var ise tanımasına ve buna göre hareket etmesine olanak sağlamıştır. Bu sayede aracımız engellere takılmadan varacağı noktaya ulaşabilmektedir.

Görüntü işleme; sayısal olarak alınan görüntülerin işlenerek özelliklerinin ve yapılarının değiştirilmesini, geliştirilmesini ve bu görüntüler vasıtasıyla analizlerin yapılmasını sağlayan teknolojidir. Görüntü işleme algoritmaları kullanılarak dışarıdan rasberry pi'ye bağlanan kamera ile görüntü alınıp işlenerek tanımlı olan işaretleri, levhaları tanıyarak ona göre veriyi rasberry pi den seri haberleşme kullanılarak ardunio'ya aktarılmıştır. Ardunio'ya gelen bu veriler yorumlanıp aracın yönünü tayin etmesi sağlanarak istediğimiz rotada hareket etmektedir.

Anahtar Kelimeler: Görüntü işleme, Mesafe sensörü, Ardunio, Rasberry pi

SUMMARY

In this study, studies were carried out on an autonomous vehicle design using image processing algorithms and sensor software. The sensor and automation system are controlled via Arduino. As a sensor, the ultrasonic distance sensor sent sound waves to any obstacle in front of the vehicle through sound, allowing it to recognize an object if there is one, and act accordingly. In this way, our vehicle can reach its destination without getting stuck in obstacles.

Image processing; By processing the digitally received images, their properties and changes, develop and analyze their structures and analyzes through these images that technology makes it possible. Using the image processing algorithms, the camera connected to the rasberry pi from the outside, the image is taken and processed, the defined signs and plates are recognized and the data is transferred from the rasberry pi to the ardunio using the serial port. By using this data coming to Arduino, it is provided to determine the direction of the vehicle and moves on the route we want.

Key Words: Image Processing, Ultrasonic distance sensor, Ardunio, Rasberry pi

TEŞEKKÜR

Başta lisans eğitimimizde ve akademik hayatımızda desteğini ve yardımlarını hiçbir zaman esirgemeyip bilgisi ile bu çalışmanın oluşmasının yolunu açan danışmanımız PROF. DR. NECATİ ECEVİT' e,

Projemizde akademik olarak destek veren DR. ÖĞR. Üyesi Önder Şuvak' a, bize yardım eden ÖĞR. Görevlisi Murat Ceylan' a ve göstermiş olduğu desteklerinden dolayı ailemize en içten teşekkürlerimizi sunarız.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler ve Kısaltmalar		**Açıklamalar**
GPU	:	Graphics Processing Unit
σ	:	Standart sapma
SD	:	Secure Digital
USB	:	Universal Serial Bus
Hz	:	Hertz
YSA	:	Yapay Sinir Ağları
Gİ	:	Görüntü İşleme
HDMI	:	High Definition Multimedia Interface
GPIO	:	General Purpose Input/Output
SMD	:	Surface Mount Devices
GTÜ	:	Gebze Teknik Üniversitesi
k	:	Kilo
V	:	Volt
A	:	Amper
GPS	:	Global Positioning System
ROI	:	Region of Interest
ABD	:	Amerika Birleşik Devletleri
API	:	Application Programming Interface
G	:	Gradyan
Ɵ	:	Eğim
3D	:	Three Dimensional
mm	:	mili metre
CNN	:	Convolutional Neural Network
YOLO	:	You Only Look Once
%	:	Yüzde İşareti
tl	:	Türk lirası

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

1. GİRİŞ

Gerçek zamanlı çalışan sistemlerde görüntü işleme uygulamaları yapmak son zamanlarda oldukça popüler olan bir konu haline gelmiştir. Görüntülerden nesne tespiti yapma alanında kullanılan görüntü işleme algoritmaları birlikte kullanılarak, otonom otomobiller, otonom insansız hava araçları, yardımcı robot teknolojileri, taşımacılık sektörü için otonom araç teknolojileri gibi birçok alanda uygulamalar geliştirilmektedir [1]. Gerçekleştirilen bu çalışmada taşımacılık sektörü için yardımcı bir teknoloji olarak görüntü işleme ve nesne algılama yöntemleri kullanılarak işaretleri, levhaları tanıyarak bunlara göre yön tespit edilmesi amaçlanmıştır. Geleneksel görüntü işleme algoritmalarının aksine bu çalışmada derin öğrenme yöntemleri ile görüntü işleme algoritmaları birlikte kullanılmıştır. Nesne tespit etme alanında en iyi modellerden biri olan YOLO-V3 veri seti oluşturulmuştur. YOLOV3 modelinin işaretçi ve levhaları algılamada diğer modellerden daha iyi olduğu ortaya çıkarılmıştır. Saniyede 60 kare çalışma hızı ile YOLOV3 modeli gerçek zamanlı çalışan sistemlerde de kullanılabilmektedir.

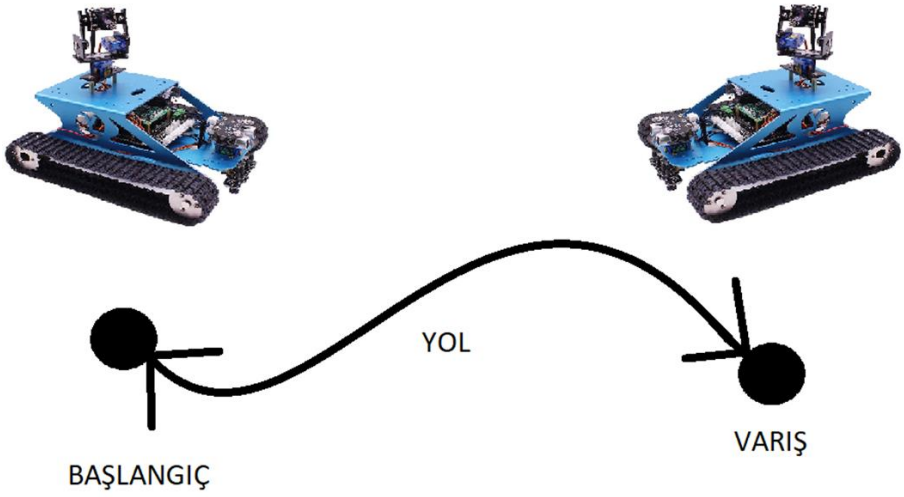
Bu proje için öncelikle aracın simülasyon ortamında test edilmesi gerçeklenmiştir. Bu simülasyon için ise CARLA simülasyon ortamından yararlanılmıştır. CARLA, sıfırdan otonom sürüş sistemlerinin geliştirilmesini, eğitimini ve onaylanmasını desteklemek için geliştirilmiş bir programdır. Simülasyon ortamında, sensörleri çevresel koşullarını, tüm statik ve dinamik aktörlerin tam kontrolünü, harita oluşturmayı ve çok daha fazlasını desteklemektedir.

Kullanılan yazılım dili ise Python olarak belirlendi. Bunun sebebi ise Python, nesne yönelimli, yorumlamalı, birimsel (modüler) ve etkileşimli yüksek seviyeli bir programlama dilidir. Python'un büyük bir standart kütüphanesi vardır. Bu sayede gerekli olan işlemler hızlı biçimde gerçekleşmektedir ve OpenCv kütüphanesi sayesinde görüntü işleme yapmak daha basit ve kullanışlı olmaktadır.

1.1. Tezin Amacı, Katkısı ve İçeriği

Görüntü işleme ve sensörler günümüzde birçok alanda yaygın olarak kullanılmaktadır. Günümüzdeki taşıma sistemlerinde zaman ve iş gücü kaybı çok fazladır. Bu proje sayesinde zamandan ve iş gücünden kaybı en aza indirmek istenmektedir.

Bu sayede az maliyetli etkin bir çözüm oluşturulabilmektedir.



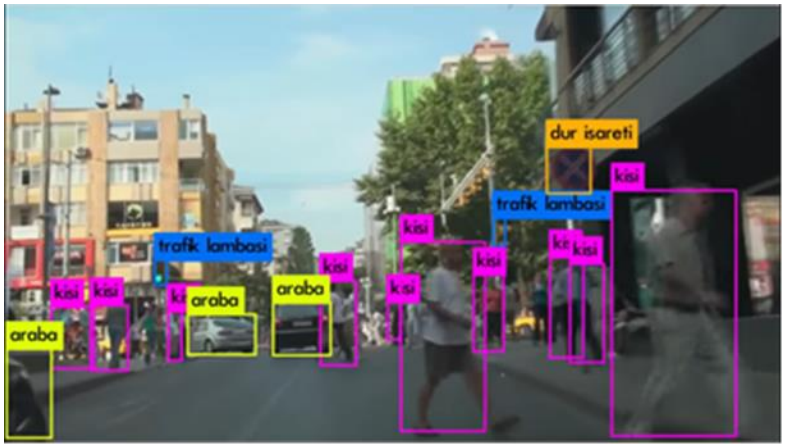
Şekil 1.1: Projenin Tasarım Amacı

Görüntünün eş zamanlı işlenmesi ve aracın buna göre hareket ettirilmesi bu tür işlemlerin bilgisayarlardaki işlem gücü ve hız ihtiyacının fazla olması nedeniyle zorlaşmaktadır. Bu zorluğun üstesinden gelebilmek için çeşitli yöntemler geliştirilmiştir. Şekil 1.1 de görüldüğü üzere bu projenin amacı en basit anlatım ile bir noktadan diğer bir noktaya otonom bir şekilde varmak ve belirlenen noktaya kargo teslimi gerçekleştirmektir.

Bu sistem ile gerçekleşen taşıma işlemi daha güvenli ve pratik olmaktadır. Örnek olarak bir kargo firmasının müşteriye göndermek istediği kargoyu bu projede tasarlanan otonom araç sayesinde hızlı ve güvenli bir şekilde müşteriye ulaştırmaktadır.

2. GÖRÜNTÜ İŞLEME

Görüntü İşleme (Gİ) askeri, endüstri, robotik, reklamcılık, astronomi, tıp, coğrafya, trafik gibi günlük yaşamın pek çok alanında kullanılan, alınan görüntünün işlenmesinden sonra devamında yapay sinir ağları (YSA), bulanık mantık gibi pek çok algoritma ile değerlendirilen bir teknolojidir. Günümüzde artık her işlemin insan yerine otonom sistemlere yaptırılması, görüntü işlemenin önemini bir kat daha artırmıştır.



Şekil 2.1: Görüntü İşleme Tanımı

Görüntü: 3 Boyutlu nesnelerin 2 Boyutlu yüzey üzerine düşürülmüş haritası olarak tanımlanabilir. Bu haritalamada her noktanın konum bilgisi f(x,y) ve renk bilgisi tutulur. Görüntü henüz sinyal şeklinde ise, insan gözünün görebileceği şekilde değilse resim haline gelmemiş demektir. Görüntü Analog ve Dijital görüntü olarak iki kısımda ele alınabilir [2].

Analog Görüntü: Analog görüntüde, görüntüyü oluşturan fonksiyonun-f(x,y) değişkenleri reel değerler alıyorsa (yani tüm sayıları kapsıyorsa) bu görüntü Analog görüntüdür. Analog bir görüntüde bir resme ne kadar yakından bakarsak bakalım (örneğin mikroskopla) orada hala görüntüyü oluşturan renkler bulunur. Sayısal bilgisayarlar, sürekli fonksiyonları/parametreleri işleyemezler. Bu fonksiyonların sayısallaştırılması gerekir.

Dijital (Sayısal) Görüntü: f(x,y) şeklinde temsil edilen sürekli görüntüyü (analog görüntü) ayrık örnekler cinsinden (discrete) ifade edilmesidir ve gösterimi f[x, y] şeklindedir.

3. YAPIM AŞAMASINDA KULLANILAN MALZEMELER

KULLANILAN MALZEMELER	ADET	FİYAT
Raspberry pi 3B	1	935tl
Robot şase	1	30tl
Motor	4	18.37tl
Motor Sürücüsü	1	25.96tl
Tekerlek	4	20tl
Mesafe Sensörü	1	15.60tl
Arduino Uno	1	110.95tl
Çizgi Takip Sensörü	2	20tl
TOPLAM		1175,88tl

Tablo 3: Malzeme Fiyat Tablosu

Tablo 3 de görüldüğü üzere kullanılan malzeme fiyat tutarı 1175,88tl olmaktadır.

3.1. Raspberry Pi

Raspberry Pi, Raspberry Pi Vakfı tarafından Birleşik Krallık'ta geliştirilen, Kredi kartı boyutundaki işletim sistemli bir bilgisayardır. VideoCore IV GPU grafik işlem birimine sahip, Booting ve veri depolaması için SD kart kullanan üzerinde USB 2.0 portları, Ethernet portu, HDMI video çıkışı, ses çıkışı, MIPI kamera girişi, GPIO arayüzü ve 5V MicroUSB güç girişi bulunmaktadır. Dünyada çocukların kodlama ve elektroniği sevmesi için geliştirilen Raspberry Pi düşük maliyeti ve açık kaynaklı bir bilgisayar olması nedeniyle beklenenden daha popüler hale geldi [3].

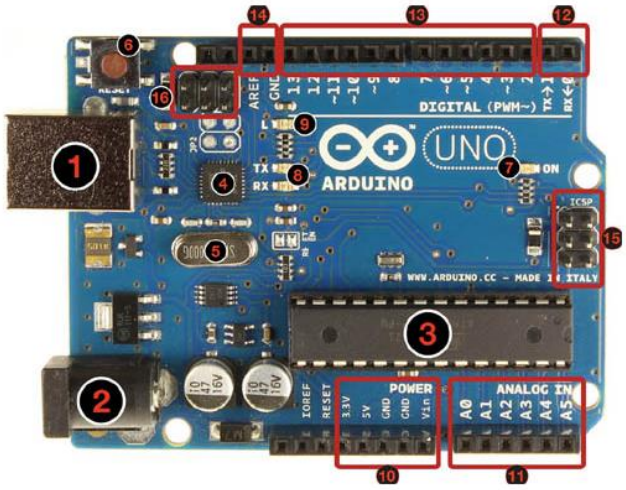
Şekil 3.1: Raspberry Pi 3B

3.2. Arduino Kartı

Arduino Uno ATmega328 mikrodenetleyici içeren bir Arduino kartıdır. Arduino 'nun en yaygın kullanılan kartı olduğu söylenebilir. Arduino Uno 'nun ilk modelinden sonra Arduino Uno R2, Arduino Uno SMD ve son olarak Arduino Uno R3 çıkmıştır. Arduino 'nun kardeş markası olan Genuino markasını taşıyan Genuino Uno kartı ile tamamen aynı özelliklere sahiptir [4].

Şekil 3.2: Arduino UNO Kartı

Arduino kartının giriş çıkış pinleri aşağıdaki şekilde gösterilebilir.



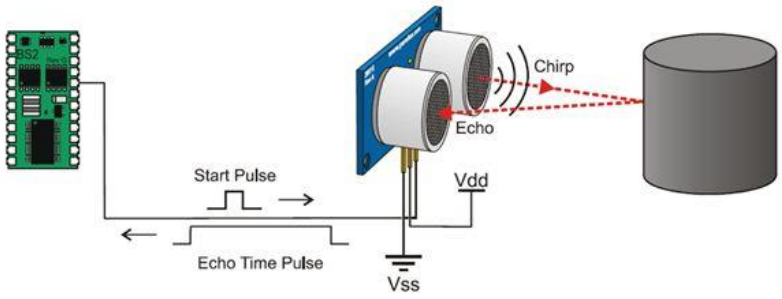
1 : USB jakı
2 : Power jakı (7-12 V DC)
3 : Mikrodenetleyici ATmega328
4 : Haberleşme çipi
5 : 16 MHz kristal
6 : Reset butonu
7 : Power ledi
8 : TX / NX ledleri
9 : Led
10 : Power pinleri
11 : Analog girişler
12 : TX / RX pinleri
13 : Dijital giriş / çıkış pinleri
14 : Ground ve AREF pinleri
15 : ATmega328 için ICSP
16 : USB arayüzü için ICSP

Şekil 3.2.1: Arduino UNO Kartının giriş/çıkış Pinleri

3.3. HC- SR04 Ultrasonik Sensör

HC- SR04 Ultrasonik Sensör sonar iletişim kullanarak karşısındaki nesneye olan mesafeyi hesaplayan bir kaynaktır. Sonar dediğimiz sistem ses dalgalarını kullanarak cismin uzaklığı hesaplamaya yardım eder. Bu tür sensörlerin esin kaynağı yunuslar ve yarasalardır. Yunuslar ve yarasalar da ses dalgası göndererek karşısına çıkabilecek

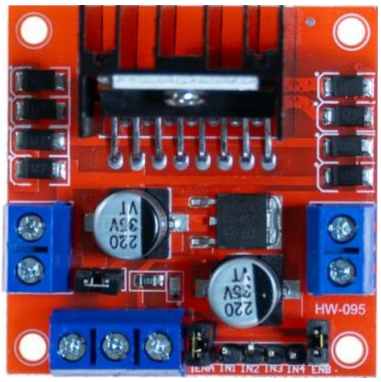
engellerin mesafelerini hesaplamaktadırlar. Sensör üzerinde 4 adet pin mevcut. Bunlar; vcc, gnd, trig, echo pinleridir. Sensörü kullanmak için Trig pininden uygulanan sinyal 40 kHz frekansında ultrasonik bir ses yayılmasını sağlar. Bu ses dalgası herhangi bir cisme çarpıp sensöre geri döndüğünde, Echo pini aktif hale gelir [5].



Şekil 3.3: Ultrasonik Sensörün Çalışma Prensibi

3.4. L298N DC Step Motor Sürücü

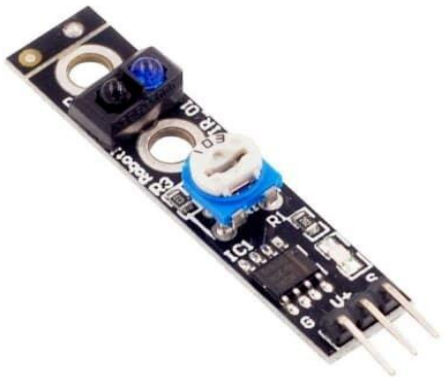
12V'a kadar olan motorları sürmek için hazırlanmış olan bu motor sürücü kartı, iki kanallı olup, kanal başına 2A akım vermektedir. Birbirinden bağımsız olarak iki ayrı motoru kontrol edebilir. Yüksek sıcaklık ve kısa devre koruması vardır. Motor dönüş yönüne göre yanan ledler vardır. Kart üzerinde dahili soğutucu vardır. Akım okuma (current sense) pinleri dışa verilmiş haldedir. Kart üzerinde L298N motor sürücü entegresi kullanılmıştır. Sumo, mini sumo, çizgi izleyen robotlarda ve çok çeşitli motor kontrol uygulamalarında kullanılabilmektedir [6].



Şekil 3.4: Motor Sürücüsü

3.5. Çizgi Takip Sensörü- KY-033 TCRT5000

Çizgi Takip Sensörü Modülü, siyah ve beyaz çizgileri algılamada ve bunları işlemede kullanılan elektronik modüldür. Üzerindeki TCRT5000 kızılötesi sensör, hassasiyet trimpotu ve LM393 entegresi aracılığıyla algılama işlemini gerçekleştirmektedir. Temel çalışma mantığı, sensörden gönderilen kızılötesi ışığın tekrar yansımasını kullanarak işleme almasıdır. Düşük güç tüketimine sahip bu modül, batarya dostu bir kullanıcı deneyimi sunar. [7]



Şekil 3.5: Çizgi Takip Sensörü

4. SİMÜLASYON ORTAMI

4.1. CARLA

CARLA, açık kaynaklı bir otonom sürüş simülatörüdür. Otonom sürüş problemiyle ilgili bir dizi görevi ele almak için modüler ve esnek bir API olarak hizmet vermek üzere sıfırdan oluşturuldu.

CARLA'nın ana hedeflerinden biri, kullanıcılar tarafından kolayca erişilebilen ve özelleştirilebilen bir araç olarak hizmet veren otonom sürüş Ar-Ge'sini demokratikleştirmeye yardımcı olmaktır. Bunu yapmak için, simülatörün, genel sürüş problemi (örneğin, sürüş politikalarını öğrenmek, eğitim algılama algoritmaları, vb.) Dahilindeki farklı kullanım durumlarının gereksinimlerini

karşılaması gerekir. CARLA simülasyonu Unreal Engine üzerine kurulmuştur, yolları ve kentsel ayarları tanımlamak için OpenDRIVE standardını kullanır. Simülasyon üzerinde kontrol, proje yaptıkça sürekli büyüyen Python ve C ++ 'da ele alınan bir API aracılığıyla sağlanır [8].

Şekil 4.1: CARLA Simülasyon Görüntüsü

CARLA simülatörü, ölçeklenebilir bir istemci-sunucu mimarisinden oluşur. Sunucu, simülasyonun kendisiyle ilgili her şeyden sorumludur: sensör oluşturma, fiziğin hesaplanması, dünya durumu ve aktörleri hakkında güncellemeler ve çok daha fazlası. İstemci tarafı, sahnedeki aktörlerin mantığını kontrol eden ve dünya koşullarını belirleyen bir dizi müşteri modülünden oluşur. Bu, sunucu ve istemci arasında aracılık eden ve yeni işlevler sağlamak için sürekli gelişen bir katman olan CARLA API'den (Python veya C ++) yararlanılarak elde edilir.

4.2. CARLA ile Şerit Takip Sistemi

Öncelikle şerit takibi yapıla bilmesi için şekilde 4.2 de görüldüğü gibi kenar tespit çalışması yapılmalıdır. Bu kenar tespitinin yapılması için ise Canny Edge Tespit algoritması kullanıldı.

Canny Kenar Tespiti algoritması 5 adımdan oluşur:

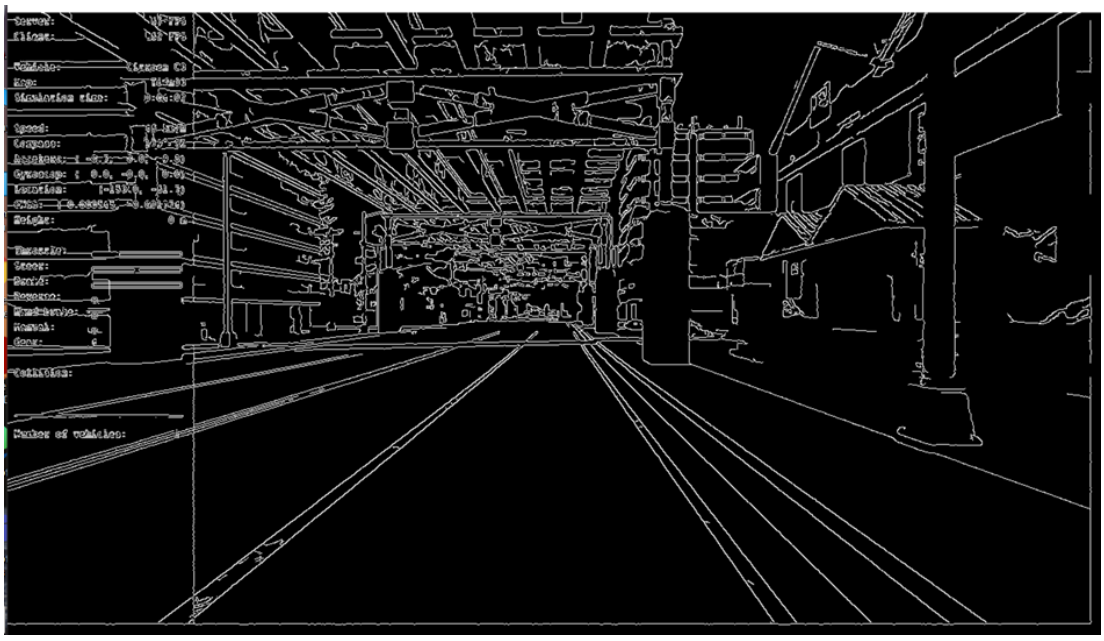
Gürültü azaltma;

Gradyan hesaplaması;

Maksimum olmayan bastırma;

Çift eşik;

Histerezis ile Kenar İzleme.



Şekil 4.2: CARLA Kenar Tespiti

Gürültü Azaltma:

Sahne arkasında yer alan matematik esas olarak türevlere dayandığından, kenar algılama sonuçları görüntü gürültüsüne karşı oldukça hassastır.

Görüntüdeki parazitten kurtulmanın bir yolu, onu yumuşatmak için Gauss bulanıklığı uygulamaktır. Bunun için Gaussian Kernel (3x3, 5x5, 7x7 vb…) ile görüntü evrişim tekniği uygulanır. Çekirdek boyutu, beklenen bulanıklaştırma etkisine bağlıdır. Temel olarak, çekirdek ne kadar küçükse, bulanıklık o kadar az görünür.

(2 k +1)×(2 k + 1) büyüklüğünde bir Gauss filtre çekirdeği denklemi şu şekilde verilir:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1) \tag{4.2}$$

Gradyan Hesaplama:

Gradyan hesaplama adımı, kenar algılama operatörlerini kullanarak görüntünün gradyanını hesaplayarak kenar yoğunluğunu ve yönünü tespit eder [9].

Kenarlar, piksel yoğunluğunun değişmesine karşılık gelir. Bunu tespit etmenin en kolay yolu, bu yoğunluk değişimini her iki yönde de vurgulayan filtreler uygulamaktır: yatay (x) ve dikey (y)

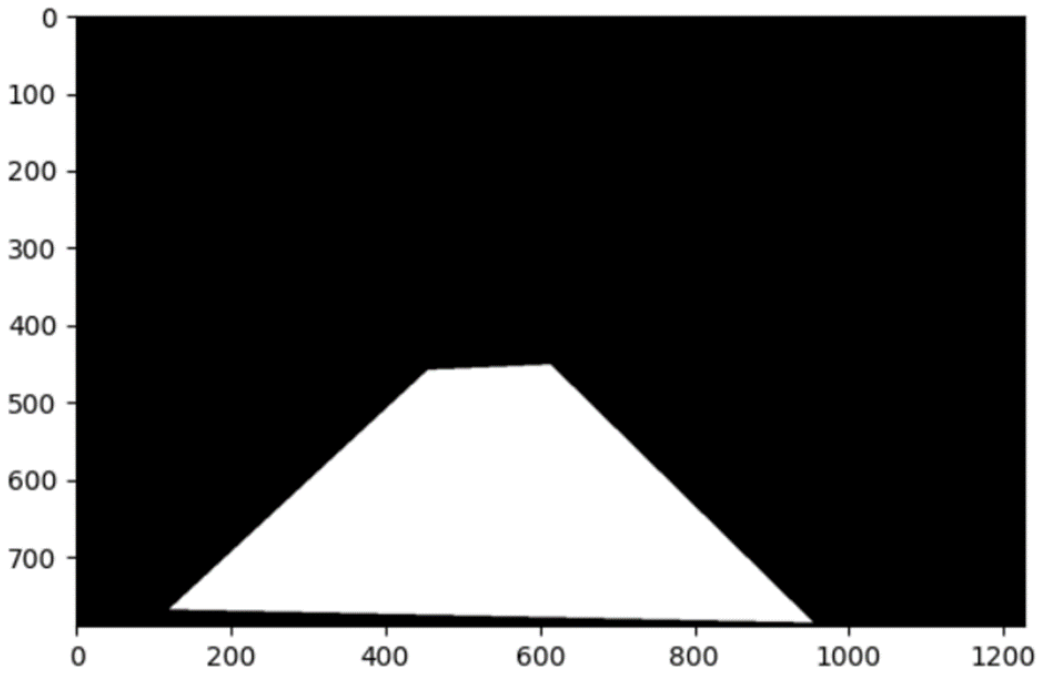
Görüntü yumuşatıldığında, Ix ve Iy wrt x ve y türevleri hesaplanır. Sırasıyla, I Sobel çekirdekleri Kx ve Ky ile evrilerek uygulanabilir:

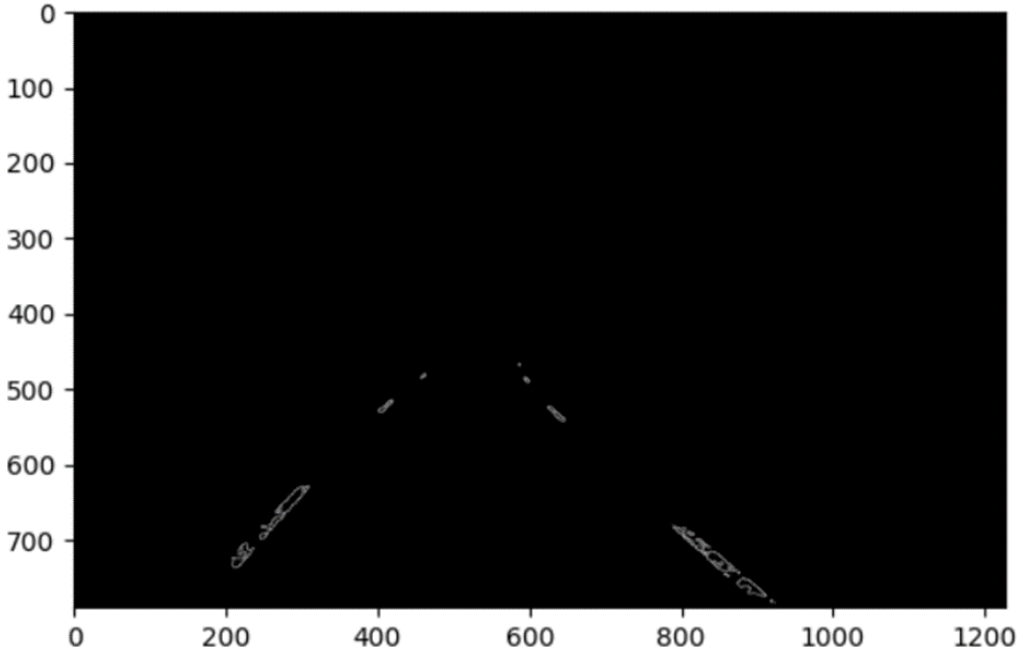
$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}. \quad (4.2.1)$$

Daha sonra, gradyanın G büyüklüğü ve eğimi θ aşağıdaki gibi hesaplanır:

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x, y) = arctan\left(\frac{I_y}{I_x}\right) \quad (4.2.2)$$

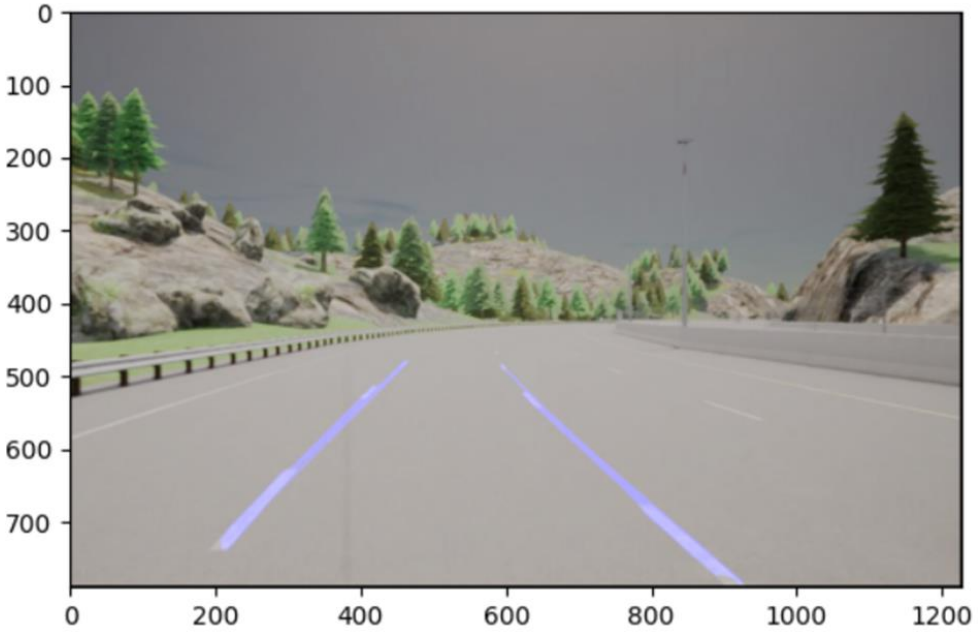
Bu aşamada bizimle alakası olmayan bir sürü nesne karşımıza çıkacak Şekil 4.2 de görüldüğü üzere (Yoldaki beyaz arabalar kenarlardaki beyaz nesneler vs.). Bunları elimine etmek için sadece arabanın önündeki kısmı alalım. Buna literatürde ROI (Region of Interest: İlgilenilen alan) denir [10].





Şekil 4.2.1: ROI Alanı

Şekilde 4.2.1 de görüldüğü üzere istenilen ROI çıkarımı yapılmıştır.



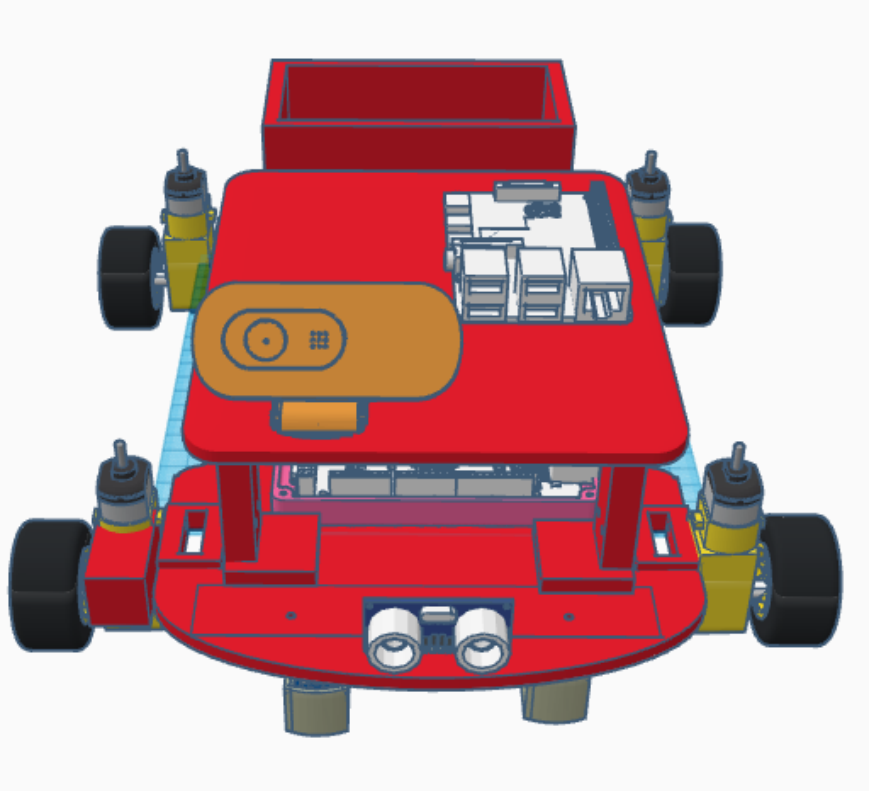
Şekil 4.2.2: CARLA Şerit Algılama

Şekil 4.2.2 de görüldüğü üzere simülasyon ortamında araç ile şerit takibi sistemi gerçek zamanda olacak bir şekilde yapıldı.

5. ROBOT TASARIMI

5.1. Tasarım Modellemesi

Tasarladığımız otonom araç kargo veya yük taşımaya uygun şekilde dizayn edilip bu tasarım uygun tasarım programı seçilerek çizimi Tinkercad ortamında yapılmıştır. Tinkercad, herkesin kolayca kullanabileceği tarayıcı tabanlı 3D tasarım ve modelleme aracıdır. Online olarak çalışmaktadır. Windows, Mac veya Linux üzerinde tüm web tarayıcılarında çalışır. Seneler içerisinde tasarım, modelleme, elektronik devre tasarımları, kodlama gibi alanlarda da kendisini geliştirmiş bir uygulama olmuştur [11]. Ardından 3D yazıcı ile robotun mekanik parçaları (şasi, GPS tutucu, motor yuvaları vs.) üretimi yapılmıştır.

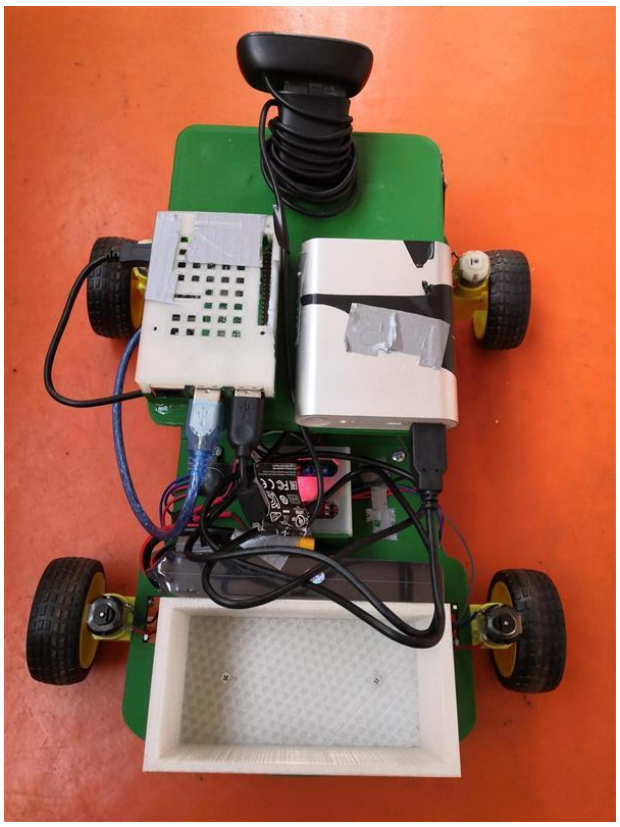


Şekil 5.1: Tinkercad Modellemesi

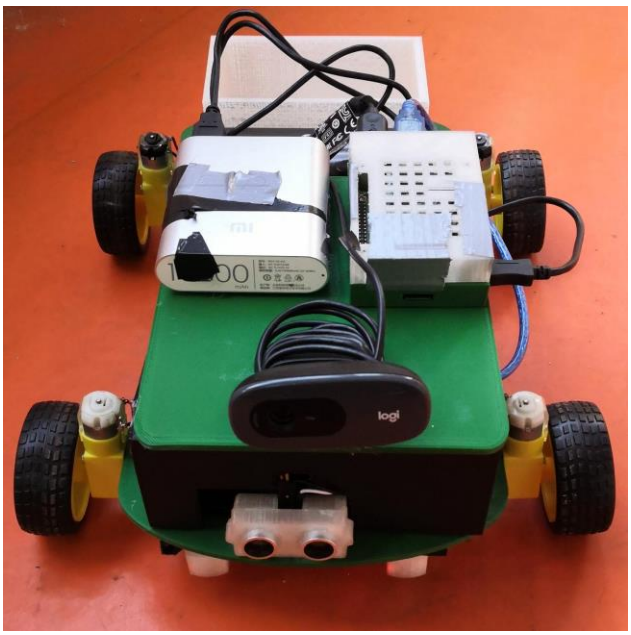
Şekil 5.1 de görüldüğü gibi Tinkercad modellemesi yapılmıştır. Otonom aracın boyutları yaklaşık olarak boyu 366.00 mm , eni 228.00 mm , yüksekliği ise 251.00 mm olarak tasarlanmıştır. Elektronik komponentler aracın alt kısmında yer almaktadır. Üst kısım ise yük taşımak için uygun şekilde tasarlanmıştır.

5.2. Tasarım Çıktısı

3D yazıcıdan elde edilen parçalar ve gerekli elektronik komponentler birleştirilerek aşağıda gösterilen şekillerdeki gibi bir araç elde edilmiştir.



Şekil 5.2: Otonom Araç Üstten Görünüm



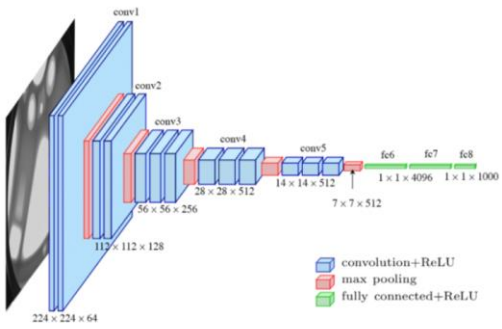
Şekil 5.2.1: Otonom Araç Önden Görünüm

Şekiller de görüldüğü gibi araç eşya taşımak için ayrıca bir yere sahiptir. İç tarafta gözüken kablo karmaşasını gizlemek için ayrıca açılır kapaklar tasarlanmıştır.

6. GÖRÜNTÜ İŞLEMEDE İNCELENEN YÖNTEMLER

6.1. Görüntü Sınıflandırma Mimarisi

Önceden de bahsedildiği gibi görüntü işleme birçok farklı alanda kullanılmaktadır (askeri, endüstri, robotik vb). Tasarımını yaptığımız otonom kargo aracı gerçek zamanlı nesne tanıma yapabilmek için görüntüyü sınıflandırmalıdır. Bu nedenle öncelikle görüntü sınıflandırma mimarileri incelenmiş olup VGG-16 mimarisinin daha doğru sonuç getireceğine karar verilmiştir.

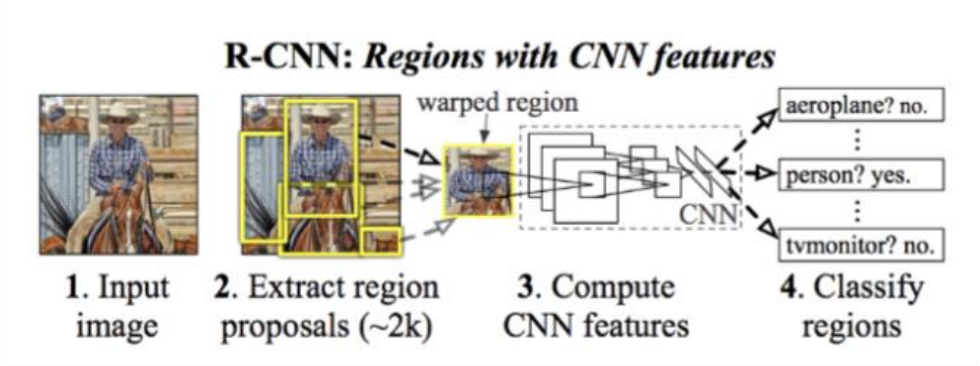


Şekil 6.1: Görüntü Mimarisi

Görüntü sınıflandırması için evrişimli bir sinir ağında bulunan VGG-16, aynı veri kümesinde eğitilmiştir ve GPU'lar üzerinde eğitimden daha fazlasına sahiptir. Daha sonra ise nesne tanımanın tam anlamıyla gerçek zamanlı olabilmesi için Opencv kütüphanesi içinde bulunan ve yaygın şekilde kullanılan algoritmalar incelenmiştir (YOLO, original R-CNN, Fast R- CNN vb.).[12]

Görüntüden farklı ilgi alanlarını almak ve o bölgedeki nesnenin varlığını sınıflandırmak için bir CNN kullanmak gerekmektedir. Bu yaklaşımla ilgili sorun, ilgilenilen nesnelerin görüntü içinde farklı uzamsal konumlara ve farklı en boy oranlarına sahip olmasıdır. Bu nedenle, çok sayıda bölge seçmek zorunda kalacaksınız ve bu hesaplamada sorun yaratabilir. Bu nedenle, R-CNN, YOLO gibi algoritmalar bu oluşumları bulmak ve hızlı bir şekilde kullanmak için geliştirilmiştir.

6.2. R-CNN Algoritması



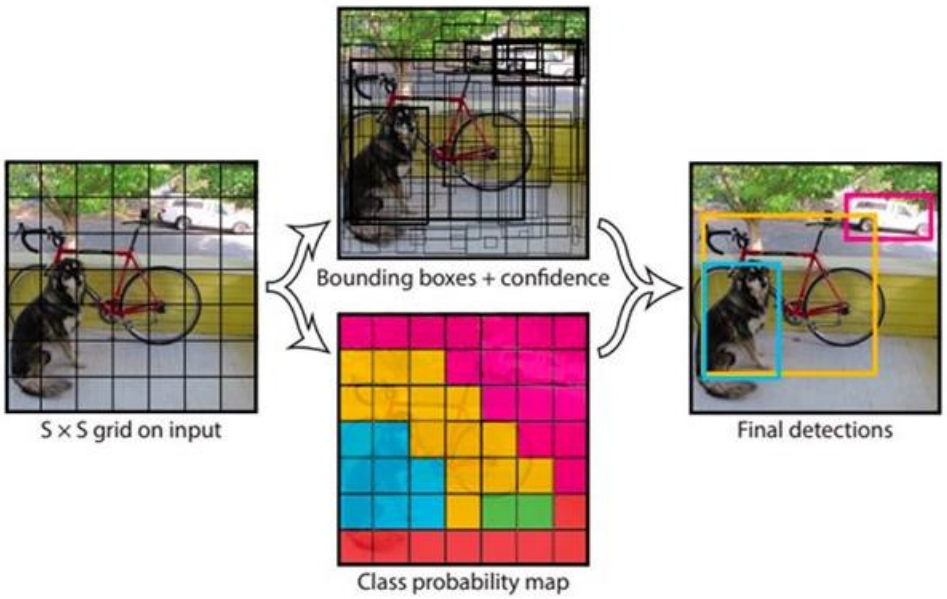
Şekil 6.2: R-CNN Algoritması

1.fotoğrafta işlem yapılacak giriş görüntümüz mevcut. 2. görüntüde ise çıkartılabilecek 2000 tane aday bölge önerisi yer almakta ve bu bölgeler bir kareye çarpıtılmaktadır. Çıktı olarak ise 4096 (64x64) boyutlu bir özellik vektörü üreten evrişimli bir sinir ağına atandı. CNN bir özellik çıkarıcı olarak işlev görür ve çıktı yoğun katman, görüntüden çıkartılan özelliklerden oluşur ve elde edilen özellikler, o aday bölge teklifindeki nesnenin varlığını sınıflandırmak için bir SVM'ye beslenir. 3. görüntüde CNN özellikleri de devreye girerek kendine has algoritması ile hesaplama yapar ve ardından belirli bölgelere göre görüntü sınıflandırması yapar. Kesit olarak aldığı bölgeye uçak, tv monitörü veya insan gibi özellikleri sorgulayarak doğruyu tahmin eder.[13]

6.3. YOLO (You Only Look Once) Algoritması

YOLO, konvolüsyonel sinir ağlarını (CNN) kullanarak nesne tespiti yapan bir algoritmadır. Açılımı "You Only Look Once", yani "Sadece Bir Kez Bak". Bu adın seçilmesinin nedeni algoritmanın nesne tespitini tek seferde yapabilecek kadar hızlı olmasıdır. YOLO algoritması çalışmaya başladığında görüntülerdeki veya videolardaki nesneleri ve bu nesnelerin koordinatlarını aynı anda tespit eder. YOLO algoritması, öncelikle görüntüyü bölgelere ayırır. Daha sonra her bir bölgedeki nesneleri çevreleyen kutuları (bounding box) çizer ve her bir bölgede nesne bulunma olasılığı ile ilgili bir hesabı yapar. YOLO algoritmasının hızlı olmasının sebebi ise elindeki görüntüyü tek bir seferde nöral ağdan geçirerek resimdeki tüm nesnelerin sınıfını ve koordinatlarını tahmin edebilmesidir. Yani bu tahmin işleminin

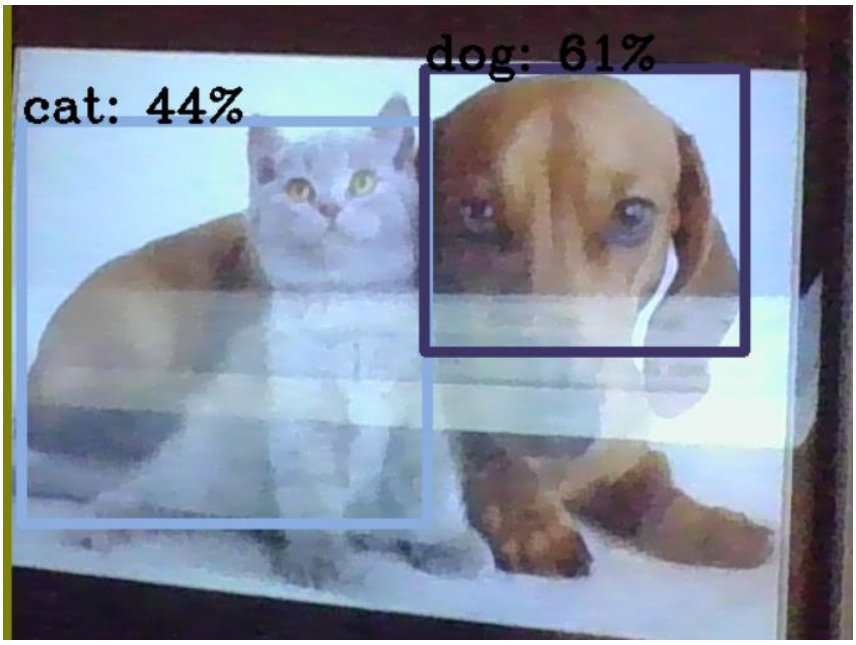
temeli, nesne tespitini tek bir regresyon problemi olarak ele almasında yatmaktadır. Algoritma ilk önce input görüntüsünü SxS'lik ızgaralara bölüyor. Bu ızgaralar 5x5, 9x9 veya 21x21'lik kutucuklara bölünmüş olabilir. Bu görüntüdeki her bir ızgara kendi içerisinde, bölgede nesnenin var olup olmadığını eğer varsa orta noktasının içinde olup olmadığını, orta noktası da içerisinde bulunuyor ise uzunluğunu, yüksekliğini ve hangi sınıftan olduğunu bulmakla sorumludur. YOLO, her bir oluşturduğu ızgara adına bir adet tahmin vektörü oluşturmaktadır.[14]



Şekil 6.3: YOLO Algoritmasının Çalışma Mantığı

Şekil 6.3.1: YOLO ile İnsan Algılama

Şekil 6.3.1 de görüldüğü üzere Yolo algoritması kullanarak insan algılama yapıldı. Şekil 6.3.1 de insan tespitini %72 oranında algılandığı tespit edildi.



Şekil 6.3.2: YOLO ile Hayvan Algılama

Şekil 6.3.2 de görüldüğü üzere Yolo algoritması kullanarak köpek ve kedi algılaması yapıldı. Şekil 6.3.2 de köpek tespitini %61 oranında algılama gerçekleşirken kedi tespiti %44 oranında elde edildi.

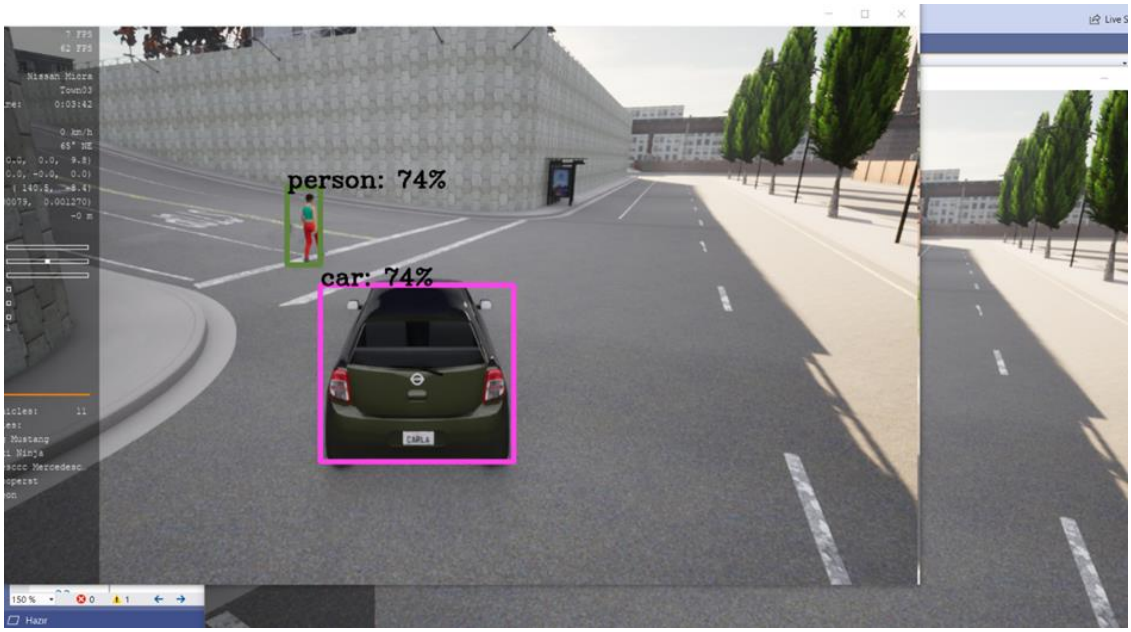
Şekil 6.3.3: YOLO ile Cisim Algılama

Şekil 6.3.3 de görüldüğü üzere Yolo algoritması kullanarak araba algılaması yapıldı. Şekil 6.3.3 de araba tespitini %74 oranında algılandığı tespit edildi.

6.4. CARLA Simülasyon Ortamında, YOLO Algoritması ile Tam Zamanlı Görüntü İşleme

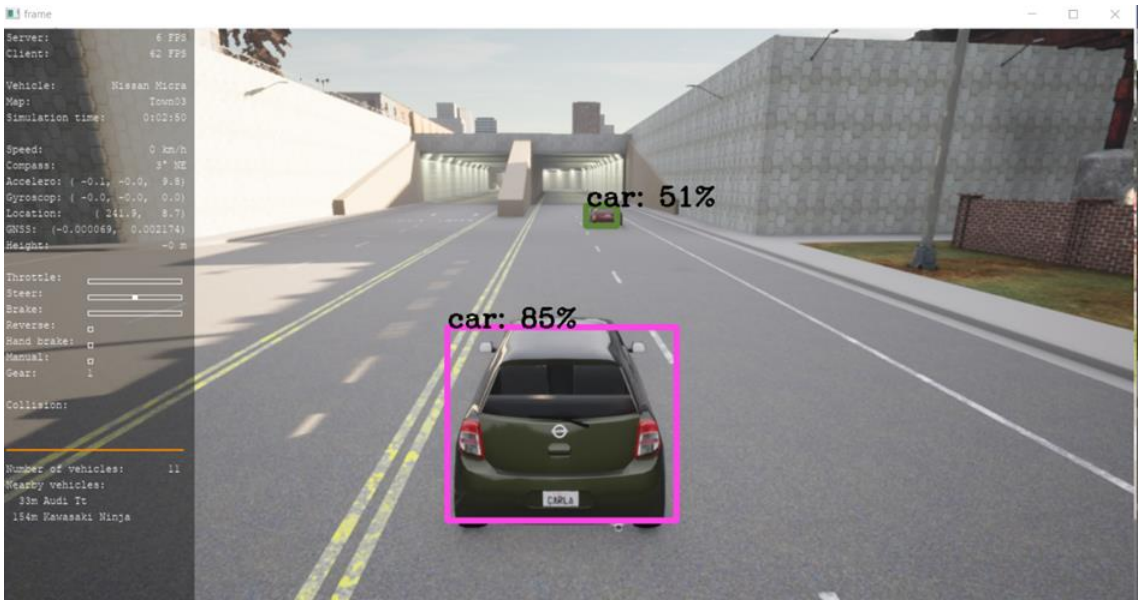
Şekil 6.4: Trafik Işık Algılama

Şekil 6.4 de görüldüğü üzere CARLA simülasyon ortamında tam zamanlı görüntü işleme yapılarak Trafik ışığı algılandı.



Şekil 6.4.1: YOLO ile İnsan Algılama

Şekil 6.4.1 de görüldüğü üzere CARLA simülasyon ortamında tam zamanlı görüntü işleme yapılarak insan algılandı.

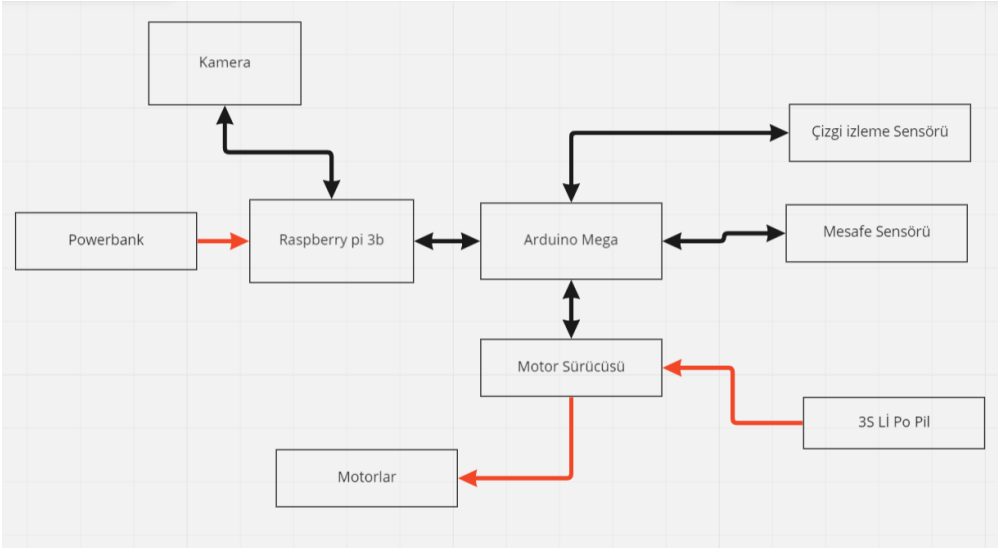


Şekil 6.4.2: YOLO ile Cisim Algılama

Şekil 6.4 de görüldüğü üzere CARLA simülasyon ortamında tam zamanlı görüntü işleme yapılarak arabalar algılandı.

7. PROJENİN YAZILIM, DONANIM ve TEST AŞAMASI

7.1. Yapım Aşamasında Kullanılan Malzemelerin Entegresi



Şekil 7.1: Robot araç blok diagramı

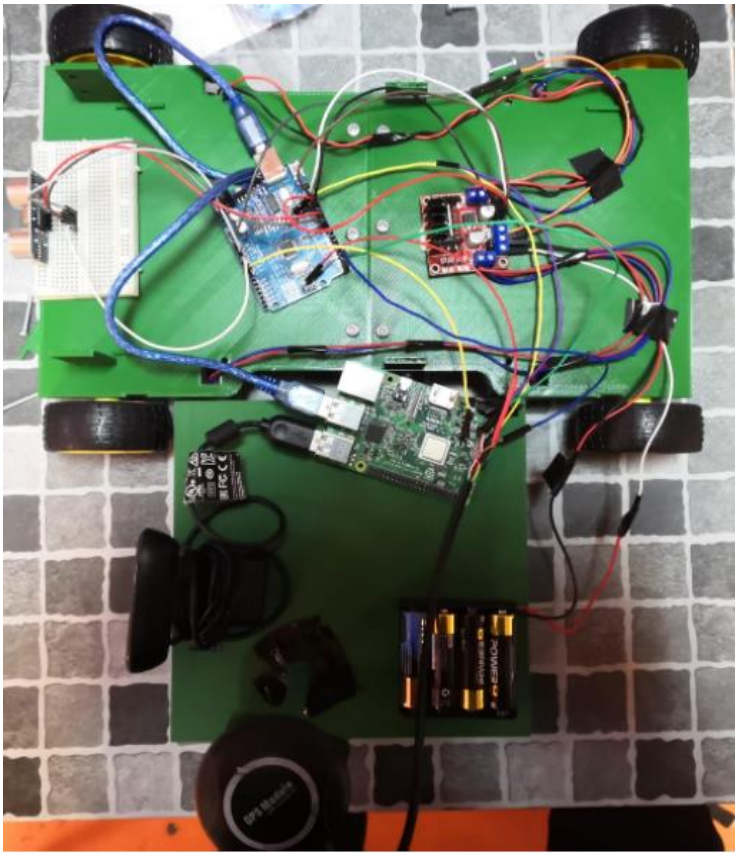
Şekil 7.1 de görüldüğü gibi aracın çalışma sistemi şu şekildedir;

Güç sistemi: Raspberry pi 3b Powerbank sayesinde çalışmaktadır. Raspberry pi den gelen güç ile de Kamera çalışmaktadır.

3S Li Po pilden çıkan 11.1 V ile Motor sürücüyü beslemektedir. Motor sürücüden çıkan 5V ile Arduino Mega kartı çalışmaktadır. Arduino kartından çıkan 5V'lar ile de hem mesafe sensörü hem de çizgi izleme sensörleri çalışmaktadır. Motor sürücüye bağlanan motorlar ise 3S Li Po pil ile çalışmaktadırlar.

Haberleşme Sistemi: Raspberry pi ve Arduino arasında gerçekleşen haberleşme USB kablo sağlanmaktadır. Kullanılan protokol ise Seri haberleşme protokolüdür. Kamera ve Raspberry pi arasındaki haberleşme kablolu ve USB haberleşme protokolüdür.

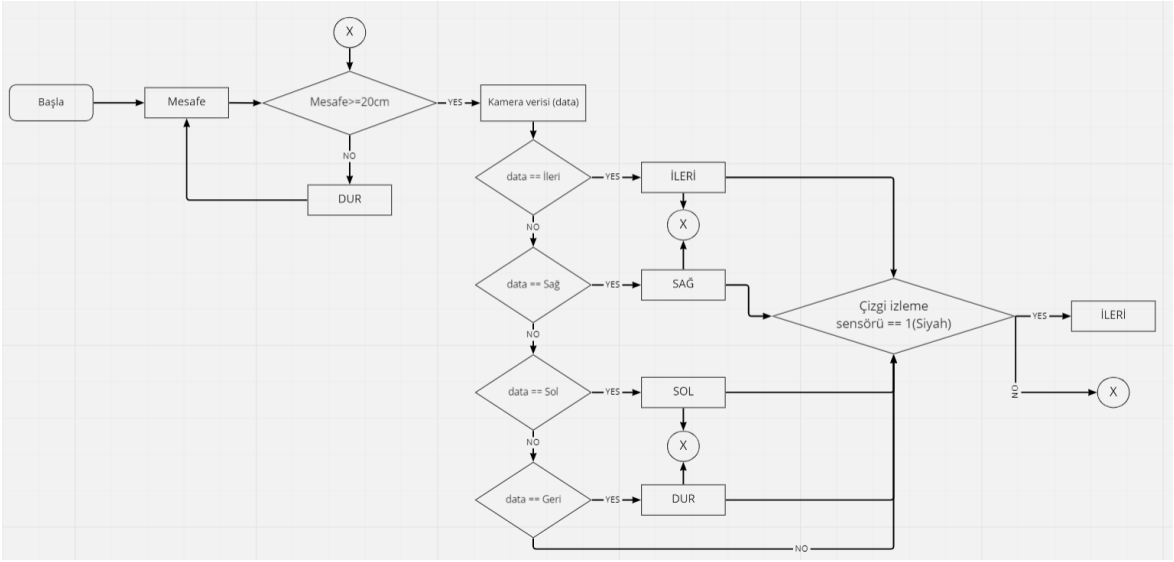
Yapım aşamasında kullanılan elektronik devreler ve malzemeler otonom araca doğru şekilde entegre edilmiştir. Aracın devre sistemleri alt kısmına sabitlenerek herhangi bir kaza halinde elektronik elemanlara olan zarar en aza indirgenmiştir. Ayrıca elektronik komponentlerin input ve outputları kontrol edilmiş olup, bataryadan bütün elektronik devrelere giden gücün kayıpsız bir şekilde gittiği test edilmiştir.



Şekil 7.2: Elektronik Devre

Şekil 7.2 de görüldüğü üzere Robot aracın elektronik devre sistemi kuruldu.

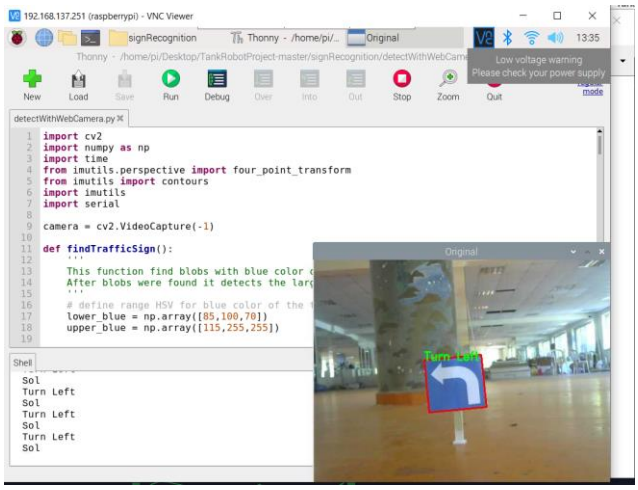
7.2. Kod Yazımı

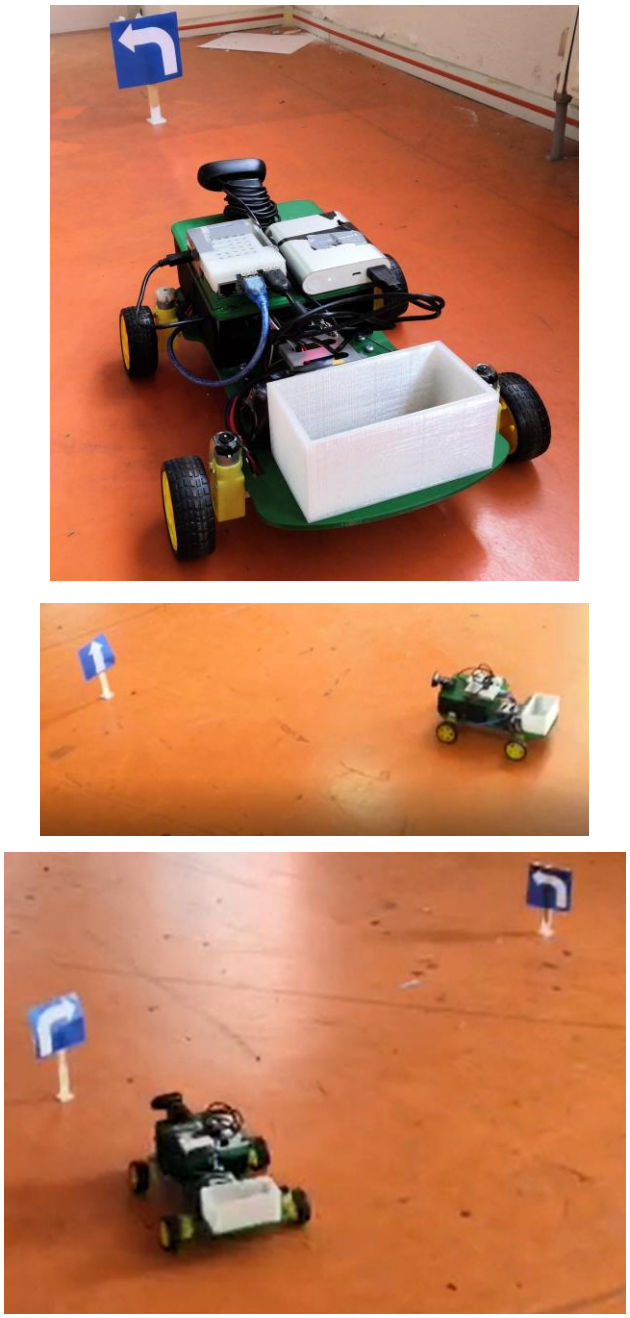


Şekil 7.3: Kodların Flowchartı

Sistem her zaman mesafe sensörü ile mesafeyi ölçüp 20cm den küçük ise araç duracaktır. Raspberry kamera sayesinde sürekli görüntü işleme ile tabela algılamakta ve algıladığı tabelayı paket ile arduino'ya iletmektedir. Sistem önünde herhangi bir cisim olmadığı zaman raspberry den gelen veriyi okuyarak aracın ileri, sağa, sola veya duracağına karar vermektedir. Eğer herhangi bir tabela olmadığı zaman ise çizgi izleme sensörleri sayesinde şerit takibi yapmaktadır.

7.3. Test Aşaması





Şekil 7.4: Görüntü işleme ile tabela algılama

Şekil 7.4 de görüldüğü üzere Raspberry pi anlık aldığı görüntüyü işleyerek tabela okuması gerçekleşmiştir. Bu görselde sol tabelası algılanmıştır ve çıktı olarak bizlere sol verisi vermektedir.

Şekil 7.5: Şerit Takip Sistemi

Şekil 7.5 de görüldüğü üzere araç 2 sensör sayesinde siyah şeritleri şeritler bitene kadar takip etmektedir.

Şekil 7.6: Mesafe Sensör Testi

Şekil 7.6 da görüldüğü üzere araç önünde cisim olduğu zaman mesafe sensörü yardımı ile önceden algılayıp durmaktadır.

8. SONUÇLAR

8.1. Algoritma Seçim Sonucu

RCNN gibi bölge temelli nesne tespit algoritmaları öncelikle nesne tespiti alanında nesnenin bulunması için muhtemel alanları belirler ve ardından bu bölgelerde ayrı ayrı CNN (Convolutional Neural Network, Evrişimsel Sinir Ağları) sınıflandırıcıları yürür. Bu sebeple RCNN algoritmaları YOLO' ya göre daha yavaş çalışmaktadır. Tasarladığımız otonom araçta gerçek zamanlı nesne tespiti yapma gerekliliğinden kullanılacak algoritmada çalışma hızı ön planda olmalıdır. Bu nedenle YOLO algoritması hem çalışma hızı hem de yazılan kodların kolaylığı açısından çok büyük avantajlar sağladığı için bu algoritma otonom aracımızda nesne tanıma prosedüründe kullanmak üzere seçilmiştir.

8.2. Tasarım ve Donanımın Sonucu

Otonom aracının tasarım aşamasında hazır olarak araç gövdesi kullanılmamıştı bunun yerine tasarım bizler tarafından geliştirilerek 3B yazıcıdan elde edilmiştir. Bunun bazı dezavantajları olmaktadır. Örnek vermek gerekirse tekerleklerin monte edildiği parçaların gövdeye sabitlendiği için tekerleklerin dönüş hareketleri istenildiği gibi yapılmadığı için kameradan gelen levha verisini işlemciler doğru ve zamanlı bir şekilde işlemesine rağmen araç bunlara doğru ve tam tepki verememiştir. En nihayetinde Araç parkurda herhangi bir engel gördüğü zaman ve siyah şeridin olduğu yerlerde ileri komutunu sorunsuz bir şekilde yerine getirmiştir. Araç aynı şekilde işlemciden gelen ileri komutunu doğru bir şekilde yerine getirmektedir.

8.3. Test Aşamasının Sonucu

Kullanılan Rasbperry pi kartının çok gelişmiş olmamasından kaynaklı alınan levha verisinin anlık Görüntü İşleme yaparken yetersiz kaldığı görülmüştür. Bu nedenle Rasbperry pi 'dan Arduino'ya serial portundan gelen levha verisi çok geç geldiğinden dolayı araç istenilen şekilde harekete geçememiştir.

KAYNAKLAR

[1] Aktaş A., (2020), "DERİN ÖĞRENME YÖNTEMLERİ İLE GÖRÜNTÜ İŞLEME UYGULAMALARI", Yüksek Lisans Tezi, Marmara Üniversitesi.

[2] Web1,(2021),Goruntu_Isleme_Ders_Notlari-1.Hafta.pdf(ibrahimcayiroglu.com), (Erişim Tarihi: 24/12/2021).

[3] Web 2, (2021), Raspberry Pi Nedir ? - diyot.net nedir ?, (Erişim Tarihi: 24/12/2021).

[4] Web 3, (2021), Arduino / Genuino Uno Özellikleri (robotiksistem.com), (Erişim Tarihi: 24/12/2021).

[5] Web 4, (2021), Sail Teknoloji, (Erişim Tarihi: 25/12/2021).

[6] Web 5, (2021), Arduino ile L298N Motor Sürücü Kullanımı - Direnc.net Blog, (Erişim Tarihi: 25/12/2021).

[7] Web 6, (2022), Çizgi Takip Sensörü Modülü - KY-033 TCRT5000, (Erişim Tarihi: 25/05/2022)

[8] Web7, (2021), CARLA-Simulator, (Erişim Tarihi: 27/12/2021).

[9] Web8, (2021), Canny Edge Algılama ,(Erişim Tarihi: 29/12/2021).

[10] Web9, (2021), Python ve OpenCV: Şerit Tespiti ,(Erişim Tarihi: 29/12/2021)

[11] Web10, (2022), Tinkercad, (Erişim Tarihi: 29/12/2021)

[12] Web11, (2021), Nesne algılama ve Yüz tanıma algoritmaları (ichi.pro) , (Erişim Tarihi: 26/12/2021).

[13] Web12, (2021), RCNN, YOLO-OpenCV ile Nesne Tanıma, (Erişim Tarihi: 27/12/2021).

[14] Web13, (2021), YOLO Nedir ? (gorselanaliz.com) ,(Erişim Tarihi: 28/12/2021)

ÖZGEÇMİŞ

Berkay TÜRK, 08.02.1999 da İstanbul'da doğdu. İlk, orta ve lise eğitimini İstanbul'da tamamladı. 2017 yılında Hayrullah Kefoğlu Anadolu Lisesinden mezun oldu. 2017 yılında girdiği Gebze Teknik Üniversitesi Elektronik Mühendisliği bölümünde son sınıf öğrencisi. Pavotek şirketin de 3 ay zorunlu stajını hem yazılım hem de donanım alanında olmak üzere yaptı. İlgi alanları arasında görüntü işleme, 3D modelleme, yapay sinir ağları, programlama yer almaktadır.

Mehmet ADANIR, 15.09.1998 da Mardin'de doğdu. İlk ve orta eğitimini Mardin'de tamamladı. 2017 yılında Gebze Süleyman Demirel Anadolu Lisesinden mezun oldu. 2017 yılında girdiği Gebze Teknik Üniversitesi Elektronik Mühendisliği bölümünde son sınıf öğrencisi. Şu an Makersan şirketinde zorunlu stajını hem yazılım hem de donanım alanında olmak üzere yapmaktadır. Bir yıldır Teknofest İHA yarışmasında Bükrek İHA takımında görev almaktadır. İlgi alanları arasında otonom araç, gömülü sistemler yer almaktadır.

Burak ÖZŞAHİN, 25.08.1999 da İstanbul'da doğdu. İlk ve orta, lise eğitimini İstanbul'da tamamladı. 2017 yılında Özel Ataşehir Fen Temel Lisesinden mezun oldu. 2017 yılında girdiği Gebze Teknik Üniversitesi Elektronik Mühendisliği bölümünde son sınıf öğrencisi. Karayolları Genel Müdürlüğü şirketinde 1 ay zorunlu stajını yazılım alanında olmak üzere yaptı. Bir yıldır Teknofest İHA yarışmasında Bükrek İHA takımında başkan olarak görev almaktadır. İlgi alanları arasında sinyal işleme, gömülü sistemler, güç sistemleri, görüntü işleme yer almaktadır.

Ek 1: Arduino Kodları

```
// This uses Serial Monitor to display Range Finder distance readings
// Include NewPing Library
#include "NewPing.h"
// Hook up HC-SR04 with Trig to Arduino Pin 9, Echo to Arduino pin 10
#define TRIGGER_PIN 9 //Ultrasonik sensörün trig pini
Arduino'nun 9.pinine tanımlandı.
#define ECHO_PIN 10  //Ultrasonik sensörün echo pini Arduino'nun 10.pinine
// Maximum distance we want to ping for (in centimeters).
#define MAX_DISTANCE 400
// NewPing setup of pins and maximum distance.
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

float duration=0, distance=0;

const int MotorLE = 11;
const int MotorRE = 3;
const int MotorR1 = 7;
const int MotorR2 = 2;
const int MotorL1 = 12;
const int MotorL2 = 13;

//Sensor Connection
 const int left_sensor_pin =4;//sol sensör 0
 const int right_sensor_pin =8;//sağ sensör 1
// Sensor State
 int left_sensor_state;
 int right_sensor_state;
```

```
void setup()
{
pinMode(MotorLE, OUTPUT); //Enable_A
pinMode(MotorRE, OUTPUT); //Enable_B
pinMode(MotorR1, OUTPUT); //inputB1
pinMode(MotorR2, OUTPUT); //inputB2
pinMode(MotorL1, OUTPUT); //inputA1
pinMode(MotorL2, OUTPUT); //inputA2
pinMode(TRIGGER_PIN, OUTPUT); //trıgger
pinMode(ECHO_PIN, INPUT); //echo

pinMode(left_sensor_pin,  INPUT); // initialize Left sensor as an input
pinMode(right_sensor_pin, INPUT); // initialize Right sensor as an input

pinMode(13,OUTPUT);
Serial.begin(9600);
}
void loop()
{
 // distance = sonar.ping_cm();
  duration = sonar.ping();
  distance = (duration / 2) * 0.0343;
  // Determine distance from duration
  // Use 343 metres per second as speed of sound
  distance= duration*0.034/2;
  if (distance >= 400 || distance <= 2)
  {
    Serial.println("Out of range");
  }
```

```
else{
   while(distance==0 ||distance>30){
  distance = sonar.ping_cm();//mesafe cm
   String data = Serial.readStringUntil('\n');
   Serial.print("You sent me: ");
   Serial.println(data);
   if(data=="Sag"){
      sag();
      delay(350);
   }
   else if(data=="Sol"){
      sol();
      delay(350);
      break;
   }
   else if(data=="ileri"){
      ileri();
      delay(150);
   }
   else if(data=="Geri"){
      geri();
      delay(150);
   }

else{
 left_sensor_state = digitalRead(left_sensor_pin);
 right_sensor_state = digitalRead(right_sensor_pin);
```

```
//Serial.println(right_sensor_state);
//beyazda sıfır siyahta 1 olur
 if(right_sensor_state==1 && left_sensor_state==1) {
  distance = sonar.ping_cm();//mesafe cm
//  if(distance<=20){
//   dur();
//   delay(150);
//}
  ileri(); //FORWARD
}
 else if(right_sensor_state==0 && left_sensor_state==1) {
  sag(); //Move Right

 }

 else if(right_sensor_state==1 && left_sensor_state==0) {
  sol(); //Move Left
}

 else if(right_sensor_state==0 && left_sensor_state==0) {
  dur(); //STOP
 }
 Serial.println(distance);
 distance = sonar.ping_cm();//mesafe cm
}
distance = sonar.ping_cm();//mesafe cm
}

  dur();  //  dur
  delay(150);
Serial.println(distance);
}
```

```
void geri(){  // Robotun ileri yönde hareketi için fonksiyon tanımlıyoruz.
  digitalWrite(MotorR1, HIGH); // Sağ motorun ileri hareketi aktif
  digitalWrite(MotorR2, LOW); // Sağ motorun geri hareketi pasif
  analogWrite(MotorRE, 100); // Sağ motorun hızı 150
  digitalWrite(MotorL1, HIGH); // Sol motorun ileri hareketi aktif
  digitalWrite(MotorL2, LOW); // Sol motorun geri hareketi pasif
  analogWrite(MotorLE, 100); // Sol motorun hızı 150



}
void sol(){ // Robotun sağa dönme hareketi için fonksiyon tanımlıyoruz.
  digitalWrite(MotorR1, HIGH); // SOL motorun ileri hareketi aktif
  digitalWrite(MotorR2, LOW); // SOL motorun geri hareketi pasif
  analogWrite(MotorRE, 200); // SOL motorun hızı 0 (Motor GERİ)
  digitalWrite(MotorL1, LOW); // SAĞ motorun ileri hareketi aktif
  digitalWrite(MotorL2, HIGH); // SAĞ motorun geri hareketi pasif
  analogWrite(MotorLE, 250); // SAĞ motorun hızı 150

}
void sag(){ // Robotun sola dönme hareketi için fonksiyon tanımlıyoruz.
  digitalWrite(MotorR1, LOW); // SOL motorun ileri hareketi aktif
  digitalWrite(MotorR2, HIGH); // SOL motorun geri hareketi pasif
  analogWrite(MotorRE, 250); // SOL motorun hızı 150
  digitalWrite(MotorL1, HIGH); // SAĞ motorun ileri hareketi aktif
  digitalWrite(MotorL2, LOW); // SAĞ motorun geri hareketi pasif
  analogWrite(MotorLE, 200); // SAĞ motorun hızı 0 (Motor GERİ)

}
```

```
void ileri(){ // Robotun geri yönde hareketi için fonksiyon tanımlıyoruz.
  distance = sonar.ping_cm();//mesafe cm
 //   if(distance<=35){
  // dur();
  // delay(150);

// }
  digitalWrite(MotorR1, LOW); // Sağ motorun ileri hareketi pasif
  digitalWrite(MotorR2, HIGH); // Sağ motorun geri hareketi aktif
  analogWrite(MotorRE, 75); // Sağ motorun hızı 150
  digitalWrite(MotorL1, LOW); // Sol motorun ileri hareketi pasif
  digitalWrite(MotorL2, HIGH); // Sol motorun geri hareketi aktif
  analogWrite(MotorLE, 75); // Sol motorun hızı 150

}


void dur(){ // Robotun sola dönme hareketi için fonksiyon tanımlıyoruz.
  digitalWrite(MotorR1, HIGH); // Sağ motorun ileri hareketi aktif
  digitalWrite(MotorR2, HIGH); // Sağ motorun geri hareketi pasif
  analogWrite(MotorLE, 0); // Sol motorun hızı 150
  digitalWrite(MotorL1, HIGH); // Sol motorun ileri hareketi aktif
  digitalWrite(MotorL2, HIGH); // Sol motorun geri hareketi pasif
  analogWrite(MotorRE, 0); // Sağ motorun hızı 0 (Motor duruyor)
}
```

Ek 2: Python Kodları

```
import cv2
import numpy as np
import time
from imutils.perspective import four_point_transform
from imutils import contours
import imutils
import serial

camera = cv2.VideoCapture(-1)

def findTrafficSign():
    '''
    This function find blobs with blue color on the image.
    After blobs were found it detects the largest square blob, that must be the sign.
    '''
    # define range HSV for blue color of the traffic sign
    lower_blue = np.array([85,100,70])
    upper_blue = np.array([115,255,255])

    #arduino to rasqberry serial com
    ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)
    ser.reset_input_buffer()

    while True:
        # grab the current frame
        (grabbed, frame) = camera.read()
        if grabbed:
            assert not isinstance(frame,type(None)), 'frame not found'
        if not grabbed:
            print("No input image")
            break
```

```
frame = imutils.resize(frame, width=300)
      frameArea = frame.shape[0]*frame.shape[1]

      # convert color image to HSV color scheme
      hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

      # define kernel for smoothing
      kernel = np.ones((3,3),np.uint8)
      # extract binary image with active blue regions
      mask = cv2.inRange(hsv, lower_blue, upper_blue)
      # morphological operations
      mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
      mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

      # find contours in the mask
      cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]

      # defite string variable to hold detected sign description
      detectedTrafficSign = None

      # define variables to hold values during loop
      largestArea = 0
      largestRect = None

      # only proceed if at least one contour was found
      if len(cnts) > 0:
         for cnt in cnts:
            # Rotated Rectangle. Here, bounding rectangle is drawn with minimum
area,
            # so it considers the rotation also. The function used is cv2.minAreaRect().
            # It returns a Box2D structure which contains following detals -
            # ( center (x,y), (width, height), angle of rotation ).
            # But to draw this rectangle, we need 4 corners of the rectangle.
            # It is obtained by the function cv2.boxPoints()
```

```
            rect = cv2.minAreaRect(cnt)
            box = cv2.boxPoints(rect)
            box = np.int0(box)

            # count euclidian distance for each side of the rectangle
            sideOne = np.linalg.norm(box[0]-box[1])
            sideTwo = np.linalg.norm(box[0]-box[3])
            # count area of the rectangle
            area = sideOne*sideTwo
            # find the largest rectangle within all contours
            if area > largestArea:
               largestArea = area
               largestRect = box
      # draw contour of the found rectangle on  the original image
      if largestArea > frameArea*0.02:
         cv2.drawContours(frame,[largestRect],0,(0,0,255),2)
      #if largestRect is not None:
         # cut and warp interesting area
         warped = four_point_transform(mask, [largestRect][0])

         # show an image if rectangle was found
#           cv2.imshow("Warped", cv2.bitwise_not(warped))

         # use function to detect the sign on the found rectangle
         detectedTrafficSign = identifyTrafficSign(warped)
         print(detectedTrafficSign)
         if detectedTrafficSign == "Turn Right":
            print("Sag")
            ser.write(b"Sag\n")
            line = ser.readline().decode('utf-8').rstrip()
            time.sleep(5)
         if detectedTrafficSign == "Turn Left":
            print("Sol")
            ser.write(b"Sol\n")
            line = ser.readline().decode('utf-8').rstrip()
            time.sleep(5)
```

```
            if detectedTrafficSign == "Move Straight":
                print("Ileri")
                ser.write(b"ileri\n")
                line = ser.readline().decode('utf-8').rstrip()
                time.sleep(5)
            if detectedTrafficSign == "Turn Back":
                print("Geri")
                ser.write(b"Geri\n")
                line = ser.readline().decode('utf-8').rstrip()
                time.sleep(5)
            # write the description of the sign on the original image
            cv2.putText(frame, detectedTrafficSign, tuple(largestRect[0]),
cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)

        # show original image
        cv2.imshow("Original", frame)
        # if the `q` key was pressed, break from the loop
        if cv2.waitKey(1) & 0xFF is ord('q'):
            cv2.destroyAllWindows()
            print("Stop programm and close all windows")
            break
def identifyTrafficSign(image):
    '''
    In this function we select some ROI in which we expect to have the sign parts. If the
ROI has more active pixels than threshold we mark it as 1, else 0
    After path through all four regions, we compare the tuple of ones and zeros with
keys in dictionary SIGNS_LOOKUP
    '''
    # define the dictionary of signs segments so we can identify
    # each signs on the image
    SIGNS_LOOKUP = {
        (1, 0, 0, 1): 'Turn Right', # turnRight
        (0, 0, 1, 1): 'Turn Left', # turnLeft
        (0, 1, 0, 1): 'Move Straight', # moveStraight
        (1, 0, 1, 1): 'Turn Back', # turnBack
    }
```

```
THRESHOLD = 150


   image = cv2.bitwise_not(image)
   # (roiH, roiW) = roi.shape
   #subHeight = thresh.shape[0]/10
   #subWidth = thresh.shape[1]/10
   (subHeight, subWidth) = np.divide(image.shape, 10)
   subHeight = int(subHeight)
   subWidth = int(subWidth)

   # mark the ROIs borders on the image
   cv2.rectangle(image, (subWidth, 4*subHeight), (3*subWidth, 9*subHeight),
(0,255,0),2) # left block
   cv2.rectangle(image, (4*subWidth, 4*subHeight), (6*subWidth, 9*subHeight),
(0,255,0),2) # center block
   cv2.rectangle(image, (7*subWidth, 4*subHeight), (9*subWidth, 9*subHeight),
(0,255,0),2) # right block
   cv2.rectangle(image, (3*subWidth, 2*subHeight), (7*subWidth, 4*subHeight),
(0,255,0),2) # top block

   # substract 4 ROI of the sign thresh image
   leftBlock = image[4*subHeight:9*subHeight, subWidth:3*subWidth]
   centerBlock = image[4*subHeight:9*subHeight, 4*subWidth:6*subWidth]
   rightBlock = image[4*subHeight:9*subHeight, 7*subWidth:9*subWidth]
   topBlock = image[2*subHeight:4*subHeight, 3*subWidth:7*subWidth]
```

```
# we now track the fraction of each ROI
leftFraction = np.sum(leftBlock)/(leftBlock.shape[0]*leftBlock.shape[1])
centerFraction = np.sum(centerBlock)/(centerBlock.shape[0]*centerBlock.shape[1])
rightFraction = np.sum(rightBlock)/(rightBlock.shape[0]*rightBlock.shape[1])
topFraction = np.sum(topBlock)/(topBlock.shape[0]*topBlock.shape[1])

segments = (leftFraction, centerFraction, rightFraction, topFraction)
segments = tuple(1 if segment > THRESHOLD else 0 for segment in segments)

#cv2.imshow("Warped", image)

if segments in SIGNS_LOOKUP:
return SIGNS_LOOKUP[segments]
else:
return None



def main():
findTrafficSign()



if __name__ == '__main__':
main()
```