
GAZİ ÜNİVERSİTESİ
Gazi
1926

OTONOM ROBOTLARDA HARİTALANDIRMA VE ÇİZGİ TAKİBİ

Kaan KAVAL

LİSANS TEZİ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ

GAZİ ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

OCAK 2021

Kaan KAVAL tarafından hazırlanan “OTONOM ROBOTLARDA HARİTALANDIRMA VE ÇİZGİ TAKİBİ” adlı tez çalışması aşağıdaki jüri tarafından OY BİRLİĞİ ile Gazi Üniversitesi Elektrik Elektronik Bölümümde LİSANS TEZİ olarak kabul edilmiştir.

Danışman: Doç. Dr. Ali SAYGIN

Elektrik Elektronik Mühendisliği, Gazi Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

...……...

Başkan: Prof. Dr. Çetin ELMAS

Elektrik Elektronik Mühendisliği, Gazi Üniverisitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

……..

Üye: Prof. Dr. Mahir DURSUN

Elektrik Elektronik Mühendisliği, Gazi Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

……..

Tez Savunma Tarihi: 29/01/2021

Jüri tarafından kabul edilen bu tezin Lisans Tezi olması için gerekli şartları yerine getirdiğini onaylıyorum.

…….…….

Prof. Dr. Nihat ÖZTÜRK

Elektrik Elektronik Mühendisliği Bölüm Başkanı

ETİK BEYAN

Gazi Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu,

bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

Kaan KAVAL

29/01/2021

OTONOM ROBOTLARDA HARİTALANDIRMA VE ÇİZGİ TAKİBİ

(Lisans Tezi)

Kaan KAVAL

GAZİ ÜNİVERSİTESİ

Elektrik Elektronik Mühendisliği

Ocak 2021

ÖZET

Günümüzde otonom robotlar bilim dünyasının ilgi alanındadır. Otonom robotlar karmaşık birçok algoritma içermektedir. Bu algoritmaları anlayabilmek ve tasarlayabilmek için her bir algoritmayı araştırmak gerekmektedir. Bu çalışmada hem üzerinde çalışması hemde tasarlaması zor olan algoritmaları Robot Operating System (ROS) üzerinde nasıl gerçekleştirileceğinden bahsedilmiştir. Sistemin simülasyonu Jetson NANO üzerinden gerçekleştirilmiştir. Jetson NANO içerisine Robot Operating System (ROS) kurularak algoritmalar tasarlanmıştır. Robot Operating System (ROS) başlıca robota veri göndermekte ve almakta görev almaktadır. ROS bu verileri Gazebo ortamından almakta ve OpenCV kütüphanesi ile işlemektedir. Rviz ortamında görselleştirilen veriler otonom hareket ve haritalandırma işlemlerinde geliştiriciye görselleştirme yapmaktadır. Çizgi takibi tarafında ise görüntüler kendi çerçevelerinde görselleştirilmektedir. Tasarımı yapılan algoritması sayesinde otonom bir robotun haritalandırma verileri görselleştirilip, çizgi takibi algoritması görselleştirilmiştir. Aynı zamanda geri beslemeli bir kontrol uygulanmaktadır. Uygulanan geri beslemeli kontrol P kontrol olup sistemin daha stabil çalışması sağlanmıştır. Görüntü işlemede HSV renk uzayı kullanılması sayesinde renk fark etmeksizin istenilen çizgi takip edilebilmektedir. Yapılan uygulamada çalışmaların açıklanması ve görselleştirilmesi sayesinde bir eğitim materyali olarak kullanılabilecektir.

Anahtar Kelimeler	:	Jetson NANO, Robot Operating System (ROS), Haritalandırma, Çizgi takibi, Otonom hareket, Lidar verilerini anlamlandırma
Sayfa Adedi	:	36
Danışman	:	Doç. Dr. Ali SAYGIN

MAPPING AND LINE TRACKING ON AUTONOMOUS ROBOTS

(License Thesis)

Kaan KAVAL

GAZİ UNIVERSITY

Electrical Electronical Engineering

January 2021

ABSTRACT

Today, autonomous robots are in the area of interest of the scientific world. Autonomous robots contain many complex algorithms. To understand and design these algorithms, it is necessary to investigate each algorithm. In this study, it is mentioned how to implement algorithms that are difficult to both work on and design on Robot Operating System (ROS). The simulation of the system was carried out on Jetson NANO. Algorithms were designed by installing Robot Operating System (ROS) in Jetson NANO. The Robot Operating System (ROS) is mainly involved in sending and receiving data to the robot. ROS takes this data from Gazebo environment and processes it with OpenCV library. The data visualized in the Rviz environment make visualization to the developer in autonomous movement and mapping processes. On the line tracking side, images are visualized in their own frames. Thanks to the algorithm designed, the mapping data of an autonomous robot and the line tracking algorithm was visualized. At the same time, a feedback control is applied. The applied feedback control is P control and the system has been provided to work more stable. Thanks to the use of HSV color space in image processing, the desired line can be followed regardless of color. It can be used as an educational material thanks to the explanation and visualization of the studies in the application.

Key Words	:	Jetson NANO, Robot Operating System (ROS), Mapping, Line tracking, Autonomous motion, Interpretation lidar data
Page Number	:	36
Supervisor	:	Assoc. Prof. Dr. Ali SAYGIN

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren saygıdeğer hocam Doç. Dr. Ali SAYGIN'a, tez hazırlama ve uygulama sürecinde desteğini esirgemeyen M. Fatih ÖZKAN'a, kıymetli tecrübelerinden faydalandığım değerli arkadaşlarım ve manevi destekleriyle beni hiçbir zaman yalnız bırakmayan aileme teşekkürü bir borç bilirim.

İÇİNDEKİLER

RESİMLER TABLOSU

ŞEKİLLER TABLOSU

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış bazı simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
$\bar{x}$	Merkez noktasının x koordinatındaki değeri
$\bar{y}$	Merkez noktasının y koordinatındaki değeri

Kısaltmalar	**Açıklama**
ROS	Robot Operating System (Robot İşletim Sistemi)
IoT	Internet of Things (Nesnelerin İnterneti)
SBC	Single Board Computer (Tek kart bilgisayar)
CPU	Central Processing Unit (Merkezi İşlem Birimi)
GPU	Graphical Processing Unit (Grafik İşlem Birimi)
BSD	Berkeley Software Distrubution
ODE	Open Dynamic Engine
IR	InfraRed(Kızılötesi)
HSV	Hue, Saturation, Value
P	Propotional (Oransal)
GPS	Global Position System (Küresel Konum Sistemi)

1 GİRİŞ

Günümüzde teknolojinin gelişmesi ile araçların otonom çalışmalar teknolojinin merkezine yerleşti. Otonom robotlar insan hayatını kolaylaştırmak, insan hatalarını azaltmak, üretim hızını artırmak gibi nedenlerden dolayı geliştirilmektedir. 2023 yılına kadar bütün otomobillerin elektrikli üretileceği ön görülmektedir. Sanayi 4.0 ile IoT (Internet of Things) olabildiğince hayatımıza girmeye devam etmektedir.

Robot tanımı çok eski olmasına rağmen günümüzde tanımladığımız robotların ilk temeli 1940'larda Isaac ASIMOV 'i-Robot' adlı kitabında robotiğin 3 yasasını ortaya atmıştır. 'Robotiğin Üç Yasası' adlı bu yasalar şöyledir;

1- Robotlar, insanlara zarar veremez ya da eylemsiz kalarak onlara zarar gelmesine göz yumamaz.
2- Robotlar, birinci yasayla çakışmadığı sürece insanlar tarafından verilen emirlere itaat etmek zorundadır.
3- Robotlar, birinci ve ikinci yasayla çakışmadığı sürece kendilerine zarar vermez.

Daha sonra robotların yapay zekalarından dolayı insanlığa zarar vermemesi adına sıfırıncı yasayı eklemiştir.

0- Robotlar, tüm insanlığa zarar veremez ya da eylemsiz kalarak insanlığın bütününe zarar gelmesine göz yumamaz.

Otonomi bağımsız karar verme, kendi kendini yönetebilme yetkisi olarak tanımlanmaktadır. Otonomi robotlarda ilk olarak milattan önce 240'lı yıllara kadar gitse de günümüzdeki sanayileşmiş gerçek anlamdaki ilk robot 1942 yılında otonom olarak çalışan boya makinesidir. Daha sonra yapılan yapay zekâ çalışmaları sayesinde otonom robotlarda büyük bir adım atılarak robot üretim şirketleri, yapay zekâ çalışmaları başladı. Günümüzde otonom hareketlerde göz önünde savunma sanayi, üretim tesisleri ve otonom otomobiller bulunmaktadır. Bu çalışmalar insan hatalarını azaltmak, yaşam kalitesini artırmak gibi birçok amaç üzerine üretilmekte ve kullanılmaktadır [1]

Otonom robotlar, günümüzde robot algoritmalarını IoT ile birleştirmektedir. Günümüzdeki robotlar gelişmiş kontrol sistemlerini kullanan, robotun durum verilerini ve çevresel verileri analiz ederek bağımsız karar verebilme yeteneğine sahip olan akıllı sistemlerdir.

Görüntü işleme, yapay zekanın en çok kullanılan alanlarındandır. Yapay zekaya olan ilgi sosyal medya, günlük teknolojiler, askeri teknolojiler ve endüstriyel teknolojilerin içerisinde sürekli olarak kullanılmasından dolayı toplumun ilgisini çekmektedir.

Bu çalışmada otonom bir robotun haritalandırma için lidar verileri kullanma ve çizgi izlemek için kamera verileri ile görüntü işleme tekniklerinin birlikte kullanılması amaçlanmıştır. Tez çalışması için literatür taraması yapılarak güncel yöntemlerin kullanılması amaçlanmıştır.

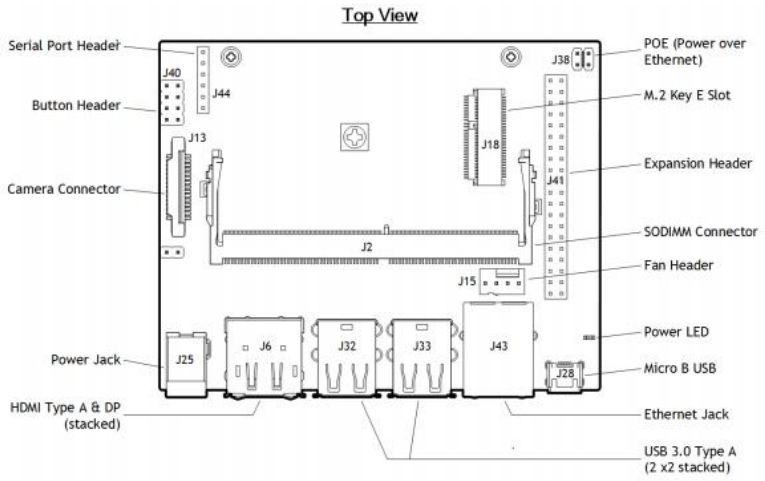
2 JETSON NANO

Nvidia şirketinin üretmiş olduğu Jetson NANO Dev Kit tek kart bilgisayar (SBC) olarak tanımlanmaktadır.

2.1 Jetson Nano Donanımı

Jetson NANO Dev Kit aslında Resim 2.1'de gösterilen Jetson Nano'nun kendisi ve Şekil 2.1'de gösterilen üzerinde bağlantı noktaları olan prototipleşmeyi kolaylaştıran kısım olmak üzere iki parçadan oluşmaktadır. Kredi kartından daha küçük boyutları olan bu kartın 5w ile bütün işlemleri kolaylıkla yapabilmektedir [2].

Resim 2.1 Jetson NANO Dev Kit



Şekil 2.1 Jetson NANO soket yapısı

2.2. Teknik Özellikleri

Jetson NANO 128 çekirdekli Maxwell GPU, dört çekirdekli ARM A57 64-bit CPU ve 4GB LPDDR4 belleğe sahiptir. Detaylı özellikleri Şekil2.2'de verilmiştir.

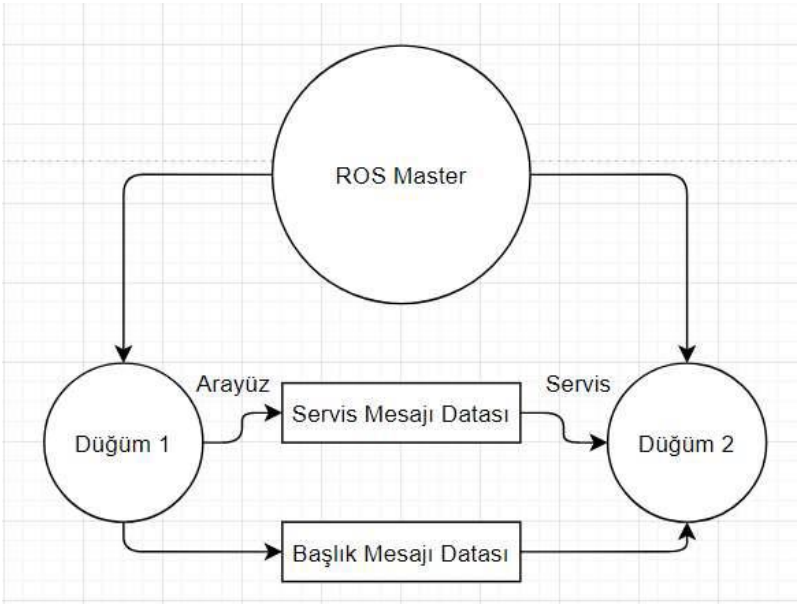
Modülün Teknik Özellikleri	
GPU	128 NVIDIA CUDA® çekirdekli, 0,5 TFLOP (FP16) NVIDIA Maxwell™ mimarisi
CPU	Dört çekirdekli ARM® Cortex®-A57 MPCore işlemci
Bellek	4 GB 64 bit LPDDR4 1600 MHz - 25,6 GB/sn
Depolama	16 GB eMMC 5.1 Flash
Video Enkodlama	4Kp30 | 4x 1080p30 | 9x 720p30 (H.264/H.265)
Video Dekodlama	4Kp60 | 2x 4Kp30 | 8x 1080p30 | 18x 720p30 (H.264/H.265)
Kamera	12 şerit (3x4 veya 4x2) MIPI CSI-2 D-PHY 1.1 (18 Gb/sn)
Bağlantı	Wi-Fi, harici yonga gerektirir
	10/100/1000 BASE-T Ethernet
Ekran	HDMI 2.0 veya DP1.2 | eDP 1.4 | DSI (1 x2) 2 eş zamanlı
UPHY	1 x1/2/4 PCIE, 1x USB 3.0, 3x USB 2.0
G/Ç	3x UART, 2x SPI, 2x I2S, 4x I2C, GPIO
Boyut	69,6 mm x 45 mm
Mekanik	260 pinli kenar bağlayıcısı

Şekil 2.1 Jetson NANO Teknik Özellikleri

Görüntü işlemedeki gücü ve aynı anda gerçekleştirebildiği işlemlerden dolayı Jetson NANO tercih edilmiştir.

3 ROBOT OPERATING SYSTEM (ROS)

ROS; araç, kitaplık ve kurallardan oluşan robotun yazılım kısmında esnek yapı oluşturan bir platformdur. Detaylı robotlarda karmaşıklaşan yazılım tarafını kolaylaştırmaktadır. Mesaj ve paket yapısı sayesinde robotun fonksiyonlarının ayrı ayrı kodlanıp bu fonksiyonların birbiri ile mesajlaşmasını sağlar. Paketler kendi işlemlerini yapar [3].



Şekil 3.1 ROS Kalıbı

3.1 Robot Operating System Yapısı

ROS ile çalışan bir grupta kişiler kendi uzmanlık alanlarıyla ilgili bölümü kodlar. Daha sonra bu ayrık programlar robotun işletim sisteminde birleştirilip yapılandırılır. ROS bünyesinde kullanıcıların yayınladığı paketler, kütüphaneler ile kümülatif şekilde büyümekte ve eksiklerini gidermektedir.

ROS düğümler, mesajlar ve konular üzerinden haberleşen bir mimaridir. Düğümler Publish-Subscribe (Yayınlama-abone) olarak gereken mesajları yayınlarlar veya dinlerler. Aldıkları mesaj üzerine gereken çıktıyı robot üzerinde gerçekleştirirler.

3.1.1 ROS master

Düğümlerin haberleşebilmesi ve senkron olarak hareket edebilmeleri için bir sunucuya ihtiyaç vardır. Bu bağlantıyı ROS Master sağlamaktadır.

3.1.2 Düğümler

Robot üzerinde gerçekleştirilecek işlemler düğümlerin içerisinde kodlanır. Her bir düğümün ROS Master üzerinde bir adı vardır. Birçok düğüm diğer düğümlerden gelen veriler ile eylemler gerçekleştirir. Yapı düğümlerin birbirlerine veri gönderip alması olduğu için düğümler ROS yapısının ana yapılarıdır.

3.1.3 Konular

Düğümlerin verileri gönderdiği ve dinlediği başlıklardır. Konuların kendine özel adları vardır. Abone olma, yayınlama özellikleri bu konular üzerinden yapılmaktadır. Hiçbir düğüm, konuya hangi düğümün yayınlama/abone olduğunu bilemez. Düğümler sadece konuya veri gönderir veya konudan veri alır.

3.1.4 Mesajlar

Düğümler, konular üzerinden abone olduklarında veya yayın yaptıklarında gönderdiği verilere mesaj denir. Mesajlar bir değişken gibi küçük boyutlu olabilir veya işlenmiş bir görüntü gibi büyük boyutluda olabilir.

3.2 Araçlar

ROS bünyesinde arayüz programı, veri kaydı aracı, yapılandırma aracı ve kendi kabuk yapısını barındırır.

3.2.1 Rviz

Üç boyutlu görselleştirme aracıdır. Robottan alınan çevresel veriler, sensör verileri, haritalandırma verileri gibi düğümlerden alınan verileri görselleştirebilen bir araçtır.

3.2.2 Rosbag

Mesajlardaki verileri kaydedebilmemizi ve daha sonra bakabilmemizi sağlayan bir komut satırı aracıdır. Kaydedilen bu verileri kullanarak bu kodlardaki hataları gidermek büyük bir kolaylık sağlamaktadır.

3.2.3 Catkin

CMake üzerinden çalışan yapı ROS derlemesi için kullanılmaktadır. Dilden bağımsız şekilde çalışır.

3.2.3.1 cmake

Yazılım geliştirmede derleyiciden bağımsız derleme, test etme, paketleme gibi özellikleri dil fark etmeksizin derleyen açık kaynaklı bir yazılımdır [7]. Birçok kitaplığa bağlı olan kod bütünlerini ve uygulamaları destekler.

3.2.4 Rosbash

Ubuntu'nun kendi kabuğunun işlevselliğini artıran bir araçtır. Ekstra klasör işlemleri kısaltmaları ekleyerek kullanıcıların dosya yollarını teker teker girmelerine gerek kalmadan hızlıca işlemler yapabilmelerini sağlar.

3.2.5 Roslaunch

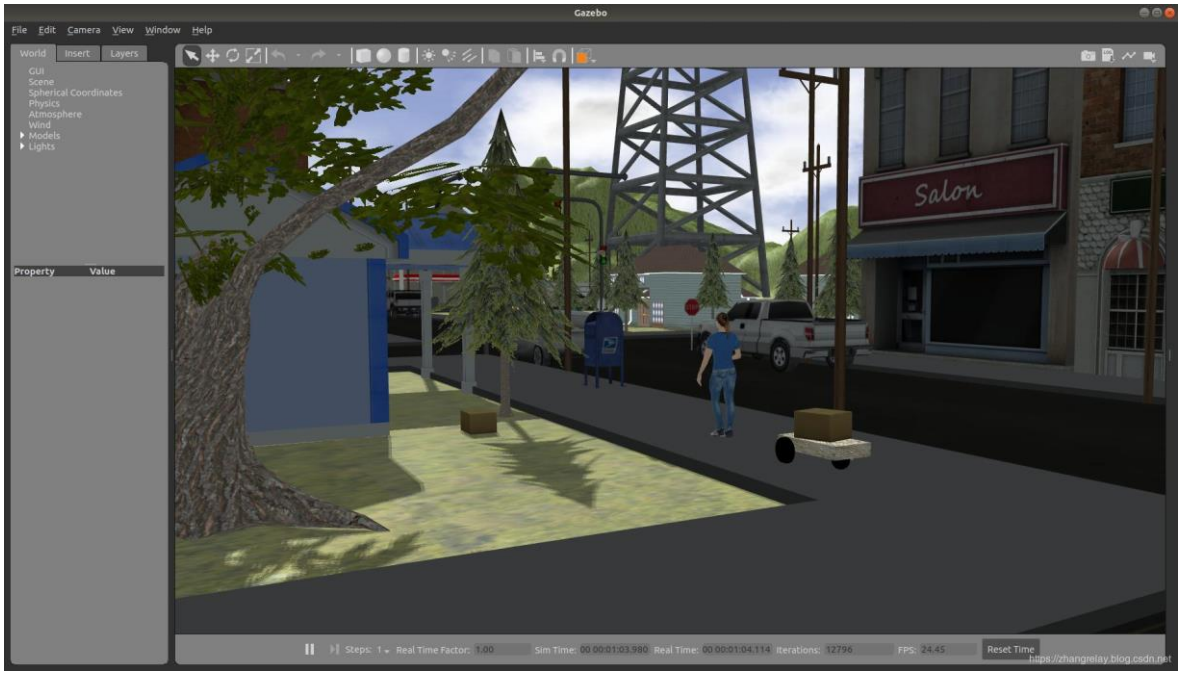
Düğümleri başlatmak için kullanılan bir komut satırı aracıdır .xml yapılandırma dosyalarını çalıştırır .xml yapılandırma dosyalarının karmaşık başlatma sürecini tek bir komutta kolayca gerçekleştirilmesini sağlar.

4 GAZEBO

Açık kaynaklı bir üç boyutlu robot simülatörüdür. Gerçekçi senaryolar sunmaktadır. Algoritmaları test etmek için veya yapay zekâ eğitmek için de kullanılabilir. Robotun testini kolaylaştırmak için ara yüzünde birçok aracı vardır.

Gazebo, 2004'ten 2011'e kadar oyuncu projesi olarak geliştirilmiştir. Gazebo'ya 2011'de ODE fizik motorunu, OpenGL'i, sensör simülasyonu için uygunluğu ve eyleyici kontrolünü sisteme entegre ederek bugünkü haline getirildi [4].

ODE, Bullet vb. gibi birden fazla yüksek performanslı fizik motorunu çalıştırabilir. Yüksel kaliteli aydınlatma, gölgeler ve dokular gibi ortamın gerçekçilik seviyesini artıran özelliklerini kullanıcıya sunmaktadır. Lidar, kamera görüntüleri gibi simülasyon ortamından görüntüleri ve sensörlerden gelen verileri rahatlıkla kullanabilir [4].



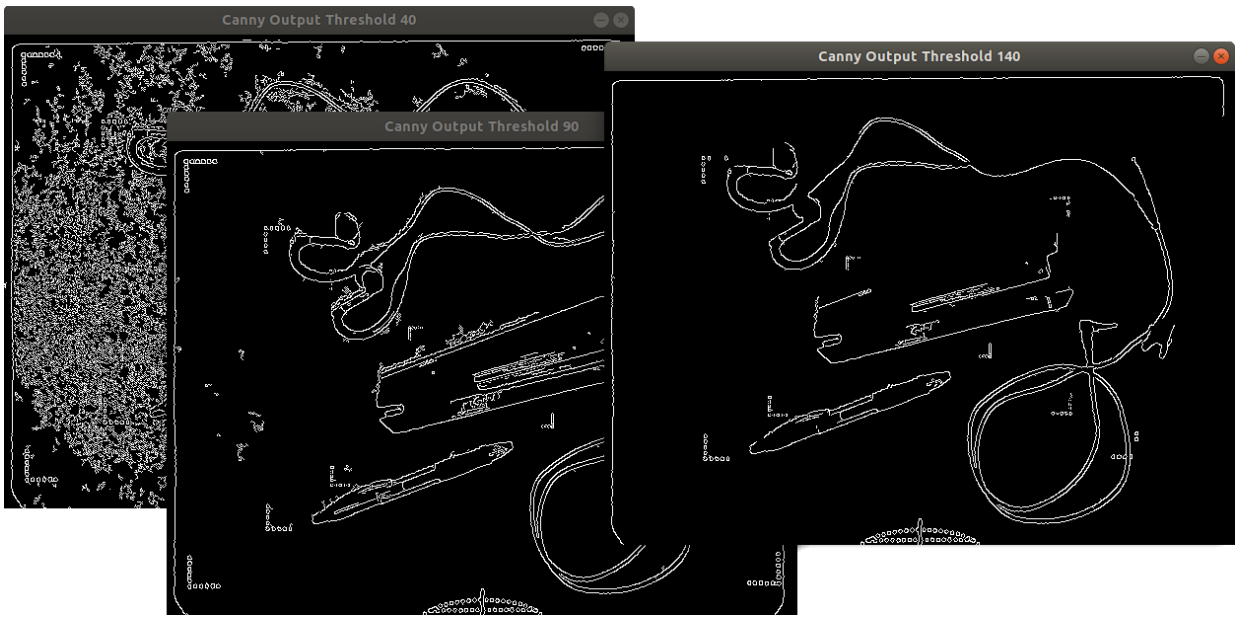
Resim 4.1 Gazebo Ortamı

5 OPENCV

OpenCV, açık kaynaklı bir bilgisayar görme ve makine öğrenimi kütüphanesidir. Bilgisayarla görme uygulamaları için ortak bir altyapı sağlamak ve ticari ürünlerde makine algısının kullanımını hızlandırmak için yapılmıştır. BSD lisanslı bir üründür [5].

Kütüphane 2500'den fazla optimize algoritmaya sahiptir. Genellikle yüzleri algılamak, nesneleri tanımlamak, insan eylemlerini sınıflandırmak, kamera görüntülerini işlemek, hareketli nesneleri takip etmek, stereo kameralar ile üç boyutlu nokta bulutları oluşturmak, veri tabanından benzer görüntüler bulmak, görüntüleri eşleştirmek ve artırılmış gerçeklik vb. uygulamalarda kullanılan algoritmaları sayesinde çok geniş bir kullanım yapısı ve kolaylığı vardır.

C++, Python, Java ve MATLAB ara yüzlerine sahip OpenCV Windows, Linux, Android, MacOS'yi destekler. CUDA için hali hazırda bir ara yüz geliştirilmektedir [6].



Resim 5.1 OpenCV Filtremele İşlemi

6 TEZ ÇALIŞMALARI

Bu çalışmada Jetson NANO üzerinde ROS ekosistemi sayesinde hazırlanmış lidar verileri ile gmapping uygulaması ve kamera üzerinden şerit takibi yapılmıştır. Simülasyon ortamında gerçekleştirilen çalışmada turtlebot3 kullanılmaktadır.

Gmapping otonom hareketlerde rota hesapları ve engellerin ön görülmesi için önemli bir rol almaktadır. Lidar sayesinde alınan veriler toplanarak rviz üzerinden biriktirilir. Yeterli verinin oluştuğuna karar verildiğinde bu veriler kaydedilir. Daha sonra kaydedilen veriler harita olarak kullanılabilir. Rviz bu haritayı görselleştirmek ve robotun gerçek zamanlı olarak hareketini izlemek için Güvenilir bir otonom hareket için lidar ve kamera verileri gerçek zamanlı olarak işlenmelidir. NANO üzerinde bulunan güçlü işlemcileri sayesinde bu işlemler gerçek zamanlı olarak kullanılabilir bir hızda işlenmektedir.

6.1 Donanım

Tez çalışmaları Jetson NANO üzerinde gerçekleştirilmiştir. Jetson NANO'ya, Intel AC8265 Wireless/Bluetooth M.2 Network Adapter ve Raspberry Pi Camera Module V2 eklenmiştir.

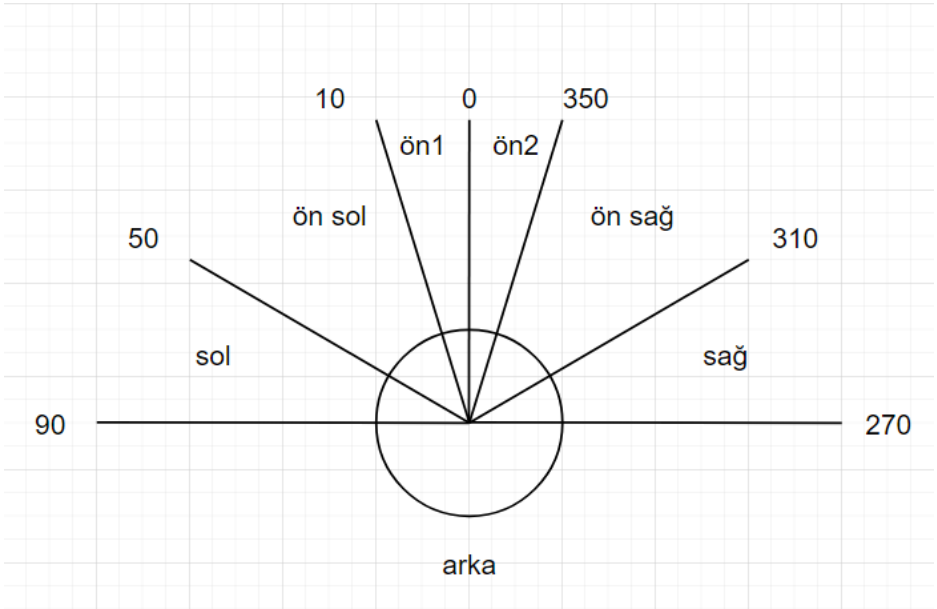
Resim 6.1 Donanım

6.2 Yazılım

Yazılım kısmında adım adım yol kat edilmiştir. İlk olarak Gazebo simülasyon ortamında Lidar verilerinin okunması ve anlamlandırılması üzerine çalışmalar yapılmıştır. Lidar verileri üzerinde bilgi sahibi olunduktan sonra, Lidar verileri kullanılarak bir otonom hareket senaryosu geliştirilmiştir. Daha sonra gmapping paketi ile otonom hareket senaryosu entegre edilerek haritalandırma yapılmıştır. Haritalandırma üzerinde eksikler giderildikten sonra çizgi izleme algoritmaları üzerinde araştırmalar yapılmıştır. Son olarak python senaryosu yazılıp tez çalışması tamamlanmıştır.

6.3 Lidar Verileri

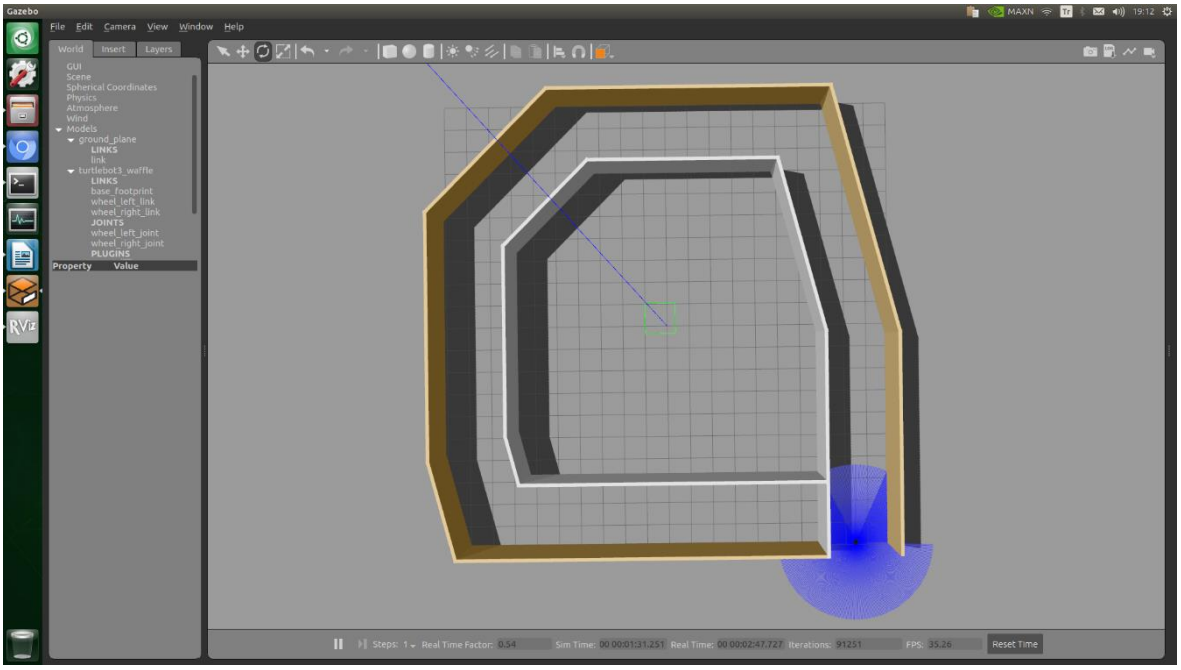
Lidar verileri bir IR Led ve IR alıcının çeşitli derecelerde döndürülmesi ve çevreden alınan uzaklık verilerinin anlamlandırılması ile çalışan bir yapıdır.
Lidar verileri LaserScan konusu üzerinden sensor_msgs.msg dosyası ile ROS üzerinden alınır. Alınan veri 'veri.ranges' adında bir diziye kaydedilir. Dizi robotun bölgelerine ayrılır. Ayrılan bölgelerdeki en düşük veri o bölgedeki nesneye olan uzaklığımız kabul edilir.



Resim 6.2 Lidar verileri

Resimdeki gibi bölgelendirilen lidar verilerinin simülasyonu ROS ortamında aşağıdaki python kodu ile gerçekleştirilmiştir.

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import rospy

from sensor_msgs.msg import LaserScan

def lidar_data(veri):
    """

    adet = len(veri.ranges)

    print 'adet:',adet


    print '1:',veri.ranges[0]
    print '90:',veri.ranges[89]
    print '180',veri.ranges[179]
    print '270',veri.ranges[269]

    """
    bolgeler = {

        'on1':  min(min(veri.ranges[0:9]), 3.5),
        'on2': min(min(veri.ranges[349:359]), 3.5),
        'on_sol':  min(min(veri.ranges[10:49]), 3.5),
        'sol':  min(min(veri.ranges[50:89]), 3.5),
        'arka':   min(min(veri.ranges[90:268]), 3.5),
        'sag':   min(min(veri.ranges[269:308]), 3.5),
        'on_sag':   min(min(veri.ranges[309:348]), 3.5),
    }

    print bolgeler


if __name__ == '__main__':

    rospy.init_node('tb3_lidar',anonymous=True)

    rospy.Subscriber('/scan', LaserScan, lidar_data)

    rospy.spin()
```



Resim 6.3 Lidar verilerinin simülasyon ortamındaki görüntüsü

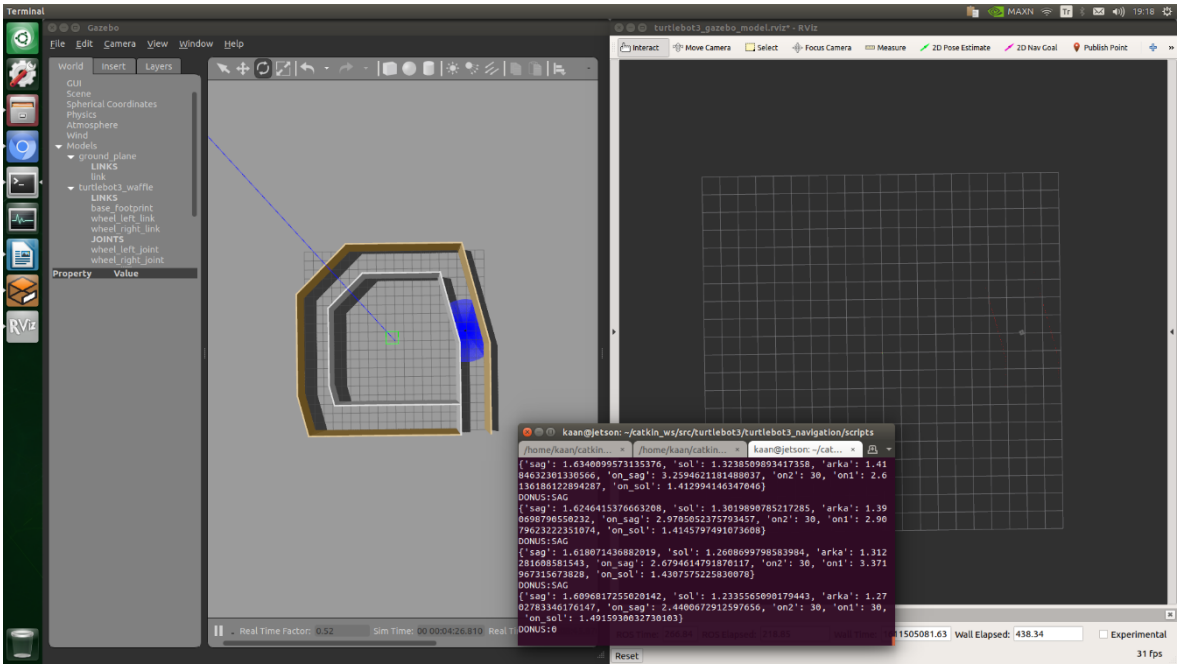
6.4 Otonom Hareket

Lidar verileri anlamlandırıldıktan sonra alınan veriler /scan konusu üzerinden alınıp işlenir. İşlenen veriye göre bir hareket çıktısı oluşturulur. Oluşturulan çıktı cmd_vel konusu üzerinden Twist mesajı olarak yayınlanır.

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

def lidar_data(veri):

    bolgeler = {

        'on1':  min(min(veri.ranges[0:9]), 30),
        'on2': min(min(veri.ranges[349:359]), 30),
        'on_sol':  min(min(veri.ranges[10:49]), 30),
        'sol':  min(min(veri.ranges[50:89]), 30),
        'arka':   min(min(veri.ranges[90:268]), 30),
        'sag':   min(min(veri.ranges[269:308]), 30),
        'on_sag':   min(min(veri.ranges[309:348]), 30),

    }

    print bolgeler
    hareket(bolgeler)
```

```
def hareket(bolgeler):

    if bolgeler['on1'] and bolgeler['on2'] > 0.8:

        hiz = 0.8

        if bolgeler['on_sag'] - bolgeler['on_sol'] > 1.0:

            print ('DONUS:SAG')
            hiz = 0.4
            donus = -0.4

        elif bolgeler['on_sol'] - bolgeler['on_sag'] > 1.0:

            print ('DONUS:SOL')
            hiz = 0.4
            donus = 0.4

        else:
            print('DONUS:0')
            hiz = 0.8
            donus = 0.0

    else:

        print('DUR')
        hiz = 0.0
        donus = 0.0

    obje.linear.x = hiz
    obje.angular.z = donus
    pub.publish(obje)

def durdur():

    rospy.loginfo("robot durduruldu!")
    pub.publish(Twist())

if __name__ == '__main__':

    rospy.init_node('tb3_otonom',anonymous=True)

    rospy.Subscriber('/scan', LaserScan, lidar_data)

    rospy.loginfo("Sonlandirmak icin: CTRL + C")
    rospy.on_shutdown(durdur)

    pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    obje = Twist()

    rospy.spin()
```

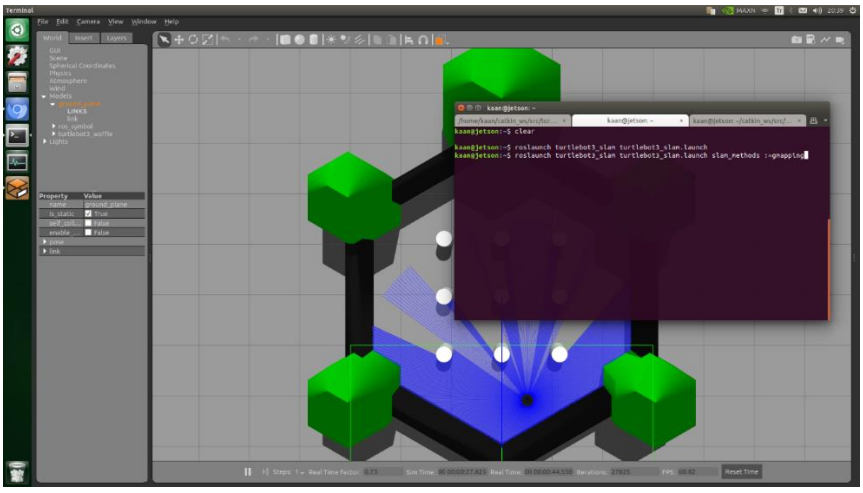
Otonom hareket içinde anlamlı lidar verileri için kullanılan algoritmanın üzerine hareket algoritması eklenmiştir. LaserScan konusu üzerinden sensor_msgs.msg mesajı alınır. Twist konusu üzerinden geometry_msgs.msg mesajı gönderilir. Twist üzerinden x koordinatındaki hız verisi, z koordinatındaki açısal dönüş verisi yayınlanır. Durdurmak için boş bir Twist mesajı gönderilmelidir aksi takdirde robot en son yayınlanan Twist mesajına göre hareketine devam etmektedir.



Resim 6.4 Gazebo ortamında otonom hareket

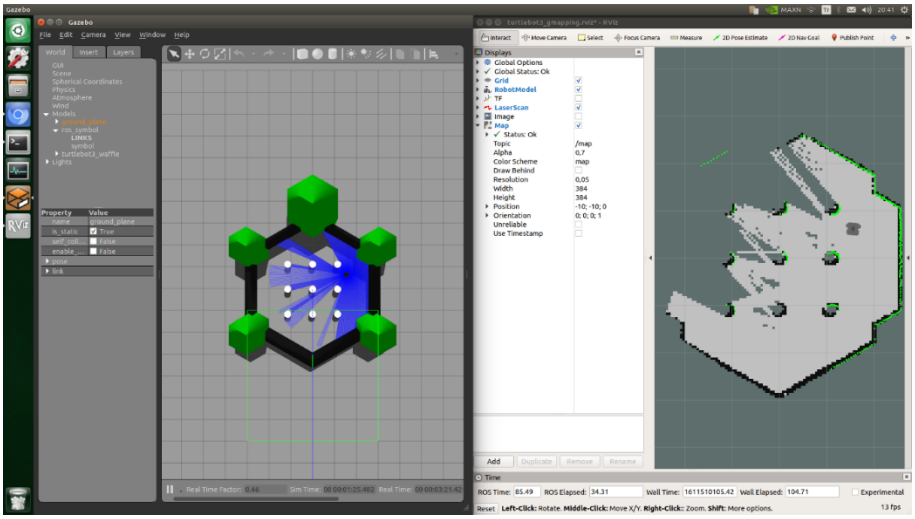
6.5 Haritalandırma

Tez çalışmasında haritalandırma ROS bünyesinde bulunan gmapping paketi üzerinden gerçekleştirilmiştir. Gmapping paketi Simultaneous Localization and Mapping (SLAM) tekniğini kullanmaktadır. Lidar verileri ve pozisyon bilgisi sayesinde haritalandırma yapabilmektedir. Kullanılan turtlebot3 isimli robotu Gazebo ortamında bir ortama yerleştirilip ortamda engelden kaçması için hazır olan bir algoritma çalıştırılmıştır.



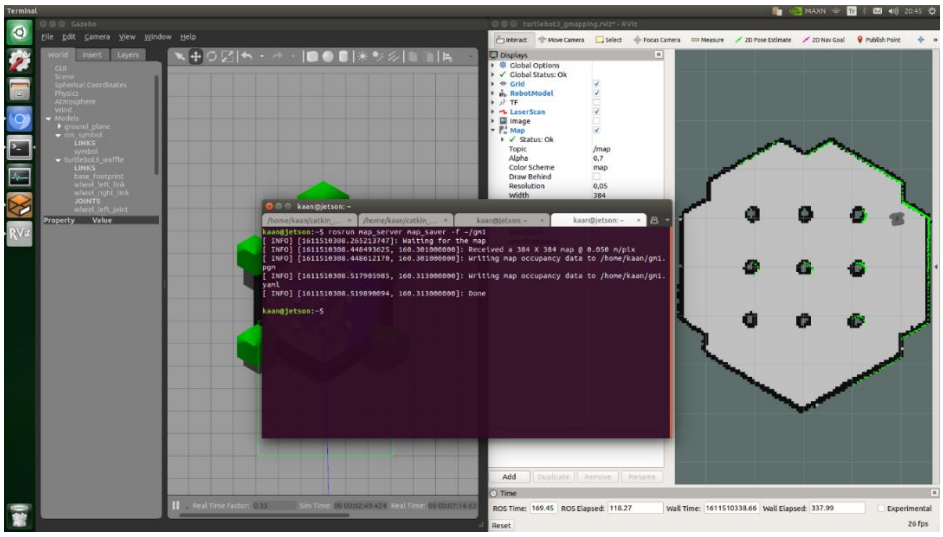
Resim 6.5 Haritalandırma için kullanılan metod

Robot engellerden kaçarken gmapping paketi ile haritalandırma yapılmıştır.



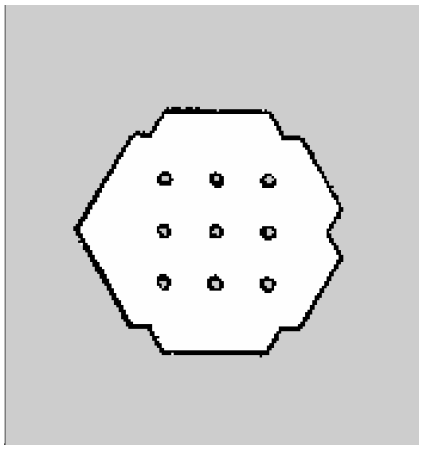
Resim 6.6 Haritalandırma işlemi

Haritalandırma işlemini robotun haritayı tamamen çıkardığını düşündüğümüz zaman durdurabilir veya kaydedebiliriz. Kaydetme işlemi daha sonra robotun rota bulma, olası engellerden kaçma, konum bilgisinin doğruluğunu artırma gibi durumlar için kullanılmaktadır.



Resim 6.7 Harita kaydı

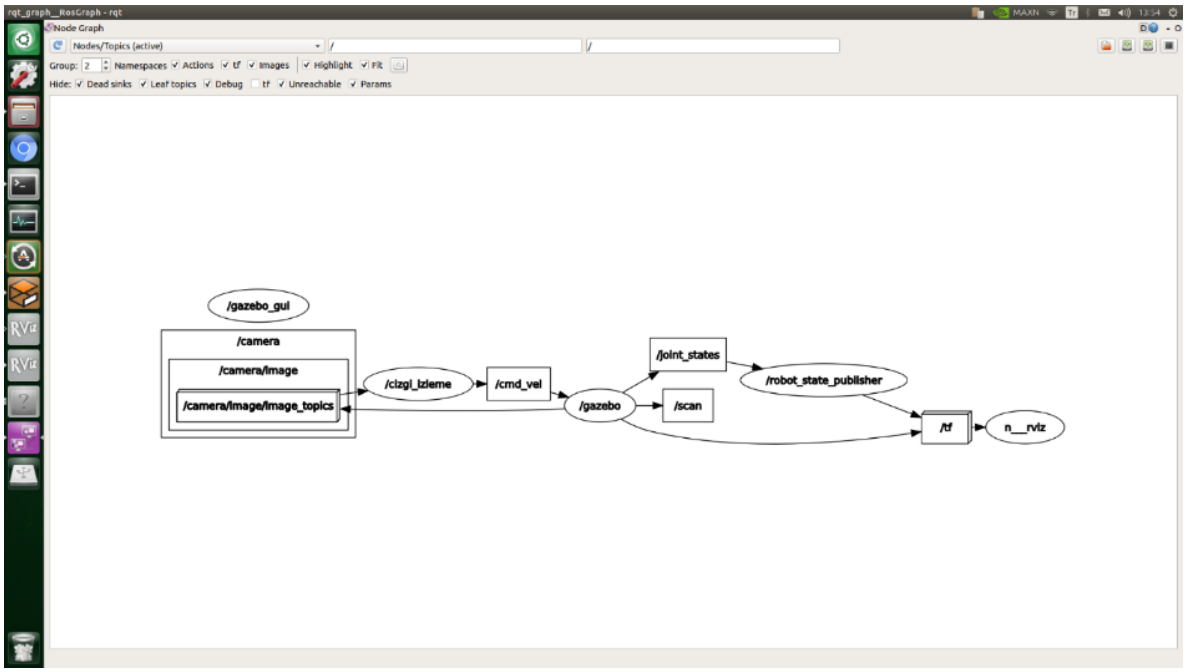
Kaydedilen harita .pmg uzantılı bir dosya içerisindedir.



Resim 6.8 Kaydedilen harita

6.6 Çizgi İzleme

Çizgi izleme algoritması ROS, OpenCV, cv_bridge ve Numpy kütüphanelerini kullanmaktadır. Görüntü işleme adımlarına başlamadan önce rqt_graph ara yüzünden algoritmanın nerelerden veri aldığını, nerelere veri gönderdiğini görselleştirmek daha açıklayıcı olacaktır.

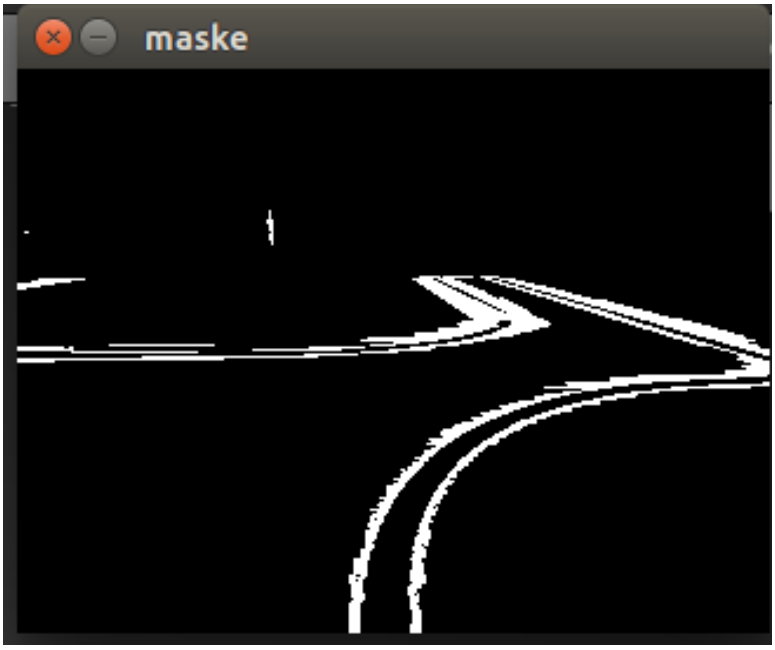


Şekil 6.1 rqt_graph ile çizgi izleme algoritmasının ROS üzerinden haberleşmesi

Görüntüyü alabilmek için Image konusu üzerinden sensor_msgs.msg mesajına abone olmamız gerekmektedir. Robotun hareketlerini kontrol edebilmek için Twist konusu üzerinden geometry_msgs.msg mesajına yayın yapmamız gerekmektedir.

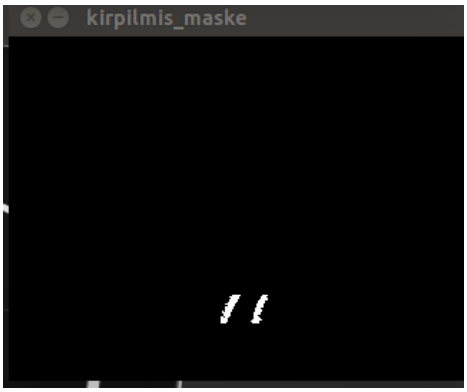
İlk olarak simülasyon ortamındaki robotumuzdan kamera görüntüsünü ROS üzerinden alırız. Daha sonra aldığımız görüntüyü işleyebilmek adına bgr8 formatına dönüştürmeliyiz. Bgr8 bir OpenCV formatıdır. ROS ve OpenCV için kırmızı, yeşil, mavi ana renge dönüşür. Bgr8 formatındaki görüntüde işlem yapabilmek için her pikseldeki renk tonunu tanımlayan HSV renk uzayı sayesinde renkleri renk tonu, renk doygunluğu ve renk değeri olmak üzere üç değişkenle tanımlamamız mümkün hale gelir.

Artık görüntü işlenebilir hale gelmiştir. İzlemek istediğimiz çizginin alt ve üst sınır değerleri belirlenir. HSV formatındaki görüntümüze maskeleme işlemi uygulanır.



Resim 6.9Maskelenmiş görüntü

Maskeleme işleminden sonra görüntüdeki fazladan gözüken çizgileri ve çizgi ile aynı renkle olabilecek herhangi bir cismi görüp algoritmamızın bozulmaması için kırpma işlemini gerçekleştirilmektedir.



Resim 6.10 Kırpılmış görüntü

Gürültülerden arındırılmış görüntü üzerinde bir merkez bulma işlemi yapılmalıdır. Merkez bulmak için Moment fonksiyonu adında bir formülden yararlanılmaktadır.

$$\{\bar{x}, \bar{y}\} = \left\{\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right\} \quad (6.1)$$

Moment fonksiyonunu çizgi izleme senaryomuza entegre ettikten sonra bulunan merkez bir nokta ile işaretlenir.

Resim 6.11 Bulunan merkez noktası

Çizgi ile merkez noktası arasındaki fark bir hata değişkenine atanır. Hata değişkeni bir P geri beslemeli denetim ile sıfıra indirgenmeye çalışır.

```
#!/usr/bin/env python
#-*-coding: utf-8 -*-

import rospy, cv2, cv_bridge, numpy
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist

class Cizgi:

    def __init__(self):

        self.bridge = cv_bridge.CvBridge()

        self.image_sub = rospy.Subscriber('camera/image',Image, self.func)
        self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)

        self.twist = Twist()

    def func(self, ros_goruntu):

        cv_goruntu = self.bridge.imgmsg_to_cv2(ros_goruntu,'bgr8')

        hsv_goruntu = cv2.cvtColor(cv_goruntu, cv2.COLOR_BGR2HSV)

        lower_yellow = numpy.array([ 10, 10, 10])
        upper_yellow = numpy.array([255, 255, 250])

        mask = cv2.inRange(hsv_goruntu, lower_yellow, upper_yellow)

        cv2.imshow('maske', mask)

        h, w, d = hsv_goruntu.shape

        search_top = 3*h/4
        search_bot = 3*h/4 + 20

        mask[0:search_top, 0:w] = 0
        mask[search_bot:h, 0:w] = 0

        cv2.imshow('kirpilmis_maske', mask)

        M = cv2.moments(mask)

        if M['m00'] > 0:

            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])

            cv2.circle(cv_goruntu, (cx, cy), 20, (0,0,255), -1)

            err = cx – w/2

        self.twist.linear.x = 0.1
        self.twist.angular.z = -float(err) / 100
        self.cmd_vel_pub.publish(self.twist)

        cv2.imshow('cizgi izleme', cv_goruntu).
        cv2.waitKey(3)

if __name__ == "__main__":

    rospy.init_node('cizgi_izleme')
    obje = Cizgi()
    rospy.spin()
```

7 SONUÇ

Bu tez çalışmasında haritalandırma ve çizgi takibi yapan bir robot simülasyonu tasarlanmıştır. Robotlar gün geçtikçe yeni bir alanda insan hayatına girmektedir. Bu nedenle robotların otonom olması ve insan hayatını kolaylaştırması önem arz etmektedir. Otonom robotların haritalandırma ve görüntü işleme teknikleri üzerinde durulan bu tez çalışmasında deneyimler elde edilmiştir.

Haritalandırma, rota belirleme, engelleri öngörme ve konum bilgisinin doğruluğunu artırmada önemli bir tekniktir. Özellikle kapalı alanlarda çalışan robotlar için konum bilgisi öğrenme gibi konularda GPS'den çok daha yüksek çözünürlükte bilgi verir. Bu nedenle GPS verilerinin yeterli çözünürlükte alınamadığı alanlarda, GPS verileri yeterli alınsa bile küçük hareketleri lokal haritalandırma teknikleri ile belirlemek çok daha sağlıklı robot hareketleri sağlamaktadır. Haritalandırmada karşılaşılan sıkıntılar genellikle robotun ortam hakkında bir bilgisi olmadığı için daha önce geçtiği alanlarda takılı kalmasından kaynaklanan bütün haritayı gezememe sorunudur. Bu sorun ilk adımda robota elle müdahale ederek giderilmiştir. Ancak algoritma içerisine yerleştirecek başka bir fonksiyon ile bu sorun ortadan kaldırılabilmektedir.

Görüntü işleme ile çizgi takibi teknikleri birçok algoritmayı bünyesinde barındırmaktadır. Bunlar, görüntü işleme, merkez bulma, geri beslemeli kontrol teknikleri gibi gerekli adımları gerçekleştirdikten sonra robota aktarılabilmektedir.

Görüntü işleme tekniklerinden, HSV renk uzayı ile renk tanımlama ve OpenCV sayesinde tanımlanan rengi maskeleme ve kırpma işlemleri gerçekleştirilmiştir. Görüntü işlemede karşılaşılan sıkıntılar robotun kamera konumundan kaynaklanan robotun gördüğü çizgi ile kendi konumu arasında olan farklardan kaynaklı dönüş noktalarını daha yumuşak içeriden almasına sebep olmuştur. Bu sorun birçok şekilde giderilebilir. Daha iyi bir geri beslemeli kontrol ile dönüş noktalarını daha düzgün dönmesi sağlanabilir. Kamera daha alçak bir konumda veya daha eğik bir pozisyonda tutularak robotun önünü daha iyi görmesi sağlanabilir. Çalışmada amaçlanan çizgi takibi işlemi gerçekleştirilmiş olup, ROS ortamının uygunluğu üzerindeki çalışmalar başarıyla gerçekleştirilmiştir.

ROS ortamının robot tasarımı konusunda tasarımcıya sunmuş olduğu araçlar ve düğümlerden oluşan yapısı robot algoritmaları sayesinde daha anlaşılır bir çalışma gerçekleştirilmiştir. ROS ortamının sağlamış olduğu düğüm yapısı, görselleştirme araçları, simülasyon ortamı, çoklu dil desteği, geniş topluluğu ve geniş eğitim kaynakları sayesinde gelecekte robot tasarımcılarının vazgeçilmez ortamı olacaktır.

Araştırmalar üzerinde Jetson NANO kartının rakiplerinden çok daha yüksek performans gösterdiği gözlemlenmiştir. Bu nedenle seçilen Jetson NANO, gücü ile 40-50 fps arasında sağlıklı şekilde simülasyonu gerçekleştirmiştir. ROS ortamının kümülatif büyümesi bu stabil çalışma ortamının en büyük nedenidir. Oluşan hatalar ROS topluluğu üzerinden araştırarak veya konu açarak kolayca çözümlenmiştir.

Alınan değerler, yapılan çalışmalar, karşılaşılan zorluklar karşısında çözüm araştırmaları, hazırlanan algoritmalar, görselleştirilen veriler, simülasyon ortamında harita oluşturma gibi birikimler ışığında; donanım tarafında Jetson NANO'nun doğru seçim olduğu, yazılım tarafında ise ROS'un neler başarabileceğinin sadece bir bölümü görülmesine rağmen büyük bir dünyanın kapısı aralanmıştır. ROS ortamının robot tasarımında vazgeçilmez olduğu görülmüştür. OpenCV kütüphanesinin görüntü işleme teknikleri üzerinde çalışılmış ve makine öğrenmesi çalışmalarının başlangıcı yapılmıştır.

KAYNAKLAR

[1] İnternet: “Özerklik”
https://tr.wikipedia.org/wiki/%C3%96zerklik
[2] İnternet: “Jetson NANO Documentation”
https://www.nvidia.com/tr-tr/autonomous-machines/embedded-systems/jetson-nano/
[3] İnternet: “ROS Documentation”
http://wiki.ros.org/Documentation
[4] İnternet: “Gazebo Simulator”
https://en.wikipedia.org/wiki/Gazebo_simulator
[5] İnternet: “Gazebo About”
http://gazebosim.org/
[6] İnternet: “OpenCV About”
https://opencv.org/about/
[7] İnternet: “Cmake”
https://cmake.org/

ÖZGEÇMİŞ

Kaan KAVAL
Pamukkale/DENİZLİ
E-mail: kaankaval14@gmail.com
GSM: 0(505) 780 6389

KİŞİSEL BİLGİLER

Doğum Tarihi ve Yeri: 14.08.1998 Gölhisar
Medeni Durum : Bekar
Askerlik Durumu : Tecilli

EĞİTİM BİLGİLERİ

Eğitim derece	Eğitim Birimi	Mezuniyet Tarihi
Lise	Nalan Kaynak Anadolu Lisesi	2012-2014
Lise	Final Koleji	2014-2016
Lisans	Gazi Üniversitesi Teknoloji Fakültesi Elektrik Elektronik Mühendisliği	2016-Devam Ediyor

İngilizce [B1+]