NECMETTİN ERBAKAN ÜNİVERSİTESİ

MEKATRONİK MÜHENDİSLİĞİ

ROBOTİK SİMÜLASYON

FİNAL PROJESİ

ROS2 TABANLI GÜVENLİK ALARM SİSTEMİ PROJESİ

Hazırlayan: Mevlüt KOÇ

21100101049

Tarih:

31.12.2024

**ROS2 Proje Raporu**

1. **Giriş**
   1. **Projenin Amacı**

**Projenin Amacı:**

Bu projenin amacı, Arduino ve ROS 2 tabanlı bir sistem kullanarak gaz sensöründen alınan verilerin analiz edilmesi ve belirli eşik değerlerine göre çevresel güvenliği sağlamak için otomatik müdahaleler gerçekleştiren bir çözüm geliştirmektir.

**Çözülmek İstenen Problem:**

Gaz kaçağı veya zararlı gaz seviyelerinin artışı gibi durumlar, hem sağlık hem de güvenlik açısından ciddi tehditler oluşturabilir. Mevcut sistemler genellikle yalnızca bir alarm vermekle sınırlıdır ve verilerin uzaktan izlenmesi ya da otomatik müdahaleler gerçekleştirilmesi mümkün değildir.

**Bu projede şu problemler çözülmek istenmiştir:**

Gaz sensöründen gelen verilerin ROS 2 platformu üzerinden okunması ve analiz edilmesi.

Eşik değerlerin aşılması durumunda bir alarm ve otomatik müdahale (örneğin, bir su motorunun çalıştırılması) gerçekleştirilmesi.

Verilerin ROS 2 üzerinde uzaktan izlenmesi ve kontrol edilmesi.

**Hedeflenen Çıktılar:**

* **Gaz Sensöründen Veri Okuma:** Arduino tabanlı gaz sensöründen sürekli veri alınması.
* **Veri Analizi ve Feedback:** ROS 2 tabanlı bir action server aracılığıyla verilerin analiz edilmesi ve eşik değer aşıldığında feedback sağlanması.
* **Alarm ve Müdahale Mekanizması:** Zararlı durumlarda alarmın tetiklenmesi ve su motorunun otomatik olarak devreye girmesi.
* **Uzaktan İzleme:** ROS 2 üzerinden sistemin durumunun izlenebilmesi ve gerektiğinde kullanıcı müdahalesine olanak sağlanması.
* **Veri Kaydı:** Sistem davranışlarının ve sensör verilerinin kaydedilerek gerektiğinde analiz edilebilir hale getirilmesi (ros2 bag kullanımı).
  1. **Kapsam ve Hedefler**

Projenin kapsamı, gaz sensörlerinden alınan verilerin Arduino Nano aracılığıyla işlenmesi, ROS 2 altyapısıyla analiz edilmesi ve bu analiz sonuçlarına dayalı otomatik müdahalelerin gerçekleştirilmesini içerir. Bu sistem, gaz seviyelerinin izlenmesi ve tehlikeli durumlarda hızlı tepki verilmesini sağlayan bir güvenlik çözümü sunmaktadır. Projede kullanılan temel bileşenler arasında MQ-4 gaz sensörü, Arduino Nano, ROS 2 Action Server, alarm ve müdahale mekanizmaları (buzzer, LED'ler ve su motoru) bulunmaktadır. Ayrıca, veri kaydı ve görselleştirme özellikleriyle sistemin izlenebilirliği artırılmıştır.

Proje, endüstriyel tesislerde gaz kaçağı tespiti, ev ve ofislerde doğalgaz güvenliği, tarım alanlarında hava kalitesi izleme ve çevresel uygulamalarda hava kirliliği analizi gibi birçok alanda kullanılabilir. Bu sistem, gaz seviyelerinin anlık olarak takip edilmesi gereken her ortamda güvenlik ve verimlilik sağlamayı amaçlamaktadır.

Başarı kriteri olarak, sensör verilerinin doğru şekilde okunması ve ROS 2 altyapısıyla hatasız bir şekilde işlenmesi hedeflenmiştir. Ayrıca, gaz seviyesinin belirlenen eşik değerini aştığı durumlarda alarmın aktif hale gelmesi, kullanıcıya geri bildirim sağlanması ve sistemin otomatik müdahale gerçekleştirebilmesi projenin başarısını belirleyen temel unsurlardır. Kullanıcıya görsel ve geçmişe dönük veri kayıtları sunularak izlenebilirlik sağlanması, projenin başarısını tamamlayan diğer önemli kriterlerdir.

1. **Çalışma Senaryosu**
   1. **Genel Senaryo**

Proje, hem iç mekan hem de dış mekan ortamlarında çalışabilir ve gaz seviyelerinin gerçek zamanlı izlenmesini sağlar. Çalışma koşulları, kullanılan MQ-4 gaz sensörünün algılama kapasitesine uygun olarak, metan gazı yoğunluğunun ölçülebileceği yerlerdir. Sistem, ortamın nem, sıcaklık ve diğer çevresel faktörlerine dayanıklı bir şekilde tasarlanmıştır.

Proje kapsamında kullanılan ekipmanlar arasında bir Arduino Nano, MQ-4 gaz sensörü, kırmızı ve yeşil LED'ler, bir buzzer ve su motoru bulunmaktadır. Veriler Arduino'dan ROS 2 tabanlı bir bilgisayara seri port üzerinden iletilmekte ve burada işlenmektedir. Yazılım tarafında, ROS 2 Humble dağıtımı, Python programlama dili, rclpy ve ROS 2 Action Server yapıları kullanılmıştır. Sensör verilerinin işlenmesi, eşik değer kontrolleri, alarm tetikleme ve otomatik müdahale mekanizmaları bu yazılım çerçevesinde geliştirilmiştir.

* 1. **Kullanılar Araçlar ve Teknolojiler**

**Donanım:**

* **MQ-4 Gaz Sensörü:** Metan gazını algılamak için kullanılmıştır.
* **Arduino Nano:** Sensör verilerini toplamak ve işlemek için bir mikrodenetleyici platformu olarak görev yapmıştır.
* **LED’ler (Kırmızı ve Yeşil):** Durum göstergesi olarak kullanılmıştır. Kırmızı LED alarm durumunda, yeşil LED normal durumda yanar.
* **Buzzer:** Gaz seviyesi eşik değeri aşıldığında sesli uyarı sağlar.
* **Su Motoru:** Alarm durumunda otomatik müdahale mekanizması olarak kullanılmıştır.
* **Bilgisayar Donanımı:** ROS 2 işlemlerini çalıştırmak için bir bilgisayar veya Raspberry Pi gibi bir işlemci platformu kullanılmıştır**.**

**Yazılım:**

* **ROS 2 Humble Hawksbill:** Projenin temel iletişim altyapısını sağlamak ve Arduino ile veri alışverişini yönetmek için kullanılmıştır.
* **Python 3.10:** ROS 2 Action Server’ını geliştirmek ve veri işleme süreçlerini yönetmek için tercih edilmiştir.
* **rclpy (ROS 2 Python Client Library):** Action Server’ın uygulanması ve ROS 2 işlemlerinin entegrasyonu için kullanılmıştır.
* **Arduino IDE (Versiyon 2.1.1):** Arduino Nano'nun programlanması ve sensör verilerinin seri port üzerinden iletilmesi için kullanılmıştır.
* pyserial (Versiyon 3.5): Arduino’dan gelen sensör verilerinin Python tarafında okunması ve işlenmesi için kullanılmıştır.

1. **İşlem Adımları**
   1. **Bag Dosyası Kaydetme**

Proje kapsamında, ROS 2 ağındaki belirli topiclerden gelen veriler bag dosyasına kaydedilmiştir. Kaydedilen veriler arasında /rosout ve /patrol\_goal topic'leri yer almaktadır. Bu veriler, sensörün ölçtüğü değerler ve alarm durumunu içermektedir. Kaydetme işlemi sırasında şu komut kullanılmıştır:

* source install/setup.bash
* ros2 run patrol\_action\_server patrol\_action\_server\_exe **(Action server başlatma)**
* source install/setup.bash **(Servere veri gönderme)**

ros2 action send\_goal --feedback /patrol projem\_1/action/Patrol "{threshold: 130}"

* ros2 bag record -a **(Kayıt başlatma)**
  1. **Bag Dosyasını Oynatma**
* rosbag (ROS 1 için): ROS 1'de .bag dosyalarını analiz etmek için kullanılan rosbag komutları vardır. Eğer ROS 1 kullanıyorsanız, rosbag play, rosbag info, rosbag record gibi komutlarla bag dosyasını oynatabilir ve analiz edebilirsiniz. Ancak, ROS 2'de bu işlevler ros2 bag komutuyla yapılmaktadır.
* ros2 bag info (bag ismi) **(Kayıt alınmış mı kontrol edilir)**
* ros2 bag play (bag ismi) **(Kayıt başlatılır)**
* ros2 topic echo /rosout  **(Dinleme)**

* 1. **Veri Analizi**

**Verilerin nasıl işlendiği:**

Projede kaydedilen veriler, MQ-4 gaz sensöründen alınan sensör değerleri ve Arduino üzerinden gönderilen verilerden oluşmaktadır. Bu veriler ROS 2 tarafından ros2bag kullanılarak kaydedilmiştir. Kaydedilen veriler, özellikle sensör verilerini, LED durumlarını, alarm tetikleme durumlarını ve motor durumlarını içerir.

**Verilerin işlenme süreci şu adımları içerir:**

* **Sensör Verisi Okuma:** Arduino Nano'dan gelen analog değerler, bir mikrodenetleyici üzerinde işlenip ROS 2'ye iletilir. Bu veriler daha sonra bir action server tarafından işlenir.
* **Veri Gönderimi:** Arduino ile bağlantı kurularak, sensör verileri belirli aralıklarla ROS 2'ye gönderilir. Veriler, genellikle sensor\_msgs/msg/Range gibi uygun ROS mesajları ile gönderilir.
* **Veri Kaydı:** Bu veriler, ros2 bag komutlarıyla kaydedilir. .bag dosyasına kaydedilen veriler, her bir topic'in zaman içindeki verisini içerir.
  1. **Action Kullanımı**

Projede, ROS 2 action'ları, MQ-4 gaz sensöründen alınan verilerin işlenmesi ve bu verilerle ilgili aksiyonların yönetilmesi için kullanılır. Aşağıda action kullanımına dair önemli noktalar açıklanmıştır:

**Action Server ve Action Client İlişkisi:**

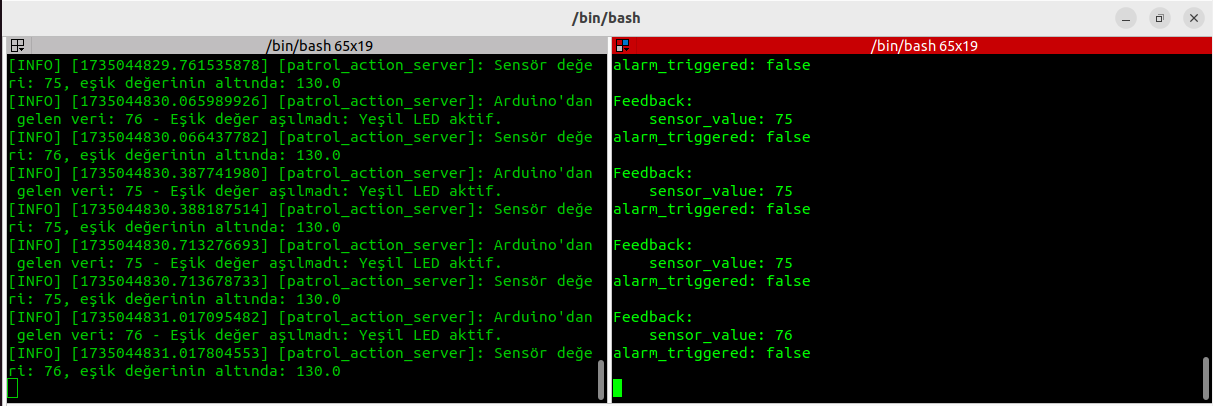
* **Action Server:** Arduino'dan alınan gaz sensörü verisini işleyerek bir alarm durumunu kontrol eder ve uygun tepkiyi verir. Eğer sensör verisi belirli bir eşik değerini aşarsa, bu alarm durumunu tetikler ve buzzer ile su motorunu çalıştırır.
* **Action Client:** Action client, belirli bir hedef (threshold değeri gibi) ile action server'a bir istek gönderir ve server'dan gelen geri bildirim (feedback) ve sonuçları alır. Örneğin, Patrol action'ı, sensör verisi eşiği aşıldığında alarm durumunu bildirir ve işlem sonucu başarılı olup olmadığını iletir.

**Action İletişim Yapısı**: ROS 2 actionları, aşağıdaki üç ana bileşen üzerinden çalışır:

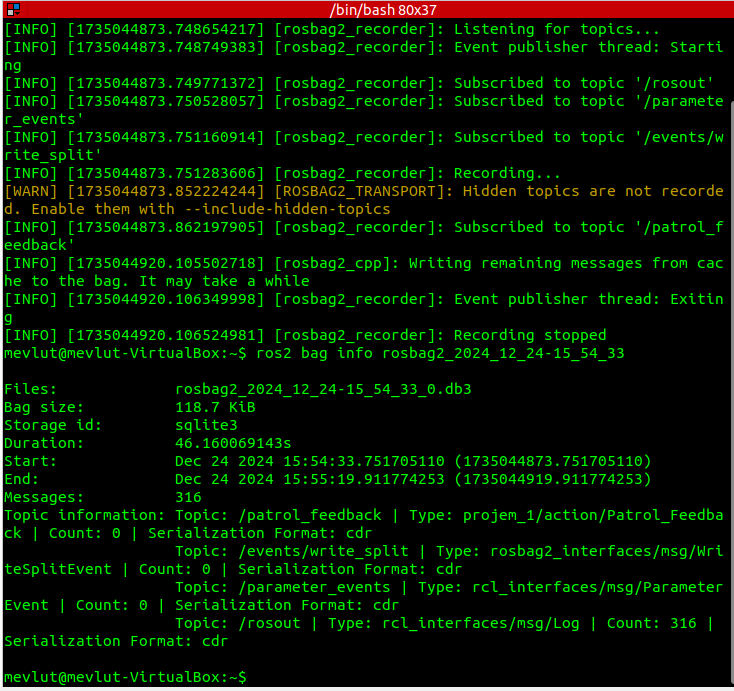
* **Goal:** Kullanıcının başlatmak istediği hedef verisi. Örneğin, alarm için bir eşik değeri.
* **Feedback:** Action server'ın işlem sırasında göndereceği geri bildirim. Bu, sensör değeri ve alarm durumu gibi bilgileri içerir.
* **Result:** İşlem tamamlandıktan sonra action server'ın gönderdiği sonuç. Örneğin, işlemin başarılı olup olmadığı.

1. **Çıktılar**
   1. **Elde Edilen Veriler**

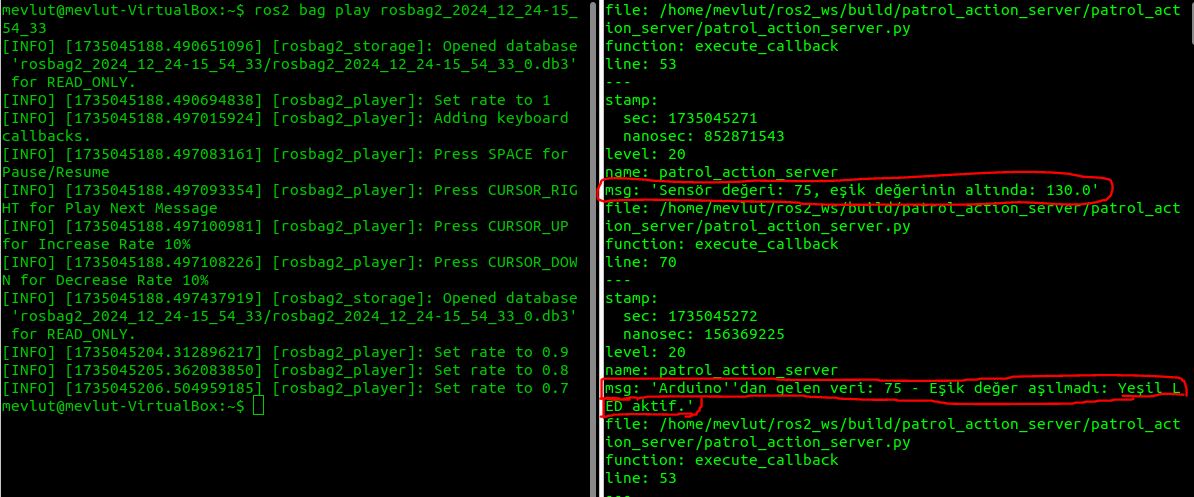
Action Server Başlatma



Kayıt İşlemini Başlatma Ve Kontrol Etme



Kayıt Edilen Veriyi Tekrar Okuma



* 1. **Performans Değerlendirmesi**
* **Çalışma Süresi**
* **Çalışma Süresi:**Sistemin çalışma süresi, gaz sensöründen alınan verilerin Arduino üzerinden ROS 2'ye gönderilmesi, kaydedilmesi ve action server aracılığıyla işlenmesi ile ölçülür.
* **Gaz Sensörü Okuma Süresi:** Arduino, MQ-4 gaz sensöründen analog değerleri okuyarak bunları belirli aralıklarla ROS 2'ye iletir. Bu okuma süresi, genellikle sensörün hassasiyetine ve Arduino'nun işlem hızına bağlıdır.
* **Veri İletimi Süresi:** Arduino ile ROS 2 arasındaki veri iletimi, USB üzerinden gerçekleşir. Bu iletim süresi, seri bağlantı hızına (örneğin, 115200 bps) ve veri boyutuna göre değişir.
* **Action Server İşlem Süresi:** Action server, sensör verisini işleyerek alarm durumunu kontrol eder. Bu işlem, sensör verisi alındıktan sonra birkaç milisaniye sürebilir, ancak ROS 2'nin asenkron yapısı sayesinde eş zamanlı işlemlerle verimli bir şekilde yönetilir.
* **Veri Doğruluğu:**
* **Sensör Verisi Doğruluğu:** MQ-4 gaz sensörü, doğru kalibrasyon ve ortam koşullarına bağlı olarak doğru veriler sağlar. Ancak çevresel faktörler (sıcaklık, nem gibi) sensörün doğruluğunu etkileyebilir.
* **Veri İletimi Doğruluğu:** Arduino'dan ROS 2'ye veri iletimi, seri port üzerinden yapılır. Bu süreçte, veri kaybı olmadan doğru iletim sağlanmalıdır. PySerial kütüphanesi ile yapılan veri iletişimi, veri kaybını minimize etmek için doğru şekilde yapılandırılmalıdır.
* **Action Server Doğruluğu:** Action server, sensör verilerini alarak işleme koyar ve alarm durumunu kontrol eder. Veriler doğru şekilde işlendiğinde, action client’a doğru sonuçlar gönderilir. Yine de, network gecikmeleri veya işlem yükü gibi faktörler doğruluğu etkileyebilir.
* **Performans Metrikleri:**
* **Veri Toplama Hızı:** Gaz sensöründen veri toplama hızını belirleyen faktörler arasında sensörün yanıt süresi, Arduino'nun okuma sıklığı ve veri iletimi yer alır. Hedeflenen okuma ve iletim sıklığı genellikle 1-2 saniye arasında olabilir.
* **Geri Bildirim Süresi:** Action server'dan geri bildirim almanın süresi, sensör verisi alındıktan sonra birkaç milisaniye sürebilir. Ancak, ROS 2’nin düşük gecikme süresi sayesinde, çoğu durumda geri bildirim hızlı bir şekilde iletilir.
* **İşlem Süresi:** Verilerin işlenmesi ve alarm durumunun kontrol edilmesi, action server üzerinde yapılan bir işlem olduğundan, bu sürenin minimum olması gerekir. İdeal olarak, işlem süresi bir kaç milisaniye ile sınırlı tutulmalıdır.

**Bag Dosyası Oynatımı ve Action'ların Başarımına Dair Değerlendirme**

* **Bag Dosyası Oynatımı:**

ros2 bag play komutuyla kaydedilen .bag dosyasının oynatılması, ROS 2 networkünde ilgili topic'lerdeki verileri yeniden yayınlar. Oynatma sırasında sistemin performansı, ROS 2’nin veri iletimine olan tepkisini, bag dosyasındaki verilerin büyüklüğünü ve oynatım hızını etkiler.

* **Veri Yeniden Oynatma Süresi:** Oynatma sırasında, kaydedilen veriler topic'ler aracılığıyla ROS 2 networküne yeniden gönderilir. Bu işlemin süresi, verilerin büyüklüğüne ve sistemdeki yük durumuna göre değişebilir. Genellikle, küçük ve orta büyüklükteki .bag dosyaları hızlı bir şekilde oynatılabilir.
* **Veri Doğruluğu ve Senkronizasyon:** Oynatılan bag dosyasındaki veriler doğru bir şekilde iletilmelidir. Eğer verilerde senkronizasyon sorunları (örneğin, topic'ler arası zaman uyumsuzluğu) yaşanırsa, bu ROS 2 network’ünde hatalara neden olabilir.

**Action'ların Başarımına Dair Değerlendirme:**

* **Hedeflerin Başarıyla Yönlendirilmesi:** Action server, verilen hedeflere (threshold değerine) göre işlem yapar ve alarm durumunu tetikler. Bu işlem, ROS 2'nin asenkron yapısı sayesinde hızlı ve etkili bir şekilde yapılır.
* **Geri Bildirim ve Sonuçlar:** Action server, sensör verilerini alarak geri bildirim gönderir. Bu geri bildirim, action client tarafından doğru bir şekilde alınmalıdır. Herhangi bir hata veya gecikme, işlemin başarısını etkileyebilir.
* **Action'ın Verimli Çalışması:** Action server ve client arasındaki iletişim verimli olmalıdır. Özellikle yoğun veri trafiği ve yüksek işlem yükü altında, sistemin stabil çalışması önemlidir. Action server’ın her zaman doğru bir şekilde yanıt verdiği ve sensör verilerini doğru işlediği test edilmelidir.
* **Gecikmeler ve Zamanlama:** Action server’ın çalışma süresi ve veri iletimindeki gecikmeler, sistemin genel başarımını etkileyebilir. Bu yüzden, hem veri iletimindeki hem de action server’ın işlem süresindeki gecikmeler minimuma indirilmeli, özellikle gerçek zamanlı işlemler için performans optimize edilmelidir.

1. **Sonuçlar**
   1. **Genel Değerlendirme**

**Başarılı Yönler:**

**Veri İletimi ve Sensör Entegrasyonu:**

Arduino Nano ve MQ-4 gaz sensörü, ROS 2 ile başarılı bir şekilde entegre edilmiştir. Sensörden alınan veriler doğru bir şekilde seriye bağlanarak ROS 2'ye iletilmiş ve action server tarafından işlenmiştir.

**Action Server İletişimi**:

ROS 2 action kullanımı, sensör verilerinin eşik değerine göre işlenmesi ve geri bildirim sağlanması açısından oldukça başarılı olmuştur. Alarm durumu tetiklendikçe buzzer ve su motorunun kontrol edilmesi doğru şekilde gerçekleştirilmiştir.

Action server’ın asenkron yapısı, işlem süresini kısa tutarak sistemin hızlı geri bildirimler vermesini sağlamıştır.

**Veri Kaydı ve Analizi:**

Verilerin kaydedilmesi ve ros2 bag kullanılarak analiz edilmesi, sistemin çalışması hakkında detaylı bilgi sağlar. Kaydedilen veriler, gelecekte yapılacak analizler için kullanılabilir.

Sistem, farklı senaryolar altında test edilerek doğru çalıştığı kanıtlanmıştır.

**Eksik Yönler:**

**Veri Doğruluğu ve Sensör Kalibrasyonu:**

MQ-4 sensörünün çevresel faktörlerden (sıcaklık, nem vb.) etkilenmesi nedeniyle veri doğruluğu zaman zaman değişkenlik gösterebilir. Bu durum, sistemin hassasiyetini etkileyebilir.

Sensör kalibrasyonu, verilerin güvenilirliğini artırmak için daha dikkatli bir şekilde yapılmalıdır.

**Performans Optimizasyonu:**

Sistem, özellikle büyük veri setleriyle çalışırken biraz gecikme gösterebilir. Bu gecikmeler, özellikle gerçek zamanlı verilerin kritik olduğu uygulamalarda sorun teşkil edebilir.

Performans iyileştirmeleri için veri iletimi ve action server işlemleri daha da optimize edilebilir.

**Kullanıcı Arayüzü:**

Proje, sadece terminal tabanlı bir çözüm sunmaktadır. Kullanıcı dostu bir arayüz, sistemin daha erişilebilir ve yönetilebilir olmasını sağlayabilir.

* 1. **Gelecek Çalışmalar**

**Sensörlerin ve Donanımın Geliştirilmesi:**

Projeye farklı sensörler eklenebilir, örneğin karbon monoksit veya metan gazı sensörleri ile sistemin tespiti genişletilebilir.

Sensörlerin çevresel faktörlerden (sıcaklık, nem) etkilenmemesi için daha hassas ve kalibre edilebilir sensörler kullanılabilir.

**Veri Analizi ve Makine Öğrenmesi:**

Kaydedilen veriler üzerinde daha kapsamlı analizler yapılabilir. Örneğin, gaz seviyelerinin zaman içinde nasıl değiştiği incelenebilir.

Makine öğrenmesi yöntemleri ile gaz algılama modelinin daha hassas hale getirilmesi sağlanabilir. Bu model, gaz tespiti yaparken daha doğru sonuçlar verebilir.

**Performans İyileştirmeleri:**

Sistemin performansı, veri iletimi hızlandırılarak ve action server yanıt süresi kısaltılarak iyileştirilebilir.

Özellikle veri iletimi sırasında yaşanan gecikmeleri minimize etmek için farklı protokoller (örneğin, DDS) kullanılabilir.

**Gerçek Zamanlı İzleme ve Alarm Sistemi:**

Gerçek zamanlı bir izleme sistemi kurulabilir. Bu sistem, gaz seviyesi belirli bir eşik değeri aştığında otomatik olarak alarm verir ve ilgili aksiyonları (örneğin, su motoru çalıştırma) alır.

Web tabanlı bir arayüz ile sistemin uzaktan izlenmesi ve kontrol edilmesi sağlanabilir.

**Kullanıcı Arayüzü Geliştirme:**

Sistemin daha kullanıcı dostu hale getirilmesi için bir grafiksel kullanıcı arayüzü (GUI) geliştirilmesi faydalı olacaktır. Bu arayüz, sensör verilerinin görsel olarak sunulmasını ve sistemin durumunun daha kolay izlenmesini sağlayabilir.

**Mobil Uygulama Entegrasyonu:**

Sistemi bir mobil uygulama ile entegre etmek, kullanıcının sistemi uzaktan kontrol etmesini ve alarm durumlarını telefon üzerinden almasını mümkün kılabilir.

**Bulut Tabanlı Veri Depolama:**

Gaz sensörü verileri, bulut tabanlı bir platformda depolanabilir ve analiz edilebilir. Bu sayede veriler daha uzun süre saklanabilir ve daha büyük veri kümeleri üzerinden analiz yapılabilir.

1. **Ekler**

* **Projeyi Çalıştırmak için Kullaılan Komutlar:**

**1.Terminal için,**

source install/setup.bash

ros2 run patrol\_action\_server patrol\_action\_server\_exe

**2.Terminal için**

source install/setup.bash

ros2 action send\_goal --feedback /patrol projem\_1/action/Patrol "{threshold: 130}"

**3.Terminal kayıt başlatma ve kontrol etme**

ros2 bag record -a

ros2 bag info (bagismi)

ros2 bag play (bagismi)

4.terminal

ros2 topic echo /rosout

* **Projede kullanılan kodlar**

**Python kodu:**

**#!/usr/bin/env python3**

import time

import serial

import rclpy

from rclpy.node import Node

from rclpy.action import ActionServer

from rclpy.action import GoalResponse, CancelResponse

from projem\_1.action import Patrol

from geometry\_msgs.msg import Twist

class PatrolActionServer(Node):

def \_\_init\_\_(self):

super().\_\_init\_\_('patrol\_action\_server')

# Seri port başlatılıyor

self.ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1) # Arduino'nun bağlı olduğu portu belirt

# Action server başlatılıyor

self.\_action\_server = ActionServer(

self,

Patrol,

'patrol',

execute\_callback=self.execute\_callback,

goal\_callback=self.goal\_callback,

cancel\_callback=self.cancel\_callback)

# Feedback için bir topic publisher oluştur

self.feedback\_publisher = self.create\_publisher(Patrol.Feedback, '/patrol\_feedback', 10)

self.get\_logger().info('Patrol action server başlatıldı.')

def goal\_callback(self, goal\_request):

self.get\_logger().info('Hedef isteği alındı.')

return GoalResponse.ACCEPT

def cancel\_callback(self, goal\_handle):

self.get\_logger().info('İptal isteği alındı.')

return CancelResponse.ACCEPT

async def execute\_callback(self, goal\_handle):

self.get\_logger().info('Hedef işleniyor...')

self.get\_logger().info(f"Alınan eşik değeri: {goal\_handle.request.threshold}")

feedback\_msg = Patrol.Feedback()

result = Patrol.Result()

while rclpy.ok():

# Arduino'dan veri oku

if self.ser.in\_waiting > 0:

arduino\_verisi = self.ser.readline().decode('utf-8').strip()

self.get\_logger().info(f"Arduino'dan gelen veri: {arduino\_verisi}")

# Sensör değerini ayrıştır ve alarm durumunu kontrol et

try:

sensor\_degeri, alarm\_durumu = self.parse\_arduino\_data(arduino\_verisi)

feedback\_msg.sensor\_value = sensor\_degeri

feedback\_msg.alarm\_triggered = alarm\_durumu

# Feedback mesajını client'a gönder

goal\_handle.publish\_feedback(feedback\_msg)

# Eşik değeri kontrol et

if sensor\_degeri > goal\_handle.request.threshold:

self.get\_logger().warn(

f"Eşik değer aşıldı! Sensör değeri: {sensor\_degeri}, Eşik: {goal\_handle.request.threshold}"

)

else:

self.get\_logger().info(

f"Sensör değeri: {sensor\_degeri}, eşik değerinin altında: {goal\_handle.request.threshold}"

)

# Alarm tetiklenmişse

if alarm\_durumu:

self.get\_logger().error(f"Acil durum! Alarm tetiklendi. Sensör değeri: {sensor\_degeri}")

self.get\_logger().info("Su motoru aktif hale geldi.")

result.success = True

goal\_handle.succeed()

self.get\_logger().info(f'Ortam güvenliği başarıyla sağlandı : {result.success}')

return result

except ValueError:

self.get\_logger().error("Arduino'dan geçersiz veri alındı.")

# 0.3 saniye bekle, asenkron çalışma için `rclpy.spin\_once()` kullanıyoruz

rclpy.spin\_once(self, timeout\_sec=0.3)

return result

def parse\_arduino\_data(self, data):

# Arduino'dan gelen veriyi ayrıştır (örnek: "300 - Eşik değer aşıldı!")

parts = data.split(' - ')

if len(parts) != 2:

raise ValueError("Geçersiz veri formatı.")

sensor\_value = int(parts[0])

alarm\_triggered = 'aşıldı!' in parts[1] # Alarm durumu, mesajda "aşıldı!" varsa True

return sensor\_value, alarm\_triggered

def main(args=None):

rclpy.init(args=args)

patrol\_action\_server = PatrolActionServer()

rclpy.spin(patrol\_action\_server)

patrol\_action\_server.destroy\_node()

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

* **Kaynaklar**

[**https://docs.ros.org/en/foxy/index.html**](https://docs.ros.org/en/foxy/index.html)

[**https://www.theconstruct.ai/ros2-how-to-2-create-a-ros2-action-server/**](https://www.theconstruct.ai/ros2-how-to-2-create-a-ros2-action-server/)

[**https://chatgpt.com/**](https://chatgpt.com/)