2023 서울 지능형 사물인터넷(AIOT) 해커톤 대회

분석 보고서

팀 mechanics

한양대학교 응용물리학과 박민혁

한양대학교 응용물리학과 이규현

한양대학교 응용물리학과 이민성

목차

1. 한강 교량 투신 자살 사건

2. 해결 방법

2-1. 개요

Hydrophone(수중음향기) 이용

2-2. 이론

* Hydrophone의 개념
* Hydrophone을 이용한 측정 방법
* 사용할 Hydrophone의 종류 ([AB] Enclosure Hydrophone M14)
* [AB] Enclosure Hydrophone M14를 선택한 이유
* [AB] Enclosure Hydrophone M14의 설치 위치 및 개수
* [AB] Enclosure Hydrophone M14를 이용한 측정 시나리오
* 측정 방법에 대한 문제점과 해결 방법
* AIOT를 이용한 [AB] Enclosure Hydrophone M14로 측정한 결과와 개발 프로그램과 통신 방법
* 예상 비용

3. 어플리케이션 개발 시현

# 1. 한강 교량 투신 자살 사건

<기사 1>에 나와있는 한강 대학생 사망 사건을 아시나요? 꽃다운 나이 22세에 자살이라는 선택을 하였는데 너무 안타까운 사건이라고 생각합니다.

이 사건 뿐만 아니라 <그래프 1>과 같이 여러 한강 교량 투신자살 사건이 증가하고 있으며, 아래 <기사 2>를 보시면 아시다시피 서울시에서는 대책을 마련하려고 여러 생각을 하고 있습니다.

라인, 도표, 그래프, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<그래프 1> 최근 5년간 한강 교량 투신 자살 시도 횟수

텍스트, 하늘, 스크린샷, 야외이(가) 표시된 사진

자동 생성된 설명

<기사 1>

텍스트, 스크린샷, 물, 하늘이(가) 표시된 사진

자동 생성된 설명

<기사 2>

# 2. 해결 방법

- 개요

저희 팀 mechanics는 이러한 한강 교량 투신 자살 사건을 해결하기 위해 원래 한강의 파동의 진폭과 주파수의 변화를 파악하여 문제를 하려고 하였지만, 조사를 통한 결과 파동을 파악하는 데는 여러 변수가 발생하는 문제점이 있었습니다. 그래서 재조사를 통하여 수중 소음 패턴을 파악할 수 있다는 더 좋은 Hydrophone이라는 것을 찾았습니다.

그래서 저희는 Hydrophone(수중 음향기)을 이용하여 한강의 수중 소음 패턴을 파악한 뒤 이상이 감지된다면 AIOT를 이용하여 관할 소방서에 어플리케이션을 통해 경보를 알리도록 할 것입니다. 이를 통하여 투신 후 빠른 대응이 가능하게 됩니다.

- 이론

1. Hydrophone의 개념

\* 기본 원리: Hydrophone이란, 수중의 초음파의 음장을 측정하기 때문에 넓은 범위의 주파수의 초음파를 고분해능으로 측정할 수 있는 수신전용 초소형 트랜스튜서입니다. 대부분의 Hydrophone은 세라믹, 피에조 전기 소재 또는 기타 민감한 소재를 사용하여 압력 변화를 감지하며, 이러한 변화를 전기 신호로 변환합니다.

\* 주요 특성:   
주파수 응답: Hydrophone은 특정 주파수 범위 내에서 가장 잘 작동합니다. 이 주파수 범위는 Hydrophone의 디자인 및 소재에 따라 다르며, 특정 애플리케이션에 따라 선택됩니다.

민감도: Hydrophone의 민감도는 감지된 압력 변화의 크기에 따라 출력되는 전기 신호의 크기를 나타냅니다.

\* 사용 용도:

수중 음향 연구: Hydrophone은 바다나 호수에서의 동물들의 소리, 지진, 해저 활동 등의 소리를 연구하는 데 사용됩니다.

해양 생물학: 돌고래, 고래 및 기타 수중 동물들의 소리를 연구하고 기록하는 데 사용됩니다.

산업 및 군사: Hydrophone은 선박의 소음 감지, 잠수함 탐지, 해저 파이프라인의 유출 감지 등에 사용됩니다.

환경 모니터링: Hydrophone은 물의 질 변화나 특정 환경 요인에 따른 수중 소음 변화를 모니터링하는 데 사용될 수 있습니다.

2. Hydrophone을 이용한 측정 방법

\* Hydrophone 설치:

Hydrophone을 적절한 위치에 설치합니다. 감지 범위와 대상 지역을 고려하여 최적의 위치에 설치하는 것이 중요합니다.

\*감지 민감도 설정

초기 설정은 평상시 물의 배경 소음을 기록하여, 이보다 큰 소음이 감지되면 경보가 작동하도록 설정할 수 있습니다.

\*배경 소음 기록

먼저 Hydrophone을 사용하여 정상적인 상황에서의 배경 소음을 기록합니다. 이를 통해 사람이 떨어지는 소리와 구분할 수 있는 기준을 설정합니다.

\*경보 시스템 연결

Hydrophone의 출력을 애플리케이션에 연결합니다. 소음이 일정 기준을 초과하면 경보가 울리고 CCTV가 켜지도록 설정합니다.

\*테스트 및 보정

Hydrophone의 설정을 테스트하기 위해 실제로 물에 무언가를 떨어뜨려 소음을 발생시켜 보고, 그에 따른 반응을 확인합니다. 필요에 따라 감지 민감도나 경보 기준을 보정할 수 있습니다.

\*지속적 모니터링 및 유지보수

Hydrophone의 작동 상태를 지속적으로 모니터링하며, 필요시 유지보수를 실시합니다.

사람이 물에 떨어질 때 발생하는 소음의 특성, 물의 깊이, 현장 조건 등을 고려하여 설정을 조정해야 합니다. 또한, 실제로 사람이 떨어질 때의 소음과 다른 원인으로 인한 소음을 구분하기 위한 추가적인 연구와 실험이 필요할 수 있습니다.

3. 사용할 Hydrophone의 종류

Imago(“<https://www.imago.co.kr/shop/item.php?it_id=2824950130>”) 에서 판매하는 [AB] Enclosure Hydrophone M14를 사용할 것입니다.

**텍스트, 케이블, 헤드폰, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

이는 한강 수심 10m에 비해 수심 100m의 수압을 견딜 수 있으며 20kHz인 상대적인 고주파의 주파수 범위에서 사운드를 캡쳐할 수 있습니다.

만약 저희가 조사한 것과 다르게 최대 20kHz의 주파수로 수중 음향 측정이 힘들다면 Rectuson (<http://www.rectuson.com/ko/product/7_product_03_2.jsp>) 에서 판매하는 TC4038을 사용할 수 있습니다. 이는 50kHz~800kHz로 훨씬 더 높은 주파수 범위의 사운드를 캡처할 수 있으며 한강 수중 음향을 더 잘 파악할 수 있습니다. 하지만 현재로서는 가격을 알 수 없어 조사를 하지 못하였습니다.



4. [AB] Enclosure Hydrophone M14를 선택한 이유

높은 민감도: 이 Hydrophone은 주파수 범위 70Hz-20kHz에서 약 60dB의 높은 민감도를 보이므로, 물 속에서 발생하는 다양한 소음을 선명하게 감지할 수 있습니다. 이를 통해 다리 주변에서의 물체의 움직임, 동물의 활동 또는 다른 수중 활동을 잘 감지할 수 있습니다.

와이드 주파수 응답: 70Hz-20kHz의 주파수 범위는 수중에서 발생하는 다양한 소리를 포착하기에 충분합니다. 이를 통해 다리에서 사람이 떨어지는 사건을 감지하는 데 유용할 수 있습니다.

깊은 수심에서도 사용 가능: ASF-G는 최대 수심 100m까지 사용할 수 있기 때문에, 깊은 물에서도 안정적으로 작동하며, 깊은 곳에서의 소음 변화도 감지할 수 있습니다.

플러그앤플레이: 3.5mm TRS 잭을 통한 연결이 가능하므로, 다양한 기기와의 호환성이 좋습니다. 이를 통해 쉽게 설치 및 모니터링 시스템과 연결할 수 있습니다.

에너지 효율: 9V 배터리로 운영되므로, 전원 공급이 간편하며, 에너지 효율이 좋습니다.

안정적인 모니터링: ASF-1 기능을 통해 안정적인 모니터링 환경을 제공하며, 물 속에서의 다양한 활동이나 변화를 실시간으로 모니터링할 수 있습니다.

이러한 장점들을 바탕으로, 한강의 다리에 이 Hydrophone을 설치하면 사건 또는 사고 감지, 환경 모니터링, 수중 동물 활동 모니터링 등 다양한 용도로 사용될 수 있습니다.

5. [AB] Enclosure Hydrophone M14 설치 위치 및 설치 개수

Hydrophone의 감지 범위: ASF-G는 주변의 소음을 60dB 이하로 제한하면서, 70Hz-20kHz의 주파수 범위에서 소리를 감지할 수 있습니다. 이는 사람이 물에 떨어질 때 발생하는 소리를 감지하는 데 충분할 것으로 보입니다.

감지 거리: 마지막으로 제공된 정보에 따르면, ASF-G는 깊은 물에서도 소리 감지를 유지하며 최대 100m의 거리까지 소리를 감지할 수 있다고 합니다.

대교의 평균 길이: 대교의 평균 길이를 고려하여, Hydrophone의 감지 범위가 겹치지 않게 설치해야 합니다.

Hydrophone의 감지 거리가 최대 100m이므로, 대교의 평균 길이를 약 1,200m로 가정하면 대략 12개의 Hydrophone이 필요할 것으로 추정됩니다. 하지만, 실제로는 다리의 길이나 구조, 그리고 다리 밑의 물의 깊이나 흐름 등 여러 요소를 고려해야 합니다.

-> 따라서, 12개를 기본으로 하되, 외부 노이즈나 간섭, 그리고 다리의 특정 위치나 수심 등의 요소를 고려하여 추가적으로 설치할 필요가 있을 수 있습니다.

Hydrophone의 설치 위치: 12개의 Hydrophone을 가지고 있다면, 다리의 전체 길이에 균일하게 분포시켜 설치하는 것이 좋습니다. 예를 들어, 다리의 양 끝, 중앙, 그리고 그 사이의 균등한 간격으로 Hydrophone을 설치하면 전체 다리를 커버할 수 있을 것입니다. 이렇게 하면 사람이 떨어지는 위치에 따라 가장 가까운 Hydrophone이 소음을 감지할 수 있게 됩니다.

6. [AB] Enclosure Hydrophone M14을 이용한 측정 가정 시나리오

한강 대교의 평균 길이 1200m, 높이 10m, 한강의 평균 깊이 10m에서 70kg인 이모씨가 날씨가 화창한 날 22시 30분에 떨어집니다. [AB] Enclosure Hydrophone M14를 다리 밑 강 속에 12개를 골고루 분포했다고 가정을 해보았습니다.

먼저, 이러한 시나리오에서 사람이 물에 떨어질 때 발생하는 소리의 데시벨 패턴을 가상으로 생성해봤습니다. 이 시나리오에서는 사람이 물에 떨어지는 것을 가상으로 1초 동안 100dB로 나타나게 하고, 그 외의 시간 동안은 40dB에서 60dB 사이의 랜덤한 소음으로 설정합니다.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.svm import SVC

# 가상의 데이터 생성

hours = 24

minutes\_per\_hour = 60

seconds\_per\_minute = 60

total\_seconds = hours \* minutes\_per\_hour \* seconds\_per\_minute

# 기본 배경 소음 (40dB ~ 60dB)

noise = np.random.uniform(40, 60, total\_seconds)

# 22시 30분에 사람이 떨어지는 소리 (100dB) 가정

fall\_time = 22 \* minutes\_per\_hour \* seconds\_per\_minute + 30 \* seconds\_per\_minute

noise[fall\_time:fall\_time + 1] = 100

# 시간 배열 생성

time = np.arange(total\_seconds) / (minutes\_per\_hour \* seconds\_per\_minute)

# 그래프 그리기

plt.figure(figsize=(12, 6))

plt.plot(time, noise, label='Detected Sound Level', color='blue')

plt.axvline(x=22.5, color='red', linestyle='--', label='Person Falls')

plt.xlabel('Time (hours)')

plt.ylabel('Decibel (dB)')

plt.title('Hydrophone Detection over 24 Hours')

plt.legend()

plt.tight\_layout()

plt.show()

스크린샷, 라인, 그래프, 직사각형이(가) 표시된 사진

자동 생성된 설명

1. \*\*배경 소음 (40dB ~ 60dB):\*\*

대부분의 자연 환경에서의 배경 소음은 약 40dB에서 60dB 사이로 측정됩니다. 이 범위 안에서의 무작위 값은 자연 환경의 일반적인 변동성을 반영하기 위한 것입니다. 이 소음은 강물의 흐름, 동물들의 소리, 바람, 기타 환경적 요소들 때문에 발생할 수 있습니다.

2. \*\*사람이 물에 떨어지는 소리 (100dB):\*\*

사람이 물에 떨어질 때 발생하는 소리는 주변 환경의 소음에 비해 상당히 크다고 가정했습니다. 100dB는 매우 큰 소음을 나타내며, 이 값은 사람이 물에 떨어지는 충격과 관련된 소리의 근사값으로 사용되었습니다. 실제로는 이 값이 더 낮거나 높을 수 있습니다.

**이러한 결과를 따르면 사람이 떨어지는 것을 Hydrophone으로 측정할 수 있습니다.**

7. 측정 방법에 대한 문제점과 해결 방법

문제점: 한강의 지나가는 배에 의한 데시벨, 비에 의한 데시벨, 태풍 등 자연재해에 인한 데시벨과 사람이 떨어지는 소음을 구분할 수 있을까?

라인, 스크린샷, 그래프, 도표이(가) 표시된 사진

자동 생성된 설명

<그림 2>

Person Falling (Impact Sound): 사람이 떨어지는 소리는 일시적이고 갑작스러운 증가를 나타내며, 주로 저주파 대역의 소음을 생성합니다.

Ship Passing: 배가 지나가는 소음은 보통 지속적이고 규칙적인 패턴을 띕니다. 이것은 특정 시간 동안 40dB의 증가로 가정되어 있는데, 이는 중간 크기의 선박이 지나갈 때 항구나 연안 지역에서 경험할 수 있는 소음 수준입니다.

Rain Sound: 비가 내리는 소리는 일정하고 연속적인 소음을 생성하며, 주로 고주파 대역의 소리를 포함합니다. 이 모델에서는 0.5dB의 상대적으로 낮은 변동으로 모델링되어 있는데, 이는 조용한 비 내림 소리를 나타냅니다.

Natural Disaster: 자연재해(예: 폭풍, 지진 등)에서 발생하는 소음은 매우 강렬하고 비정상적일 수 있으며, 여기에서는 60dB의 큰 증가로 가정되어 있습니다. 이는 강한 바람, 천둥, 또는 물리적 구조물이 파괴되는 소리 등과 같이 매우 높은 에너지를 포함하는 소음 상황을 나타낼 수 있습니다.

Clear Day Background Noise: 맑은 날의 소음은 일상적인 환경 소음을 나타내며, 매우 낮은 수준의 변동으로 모델링되어 있습니다.

소음이 일정시간동안 지속되느냐 아니면 짧은 시간의 순간적인 이상치가 발생하느냐에 따라 구분해 낼 수 있다고 생각해 임의의 시뮬레이션 예상치를 위 그래프를 통해 시각화 한 것입니다.

8. AIOT를 이용한 [AB] Enclosure Hydrophone M14로 측정한 결과와 개발 프로그램과 통신 방법

Hydrophone 센서에서 얻은 데이터를 게이트웨이를 통해 IoT 디바이스에 보내지는 방식은 총 5가지 단계로 확인할 수 있습니다.

첫번째 단계는 센서 데이터 수집입니다. 센서 데이터 수집에는 센서 작동과 데이터 변환 과정이 있습니다.

센서 작동: 센서는 특정 환경 변수(온도, 습도, 소음)를 모니터링 하고, 이러한 변수의 변화를 전기적 신호로 변환합니다.

데이터 변환: 전기적 신호는 ADC를 통해 디지털 데이터로 변환됩니다. 이렇게 변환된 데이터는 처리 및 전송을 위해 임시로 저장됩니다.

두번째는 단계는 게이트웨이로의 데이터 전송입니다. 게이트웨이로의 데이터 전송에는 총 세 가지 과정이 있습니다.

데이터 포맷팅: 디지털 데이터는 전송을 위해 특정 포맷(JSON)으로 변환됩니다.

게이트웨이 연결: 센서는 무선 통신(LoRa, Wi-Fi, Bluetooth) 또는 유선 통신을 사용하여 게이트웨이에 연결됩니다.

데이터 전송: 연결된 게이트웨이로 센서 데이터가 실시간으로 전송됩니다.

세번째 단계는 게이트웨이에서의 데이터 처리입니다. 데이터 처리에는 데이터 버퍼링과 클라우드/중앙 서버 연결 과정이 있습니다.

데이터 버퍼링: 게이트웨이는 수신된 데이터를 임시로 저장하며, 필요에 따라 일부 데이터 처리를 수행할 수 있습니다.

클라우드/중앙 서버 연결: 게이트웨이는 클라우드 또는 중앙 서버와의 통신 경로를 설정합니다.

네번째 단계는 클라우드 또는 중앙 서버로의 데이터 전송 입니다. 중앙 서버로의 데이터 전송에는 데이터 암호화, 데이터 전송 과정이 있습니다.

데이터 암호화: 보안을 위해, 데이터는 SSL/TLS와 같은 프로토콜을 사용하여 암호화됩니다.

데이터 전송: 암호화된 데이터는 클라우드 또는 중앙 서버로 전송됩니다. 이 과정에서 MQTT, HTTP, CoAP 등의 통신 프로토콜이 사용될 수 있습니다.

다섯 번째 단계는 IoT디바이스로의 데이터 전달입니다. IoT 디바이스로의 데이터 전달에는 총 세 가지 과정이 있습니다.

서버에서의 데이터 처리: 클라우드 또는 중앙 서버는 수신된 데이터를 데이터베이스에 저장하거나 필요한 처리를 수행합니다.

IoT 디바이스 연결: 서버는 IoT 디바이스와의 연결을 설정하고, 해당 디바이스로 데이터를 전송합니다.

데이터 수신 및 활용: IoT 디바이스는 서버로부터 수신된 데이터를 활용하여 특정 기능을 수행하거나 사용자에게 정보를 제공합니다.

이러한 다섯단계를 거쳐 센서에서 측정된 데이터는 안전하게 처리되고 전송되며, IoT 디바이스에서 최종적으로 활용됩니다.

특히 다섯 번째 단계에서 Python 애플리케이션을 사용해서 클라우드 서버의 데이터를 가져옵니다. 예를 들면, MQTT 라이브러리(paho-mqtt) 또는 LoRa(Python-based LoRa) 라이브러리를 사용하여 클라우드 서버와 통신하고 데이터를 수신할 수 있습니다.

MQTT(Message Queuing Telemetry Transport): MQTT는 가볍고 효율적인 통신 프로토콜로, 소량의 데이터를 전송하기 위해 설계. 특히, 제한된 네트워크 대역폭과 높은 지연 시간을 가진 환경에서 동작

LoRa: LoRa는 저전력 장거리 통신을 위한 무선 모듈레이션 기술. 이 기술은 저전력 장치들이 광대역의 거리에서 통신하기 위해 설계

ADC(Analog to Digital Converter): ADC는 아날로그 신호를 디지털 신호로 변환하는 전자 장치

Python-based LoRa 라이브러리: Raspberry Pi와 같은 플랫폼에서 Python 언어를 사용하여 LoRa 모듈을 제어하기 위한 라이브러리

JSON(JavaScript Object Notion): JSON은 데이터 교환 형식 중 하나로, 읽기 쉽고 작성하기 쉬운 텍스트 포맷

게이트웨이(Gateway): 네트워크에서 통신을 중개하는 장치나 소프트웨어로, 서로 다른 네트워크 또는 프로토콜간의 통신을 가능하게 함. IoT에서는 센서 및 디바이스로부터 데이터를 수집하고 중앙 서버로 클라우드로 전송하는 역할

SSL/TLS(Secure Sockets Layer/Transport Layer Security): SSL과 TLS는 데이터 전송 중 보안을 제공하는 암호화 프로토콜

CoAP(Constrained Application Protocol): CoAP는 임베디드 시스템에 적합하게 설계된 인터넷 프로토콜

9. 예상 비용

텍스트, 폰트, 번호, 스크린샷이(가) 표시된 사진

자동 생성된 설명

서울시 내 존재하는 22개의 대교 중 철도만이 존재하는 당산철교와 한강철교를 제외한 20개의 대교(동작대교, 동호대교, 잠실대교, 청담대교, 가양대교, 성산대교, 양화대교, 서강대교, 월드컵대교, 원효대교, 마포대교, 한강대교, 반포대교(잠수교), 한남대교, 성수대교, 영동대교, 잠실대교, 올림픽대교, 천호대교, 광진교)를 토대로 Enclosure Hydrophone M14 설치 비용을 조사하였습니다. 센서 한개의 설치비용은 센서가격 41만원과 설치 비용 10만원, 유지보수 비용 5만원을 합쳐서 총 설치비용 56만원으로 측정하였습니다. 또한 한강 대교의 평균 길이인 1200미터에 12개의 센서를 수중에 골고루 설치하여 대교 전체의 수중 음향을 감지할 수 있도록 하였습니다. 대략적인 비용 측면이며, 다리 개당 약 700만원, 전체 총 1억 4천만원이면 자살 예방을 하는 것에 충분히 투자할 가치가 있다고 판단됩니다. 특히 게이트웨이를 사용하여 센서의 데이터를 수집할 경우, 수 킬로미터의 데이터를 하나의 게이트웨이가 수집할 수 있기 때문에 경제성 측면에서의 절약 효과가 있습니다.

# 3. 어플리케이션 개발 시현과 코드 리뷰

1. 로그인 화면

텍스트, 스크린샷, 폰트, 로고이(가) 표시된 사진

자동 생성된 설명

2. 메뉴 선택 화면

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

3. 대교 선택 화면

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

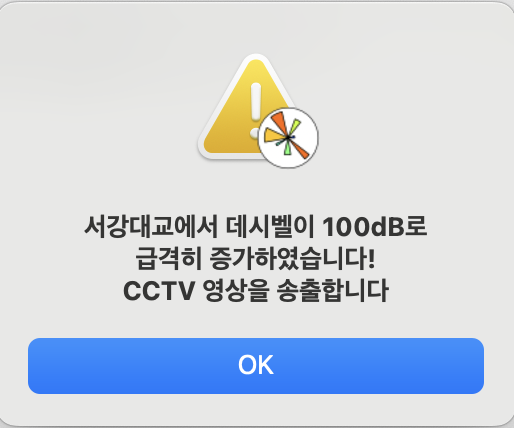
자동 생성된 설명

4. 청담대교 dB 모니터링 화면

텍스트, 스크린샷, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

5. 100dB 이상 감지 화면



6. CCTV 화면 송출

사람, 벽, 의류, 실내이(가) 표시된 사진

자동 생성된 설명

7.이상 감지 기록

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

8. 기록 선택 화면

텍스트, 스크린샷, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명

9. 코드

import tkinter as tk

from tkinter import messagebox, Toplevel

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import numpy as np

import time

import cv2

from PIL import Image, ImageTk

import threading

from matplotlib import font\_manager, rc

rc('font', family='AppleGothic')

plt.rcParams['axes.unicode\_minus'] = False

class BridgeSelection:

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

self.master = master

self.master.title("Bridge Selection")

self.bridges = ["성산대교","양화대교","서강대교","월드컵대교", "동호대교", "잠실대교", "청담대교", "가양대교",

"동작대교", "원효대교", "마포대교", "한강대교", "반포대교(잠수교)",

"한남대교", "성수대교", "영동대교", "잠실철교", "올림픽대교", "천호대교", "광진교"]

rows = 5

for idx, bridge in enumerate(self.bridges):

btn = tk.Button(self.master, text=bridge, command=lambda b=bridge: self.open\_decibel\_graph(b), width=20, height=2)

btn.grid(row=idx % rows, column=idx // rows, pady=6, padx=6)

def open\_decibel\_graph(self, bridge):

top = Toplevel(self.master)

app = DecibelApp(top, bridge)

top.title(bridge + " Decibel Monitoring")

class MenuScreen:

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

self.master = master

self.master.title("Menu Selection")

# 버튼 크기와 여백 조절

self.btn\_decibel = tk.Button(self.master, text="현재 한강의 데시벨", command=self.open\_decibel\_monitoring, width=30, height=2)

self.btn\_decibel.pack(pady=20, padx=20) # padx를 추가하여 좌우 여백도 조절

self.btn\_records = tk.Button(self.master, text="이상 감지 기록", command=self.open\_anomaly\_records, width=30, height=2)

self.btn\_records.pack(pady=20, padx=20) # padx를 추가하여 좌우 여백도 조절

def open\_decibel\_monitoring(self):

top = Toplevel(self.master)

app = BridgeSelection(top)

def open\_anomaly\_records(self):

top = Toplevel(self.master)

app = AnomalyRecordsApp(top)

class AnomalyRecordsApp:

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

self.master = master

self.master.title("이상 감지 기록")

records = [

"반포대교 12번 : 2023년 11월 3일 3시 25분 30초",

"한강대교 7번 : 2023년 11월 3일 5시 22분 17초",

"한남대교 2번 : 2023년 11월 3일 8시 23분 22초",

"마포대교 4번 : 2023년 11월 3일 11시 25분 09초",

"동작대교 1번 : 2023년 11월 3일 13시 20분 00초",

"동호대교 11번 : 2023년 11월 3일 17시 21분 19초",

"성산대교 9번 : 2023년 11월 3일 18시 01분 23초",

"영동대교 3번 : 2023년 11월 3일 19시 03분 32초",

"성수대교 1번 : 2023년 11월 3일 21시 48분 01초",

"원효대교 12번 : 2023년 11월 3일 23시 12분 08초"

]

self.buttons = [] # 버튼을 저장할 리스트를 생성

button\_width = 60 # 버튼의 너비를 지정

button\_height = 3 # 버튼의 높이를 지정

for row, record in enumerate(records):

# lambda 함수에서는 'r=record'를 사용하여 현재 레코드에 대한 참조를 저장

btn = tk.Button(self.master, text=record,

width=button\_width, height=button\_height,

command=lambda r=record: self.show\_graph\_for\_record(r))

btn.grid(row=row, column=0, padx=10, pady=2) # padx와 pady로 버튼 사이의 여백을 설정

self.buttons.append(btn) # 버튼을 리스트에 추가

self.master.grid\_columnconfigure(0, weight=1) # 열 너비가 내용에 따라 조정되도록 설정

def extract\_hour\_from\_record(self, record):

# 기록에서 시간 정보를 추출하는 함수

# 예: "2023년 11월 3일 3시 25분 30초" -> 3 (시간만 반환)

time\_segment = record.split()[-3]

return int(time\_segment.replace('시', ''))

def generate\_anomaly\_data(self, anomaly\_hour):

x = np.linspace(0, 24, 25) # 24시간 데이터 생성

y = np.random.uniform(50, 60, len(x)) # 대부분의 데이터는 50-60 dB 범위

return x, y

def show\_graph\_for\_record(self, record):

top = Toplevel(self.master)

top.title(f"{record}")

anomaly\_hour = self.extract\_hour\_from\_record(record)

x, y = self.generate\_anomaly\_data(anomaly\_hour)

fig, ax = plt.subplots()

ax.plot(x, y, '-o')

ax.set\_title(f"{record}")

ax.set\_xlabel('Time (Hours)')

ax.set\_ylabel('Decibel')

ax.set\_ylim(0, 110)

ax.set\_xlim(0, 24)

canvas = FigureCanvasTkAgg(fig, master=top)

canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

class LoginScreen:

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

self.master = master

self.master.title("Login")

self.logo\_image = ImageTk.PhotoImage(file="/Users/Documents/서울시 AIoT/photo.png")

self.logo\_label = tk.Label(self.master, image=self.logo\_image)

self.logo\_label.grid(row=0, column=0, columnspan=2)

self.label\_username = tk.Label(self.master, text="Username")

self.label\_password = tk.Label(self.master, text="Password")

self.entry\_username = tk.Entry(self.master)

self.entry\_password = tk.Entry(self.master, show="\*")

self.label\_username.grid(row=1, column=0)

self.label\_password.grid(row=2, column=0)

self.entry\_username.grid(row=1, column=1)

self.entry\_password.grid(row=2, column=1)

self.login\_button = tk.Button(self.master, text="Login", command=self.check\_credentials)

self.login\_button.grid(row=3, column=0, columnspan=2)

icon\_image = ImageTk.PhotoImage(file="/Users/Documents/서울시 AIoT/png")

self.master.iconphoto(False, icon\_image)

def check\_credentials(self):

username = self.entry\_username.get()

password = self.entry\_password.get()

# username: 서빙고, password: 119

if username == '서빙고' and password == '119':

self.master.destroy() # 현재 로그인 창 닫기

root = tk.Tk()

app = MenuScreen(root)

root.mainloop()

else:

messagebox.showerror("Error", "Wrong username or password")

class DecibelApp:

def \_\_init\_\_(self, master, bridge\_name):

self.master = master

self.bridge\_name = bridge\_name

# 각 다리에 따른 경보 발생 시점과 랜덤 값 범위

self.bridge\_alert\_times = {

"동작대교": 10, # 10초

"동호대교": 20, # 20초

"잠실대교": 13,

"청담대교": 2,

"가양대교": 8,

"성산대교": 100,

"양화대교": 100,

"서강대교": 25,

"월드컵대교": 100,

"원효대교": 17,

"반포대교(잠수교)": 30,

"마포대교": 30,

"한강대교": 12,

"성수대교": 43,

"영동대교": 31,

"잠실철교": 39,

"올림픽대교": 12,

"천호대교": 15,

"광진교": 22

}

self.bridge\_db\_ranges = {

"동작대교": (60, 65),

"동호대교": (55, 60),

"잠실대교": (60, 63),

"청담대교": (52, 61),

"가양대교": (50, 61),

"성산대교": (54, 63),

"양화대교": (53, 62),

"서강대교": (55, 60),

"월드컵대교": (44, 60),

"원효대교": (51, 60),

"마포대교": (40, 60),

"한강대교": (55, 58),

"성수대교": (57, 60),

"영동대교": (53, 60),

"잠실철교": (55, 60),

"올림픽대교": (55, 57),

"천호대교": (52, 55),

"광진교": (57, 60)

}

self.master.title("Decibel Monitoring")

self.fig, self.ax = plt.subplots(figsize=(7, 5))

self.canvas = FigureCanvasTkAgg(self.fig, master=self.master)

self.canvas.get\_tk\_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

self.alert\_triggered = False

self.reset\_data()

self.update\_graph()

def reset\_data(self):

self.start\_time = time.time()

self.decibel\_data = []

self.time\_data = []

self.alert\_triggered = False

def generate\_decibel(self):

elapsed\_time = time.time() - self.start\_time

alert\_time = self.bridge\_alert\_times.get(self.bridge\_name, 30) # 기본값 30초

if elapsed\_time > alert\_time and len(self.time\_data) > 0 and self.time\_data[-1] < alert\_time:

return 100

else:

# 다리 이름에 따른 데시벨 범위 참조

min\_db, max\_db = self.bridge\_db\_ranges.get(self.bridge\_name, (50, 65))

return np.random.uniform(min\_db, max\_db)

def play\_video(self, video\_path):

cap = cv2.VideoCapture(video\_path)

top = Toplevel(self.master)

label = tk.Label(top)

label.pack()

frames\_per\_second = cap.get(cv2.CAP\_PROP\_FPS)

delay = int(1000 / frames\_per\_second)

def update\_video():

ret, frame = cap.read()

if not ret:

cap.release()

top.destroy()

return

# 영상 크기 조절

frame = cv2.resize(frame, (int(frame.shape[1]\*1.1), int(frame.shape[0]\*0.5)))

image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

image = Image.fromarray(image)

photo = ImageTk.PhotoImage(image=image)

label.config(image=photo)

label.image = photo

top.after(delay, update\_video)

update\_video()

def show\_alert(self):

messagebox.showwarning("경보", f"{self.bridge\_name}에서 데시벨이 100dB로 급격히 증가하였습니다! CCTV 영상을 송출합니다")

def update\_graph\_color(self, color='b'):

self.ax.clear()

self.ax.plot(self.time\_data, self.decibel\_data, '-o', color=color)

self.ax.set\_xlim(0, 60)

self.ax.set\_ylim(0, 110)

self.ax.set\_xlabel('Time (S)')

self.ax.set\_ylabel('Decibel')

self.canvas.draw()

def update\_graph(self):

decibel = self.generate\_decibel()

elapsed\_time = time.time() - self.start\_time

self.decibel\_data.append(decibel)

self.time\_data.append(elapsed\_time)

if decibel == 100 and not self.alert\_triggered:

self.alert\_triggered = True

# 경보 메시지와 동영상을 별도의 스레드에서 플레이

def alert\_and\_play\_video():

self.show\_alert()

self.play\_video("/Users/Documents/서울시 AIoT/video.mp4")

threading.Thread(target=alert\_and\_play\_video).start()

else:

self.update\_graph\_color()

if elapsed\_time >= 60:

self.reset\_data()

self.master.after(1000, self.update\_graph)

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

login\_screen = LoginScreen(root)

root.mainloop()