

EAS SPS eNose

By Kelompok 3

Aldi Rochmad Romdoni 2042241050

Adrianta Fajar Surapramana 2042241052

Hanif Raditya Pratama 2042241065

Penjelasan Proyek

electronic nose (eNose) berperan dalam mendeteksi dan mengklasifikasikan aroma serta komposisi gas dari berbagai sampel uji. Untuk memahami integrasi mendalam antara sistem akuisisi data, pemrosesan sinyal, dan visualisasi hasil secara real-time, proyek ini mengembangkan sebuah aplikasi desktop interaktif yang mampu menerima dan memproses data dari multiple sensor gas yang terhubung dengan Arduino Uno R4 WiFi.

Pada kelompok kami, sampel uji yang diambil adalah 4-5 jenis daun, yaitu daun kemangi, seledri, jeruk, kari, dan pandan

Penjelasan Proyek

Aplikasi ini dirancang dengan arsitektur yang memisahkan tugas antara lapisan backend dan frontend untuk memastikan skalabilitas, maintainability, dan performa optimal. Stack teknologi yang digunakan mencakup:

- Rust sebagai backend untuk pemrosesan sinyal, manajemen data sensor, dan logika komputasi yang demand tinggi terhadap performa
- Qt Python (PySide6/PyQt6) sebagai framework GUI untuk menghadirkan antarmuka pengguna yang responsif dan intuitif
- Edge Impulse sebagai platform untuk pengembangan dan deployment model machine learning yang melakukan analisis dan klasifikasi data sensor secara real-time

Model Rust (backend)

main.rs

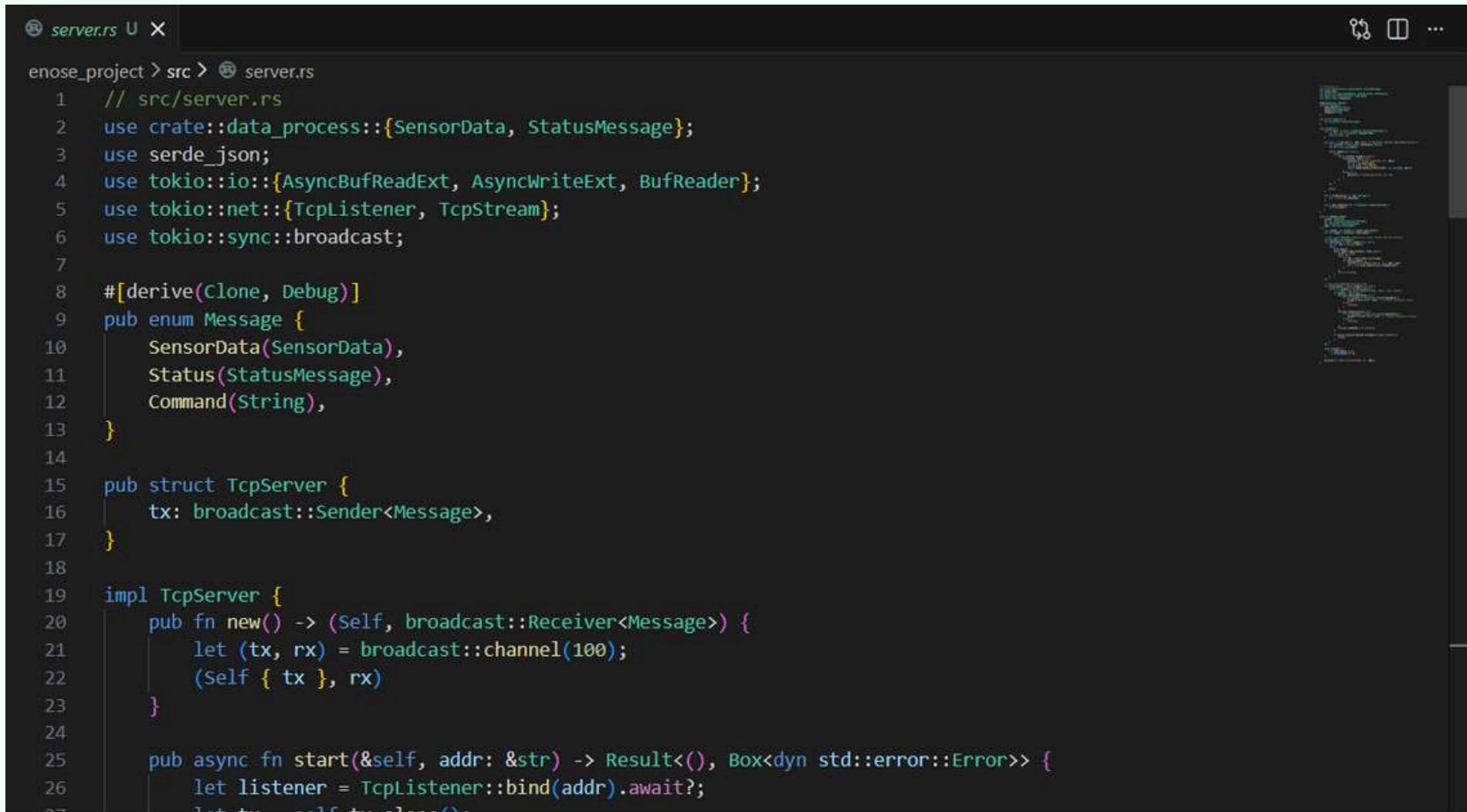
- Menentukan mode operasi: dummy atau normal (Arduino via Wi-Fi).
- Inisialisasi:
 - TcpServer
 - Receiver Arduino
 - Loop utama async untuk menangani aliran data.
- Menjembatani alur data antara Arduino → DataProcessor → Server → GUI.

```
main.rs U X
enose_project > src > main.rs
12  async fn main() -> Result<(), Box<dyn std::error::Error>> {
31      ok(())
32  }
34
35  // =====
36  // MODE 1: DUMMY - Structured Data with Manual Sample Type Change
37  // =====
38  async fn run_dummy_mode() -> Result<(), Box<dyn std::error::Error>> {
39      let (server, _rx) = TcpServer::new();
40      server.start("127.0.0.1:8080").await?;
41
42      println!("✓ TCP Server listening on 127.0.0.1:8080");
43      println!("✓ Structured dummy data generation");
44      println!("⚠ Change sample type manually in main.rs");
45      println!("✓ Press Ctrl+C to stop\n");
46      println!("{:-<70}\n", "");
47
48  // -----
49  // GANTI SAMPLE TYPE DI SINI:
50  let current_sample = "Daun Pandan"; // ← UBAH INI UNTUK GANTI SAMPLE
51  // Options: "Daun Kari", "Daun Kemangi", "Daun Jeruk", "Daun Seledri"
52  //
53
54  let mut counter = 0_i32;
55  let mut m1_step = 0_usize;
```

Model Rust (backend)

server.rs

- Mengimplementasikan server TCP asinkron untuk koneksi ke GUI.
- Menangani beberapa klien sekaligus (multiclient).
- Melakukan broadcast pesan berformat Message:
 - SensorData
 - Status
 - Command
- Menyediakan channel internal untuk menerima data dari modul lain.



The screenshot shows a code editor window with the file 'server.rs' open. The code implements a TCP server using the tokio library. It defines a message enum with three variants: SensorData, Status, and Command. The TcpServer struct contains a broadcast channel and a receiver. The start function binds a listener to an address and starts accepting connections.

```
// server.rs
use crate::data_process::{SensorData, StatusMessage};
use serde_json;
use tokio::io::{AsyncBufReadExt, AsyncWriteExt, BufReader};
use tokio::net::{TcpListener, TcpStream};
use tokio::sync::broadcast;

#[derive(Clone, Debug)]
pub enum Message {
    SensorData(SensorData),
    Status(StatusMessage),
    Command(String),
}

pub struct TcpServer {
    tx: broadcast::Sender<Message>,
}

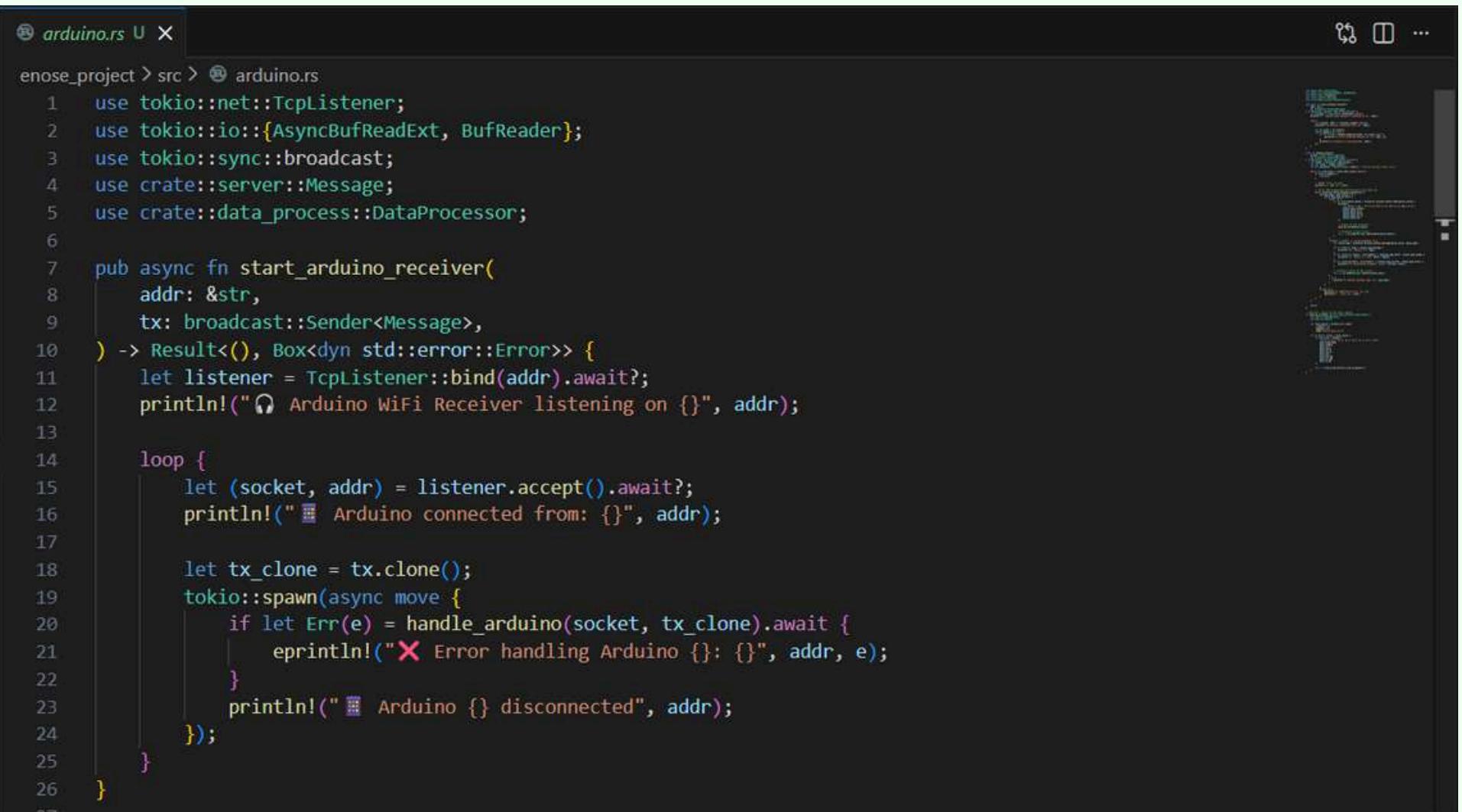
impl TcpServer {
    pub fn new() -> (Self, broadcast::Receiver<Message>) {
        let (tx, rx) = broadcast::channel(100);
        (Self { tx }, rx)
    }

    pub async fn start(&self, addr: &str) -> Result<(), Box<dyn std::error::Error>> {
        let listener = TcpListener::bind(addr).await?;
        let mut tasks = Vec::new();
        for stream in listener.incoming() {
            tasks.push(tokio::spawn(async move {
                let mut stream = stream?;
                let mut reader = BufReader::new(stream);
                let mut writer = stream;
                loop {
                    let mut buffer = [0; 1024];
                    let n = reader.read(&mut buffer).await?;
                    if n == 0 {
                        break;
                    }
                    writer.write_all(&buffer[0..n]).await?;
                }
            }));
        }
        tasks.retain(|task| task.is_finished());
        if !tasks.is_empty() {
            futures::join_all(tasks).await;
        }
        Ok(())
    }
}
```

Model Rust (backend)

arduino.rs

- Menerima data JSON dari Arduino melalui koneksi TCP/Wi-Fi.
- Memvalidasi dan meneruskan data mentah ke modul pemrosesan (DataProcessor).
- Mengirim hasil pemrosesan ke TcpServer untuk diteruskan ke GUI.
- Menangani reconnect, timeout, dan deteksi error data sederhana.



The screenshot shows a code editor window titled "arduino.rs" with the following Rust code:

```
enoze_project > src > @ arduino.rs
1 use tokio::net::TcpListener;
2 use tokio::io::{AsyncBufReadExt, BufReader};
3 use tokio::sync::broadcast;
4 use crate::server::Message;
5 use crate::data_process::DataProcessor;
6
7 pub async fn start_arduino_receiver(
8     addr: &str,
9     tx: broadcast::Sender<Message>,
10 ) -> Result<(), Box<dyn std::error::Error>> {
11     let listener = TcpListener::bind(addr).await?;
12     println!("⌚ Arduino WiFi Receiver listening on {}", addr);
13
14     loop {
15         let (socket, addr) = listener.accept().await?;
16         println!("🌐 Arduino connected from: {}", addr);
17
18         let tx_clone = tx.clone();
19         tokio::spawn(async move {
20             if let Err(e) = handle_arduino(socket, tx_clone).await {
21                 eprintln!("✖ Error handling Arduino {}: {}", addr, e);
22             }
23             println!("🌐 Arduino {} disconnected", addr);
24         });
25     }
26 }
27 }
```

Model Rust (backend)

data_process.rs

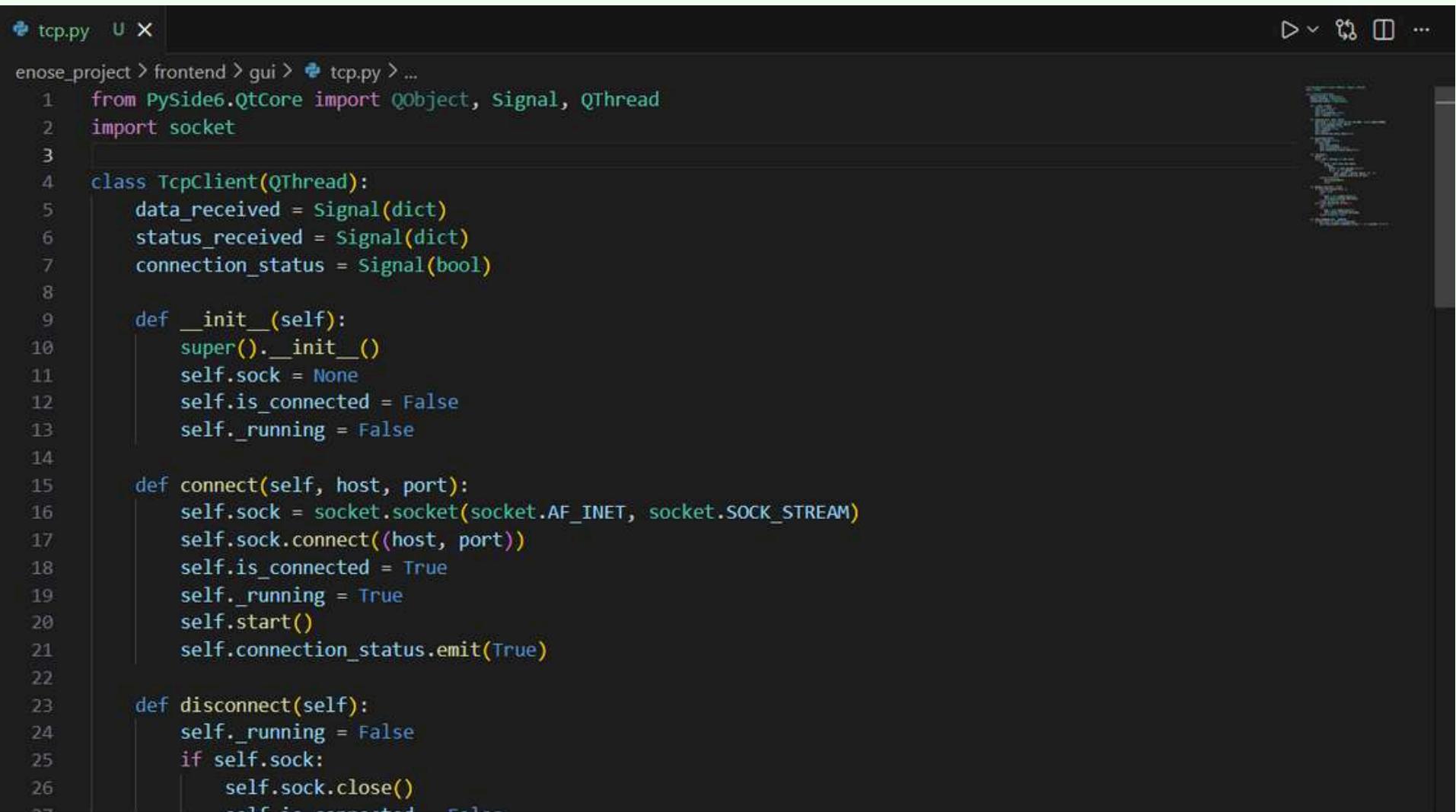
- Mendefinisikan struktur:
 - SensorData
 - StatusMessage
 - Tipe-tipe enum untuk pesan.
- Bertugas melakukan:
 - Parsing JSON
 - Normalisasi nama sampel
 - Smoothing (moving average)
 - Buffering data sensor untuk stabilitas tampilan di GUI.
- Output akhir dikemas dalam bentuk Message yang siap dibroadcast ke klien.

```
data_process.rs U X
enoze_project > src > data_process.rs
1 use serde::Deserialize, Serialize;
2 use serde_json::Value;
3 use std::collections::VecDeque;
4
5 // -----
6 // DATA STRUCTURES
7 // -----
8
9 #[derive(Debug, Clone, Serialize, Deserialize)]
10 pub struct SensorData {
11     #[serde(alias = "timestamp")]
12     pub timestamp: u64,
13     pub sample: String,
14     pub co_m: f32,
15     pub eth_m: f32,
16     pub voc_m: f32,
17     pub no2: f32,
18     pub eth_gm: f32,
19     pub voc_gm: f32,
20     pub co_gm: f32,
21 }
22
23 #[derive(Debug, Clone, Serialize, Deserialize)]
24 pub struct StatusMessage {
25     #[serde(rename = "msg_type")]
26     pub msg_type: String,
27 }
```

Model Python (frontend)

tcp.py — TcpClient

- Mengelola koneksi TCP ke backend (Rust).
- Menerima stream data mentah, memisahkan paket berdasarkan prefix:
 - DATA: → data sensor
 - STATUS: → status sistem / notifikasi
- Melakukan parsing dasar dan emit sinyal ke GUI (Qt/PySide).

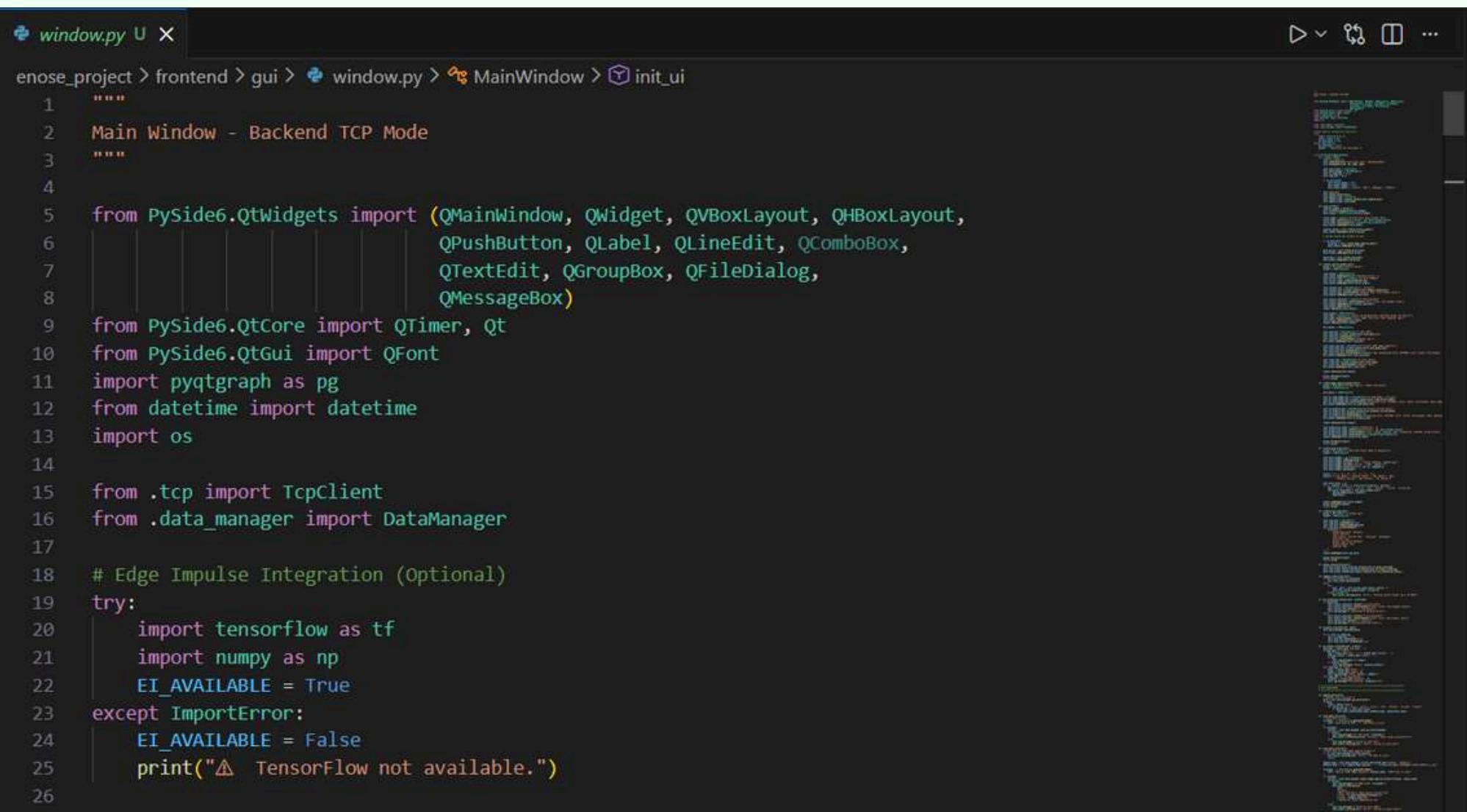


```
tcp.py  U X
eonoze-project > frontend > gui > tcp.py > ...
1  from PySide6.QtCore import QObject, Signal, QThread
2  import socket
3
4  class TcpClient(QThread):
5      data_received = Signal(dict)
6      status_received = Signal(dict)
7      connection_status = Signal(bool)
8
9  def __init__(self):
10     super().__init__()
11     self.sock = None
12     self.is_connected = False
13     self._running = False
14
15    def connect(self, host, port):
16        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17        self.sock.connect((host, port))
18        self.is_connected = True
19        self._running = True
20        self.start()
21        self.connection_status.emit(True)
22
23    def disconnect(self):
24        self._running = False
25        if self.sock:
26            self.sock.close()
27            if self.is_connected: self.is_connected = False
```

Model Python (frontend)

window.py — MainWindow

- Mengatur GUI utama:
 - kontrol koneksi
 - tombol simpan
 - panel log
 - indikator status
- Menghubungkan sinyal dari TcpClient
→ DataManager → plot real-time.
- Mendukung integrasi opsional panel model TFLite untuk inferensi lokal.



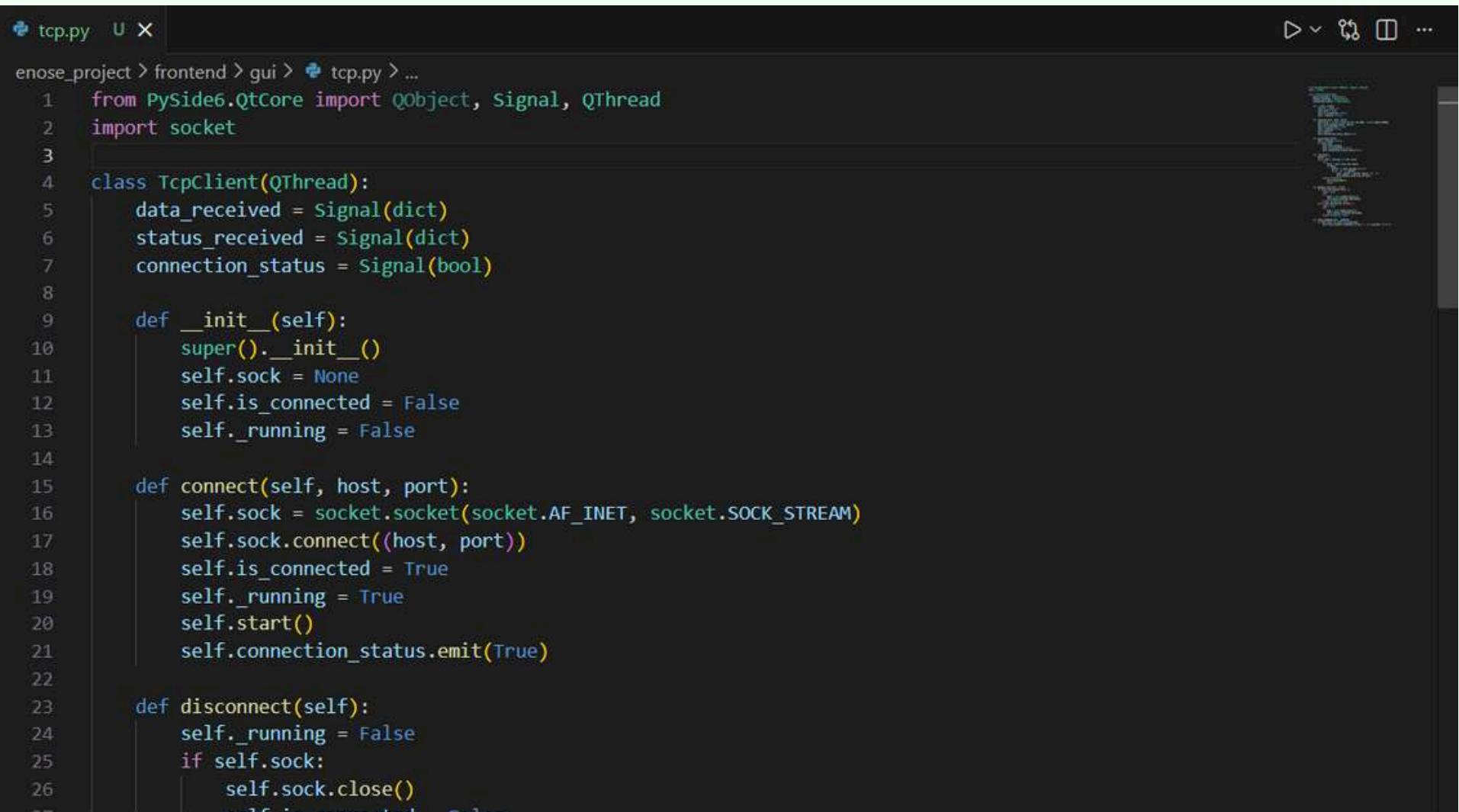
The screenshot shows a code editor window with the file 'window.py' open. The code is written in Python and defines a class 'MainWindow'. The code imports various QtWidgets and QtCore modules, along with pyqtgraph, datetime, os, TcpClient, and DataManager. It also includes a section for Edge Impulse integration using TensorFlow and NumPy, with a check for their availability. The code uses standard Python syntax with indentation and comments.

```
 1 """
 2 Main Window - Backend TCP Mode
 3 """
 4
 5 from PySide6.QtWidgets import (QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
 6                                 QPushButton, QLabel, QLineEdit, QComboBox,
 7                                 QTextEdit, QGroupBox, QFileDialog,
 8                                 QMessageBox)
 9 from PySide6.QtCore import QTimer, Qt
10 from PySide6.QtGui import QFont
11 import pyqtgraph as pg
12 from datetime import datetime
13 import os
14
15 from .tcp import TcpClient
16 from .data_manager import DataManager
17
18 # Edge Impulse Integration (Optional)
19 try:
20     import tensorflow as tf
21     import numpy as np
22     EI_AVAILABLE = True
23 except ImportError:
24     EI_AVAILABLE = False
25     print("⚠ TensorFlow not available.")
```

Model Python (frontend)

dtcp.py — TcpClient

- Mengelola koneksi TCP ke backend (Rust).
- Menerima stream data mentah, memisahkan paket berdasarkan prefix:
 - DATA: → data sensor
 - STATUS: → status sistem / notifikasi
- Melakukan parsing dasar dan emit sinyal ke GUI (Qt/PySide).



```
tcp.py  U X
enoise-project > frontend > gui > tcp.py > ...
1  from PySide6.QtCore import QObject, Signal, QThread
2  import socket
3
4  class TcpClient(QThread):
5      data_received = Signal(dict)
6      status_received = Signal(dict)
7      connection_status = Signal(bool)
8
9      def __init__(self):
10         super().__init__()
11         self.sock = None
12         self.is_connected = False
13         self._running = False
14
15     def connect(self, host, port):
16         self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17         self.sock.connect((host, port))
18         self.is_connected = True
19         self._running = True
20         self.start()
21         self.connection_status.emit(True)
22
23     def disconnect(self):
24         self._running = False
25         if self.sock:
26             self.sock.close()
27             self.is_connected = False
```

Sampel



Daun Pandan



Daun Jeruk

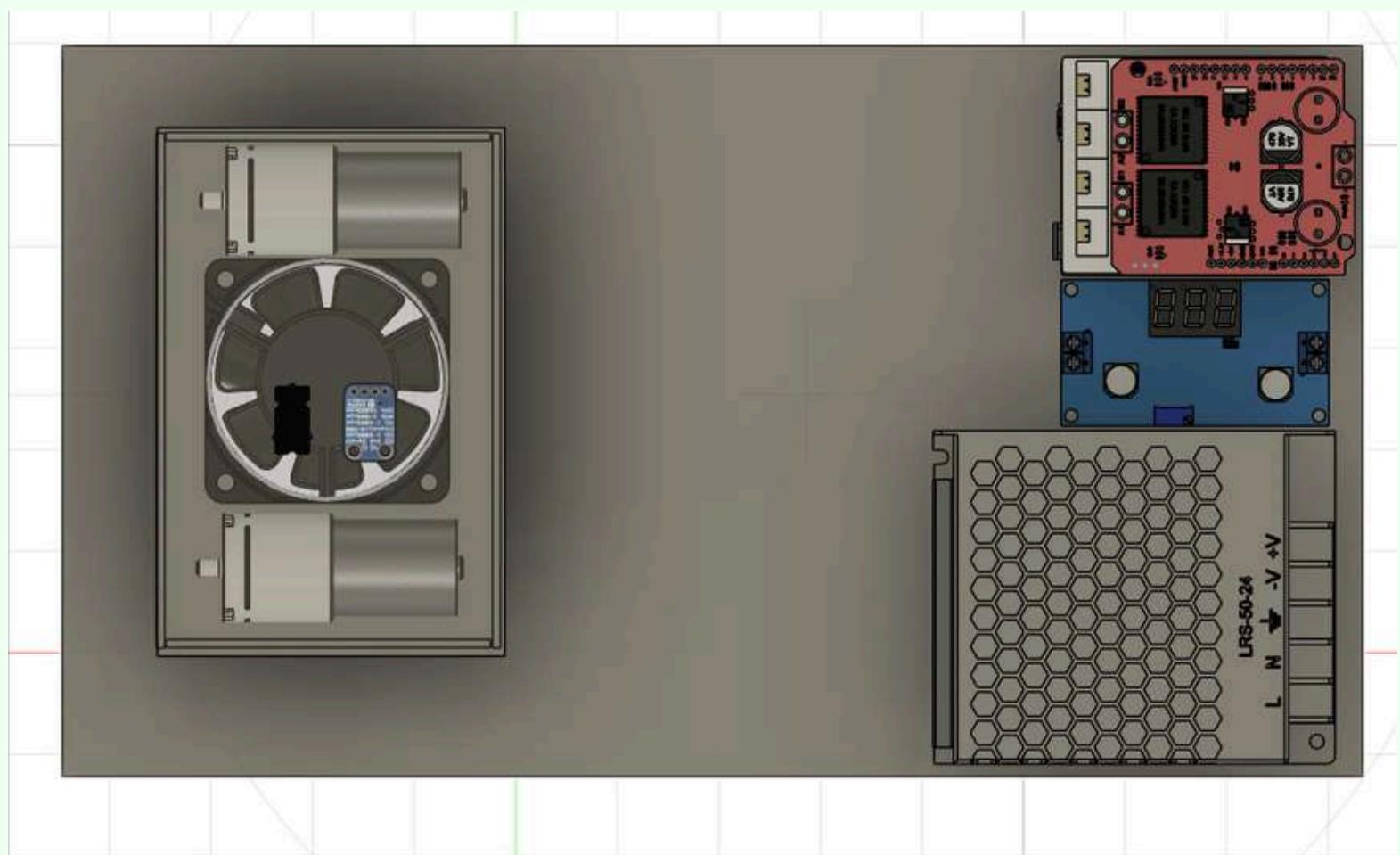
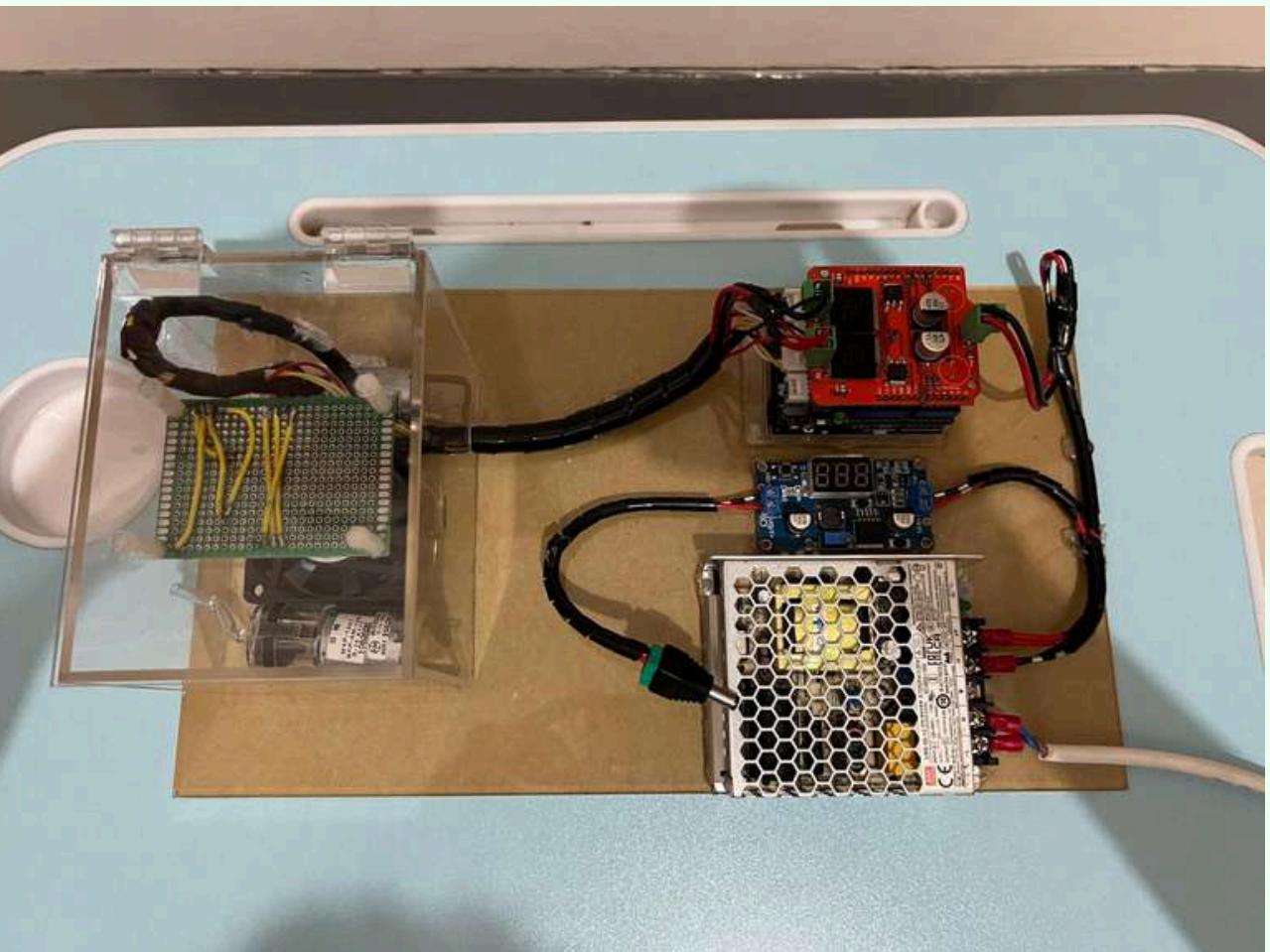


Daun Kari

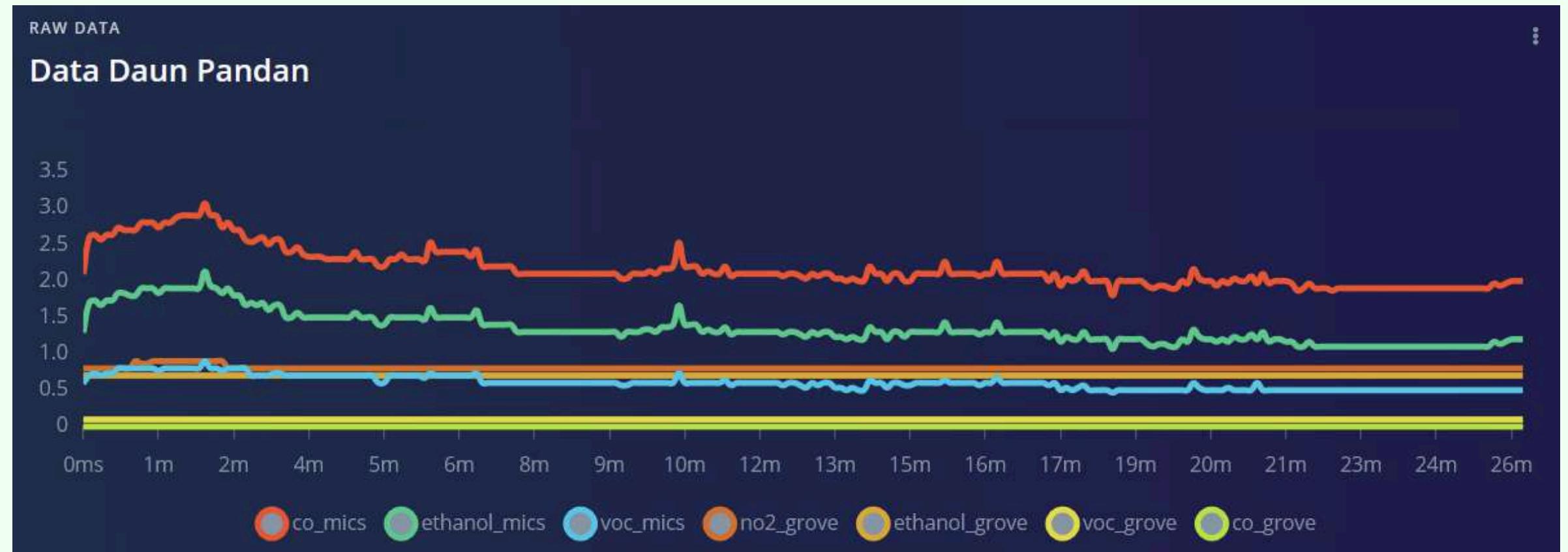


Daun Kemangi

Dokumentasi Alat

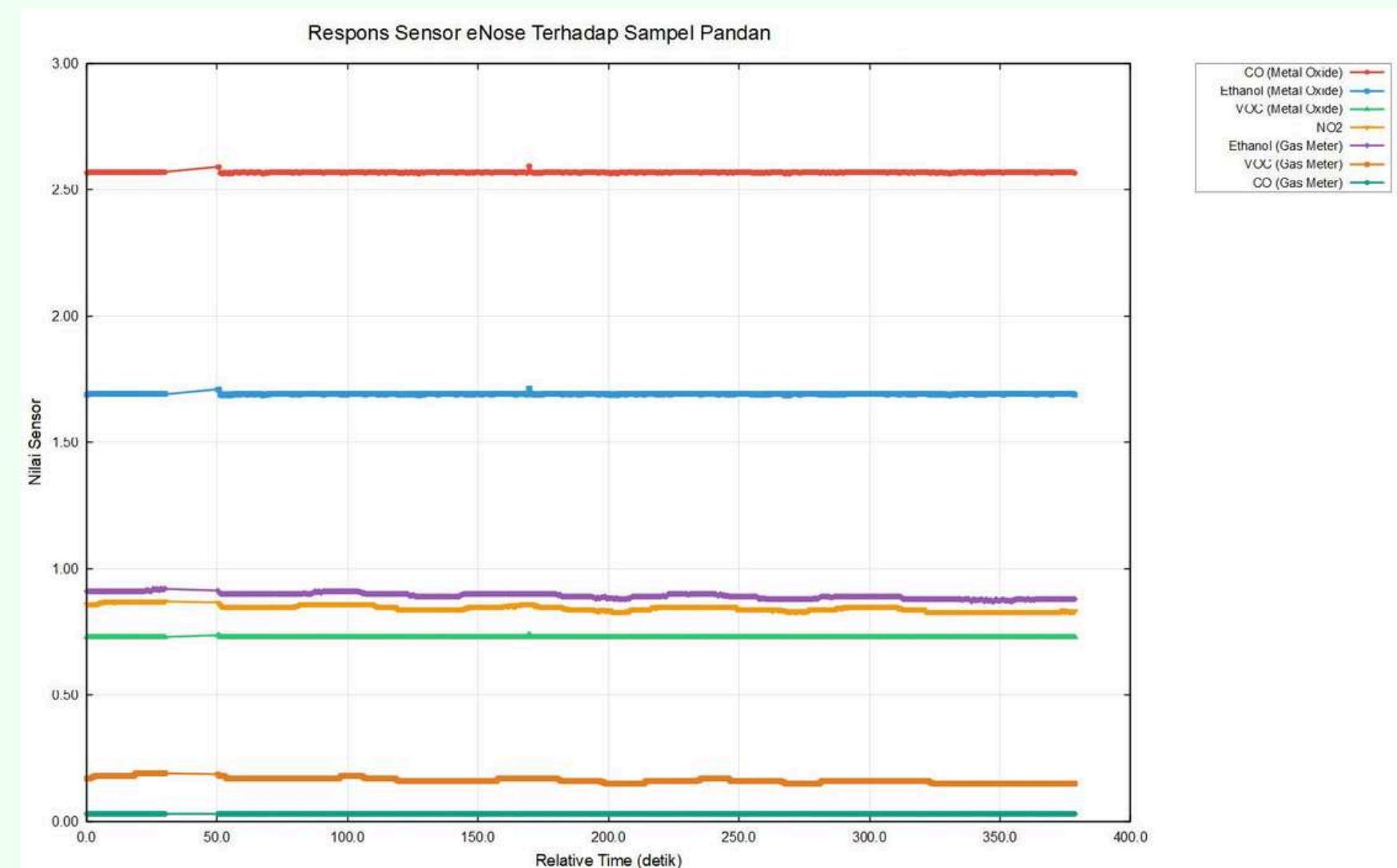


Hasil Data



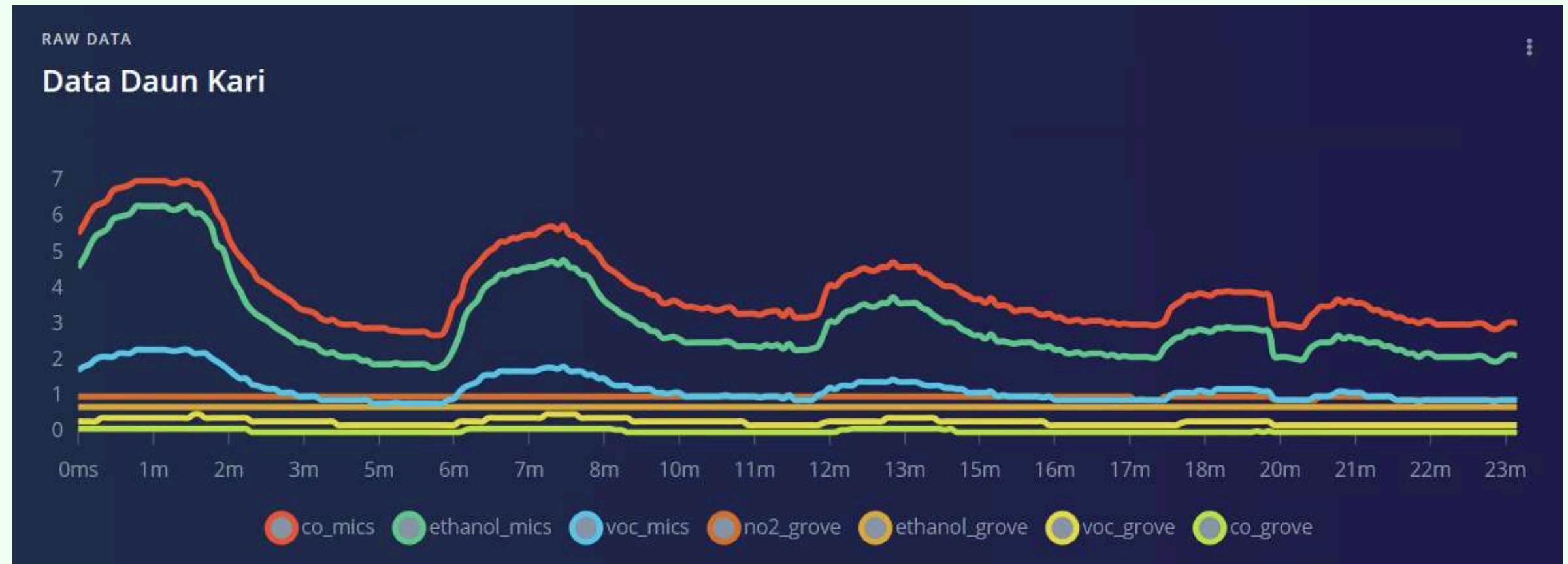
Daun Pandan Pompa 1

Hasil Data



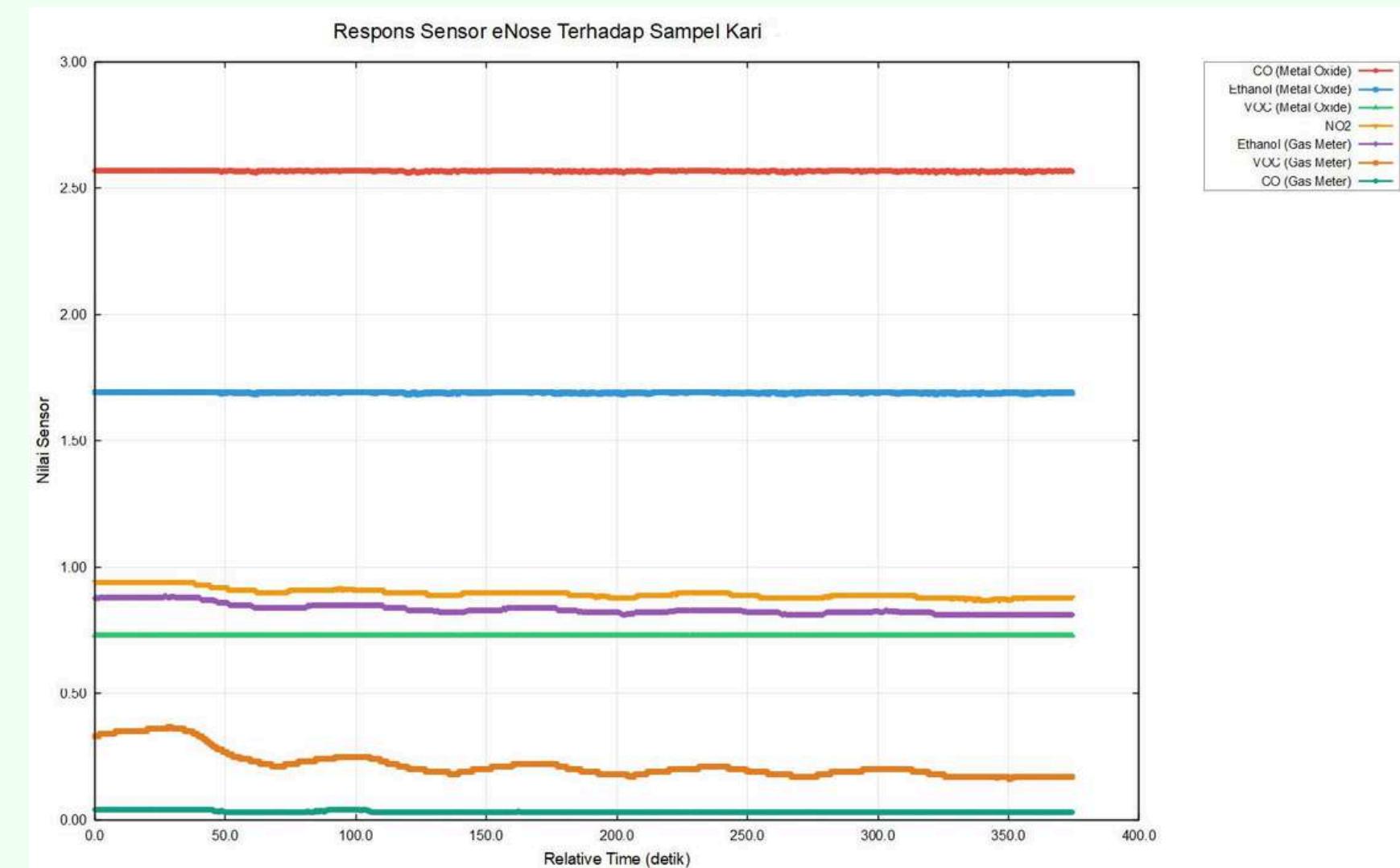
Daun Pandan Pompa 2

Hasil Data



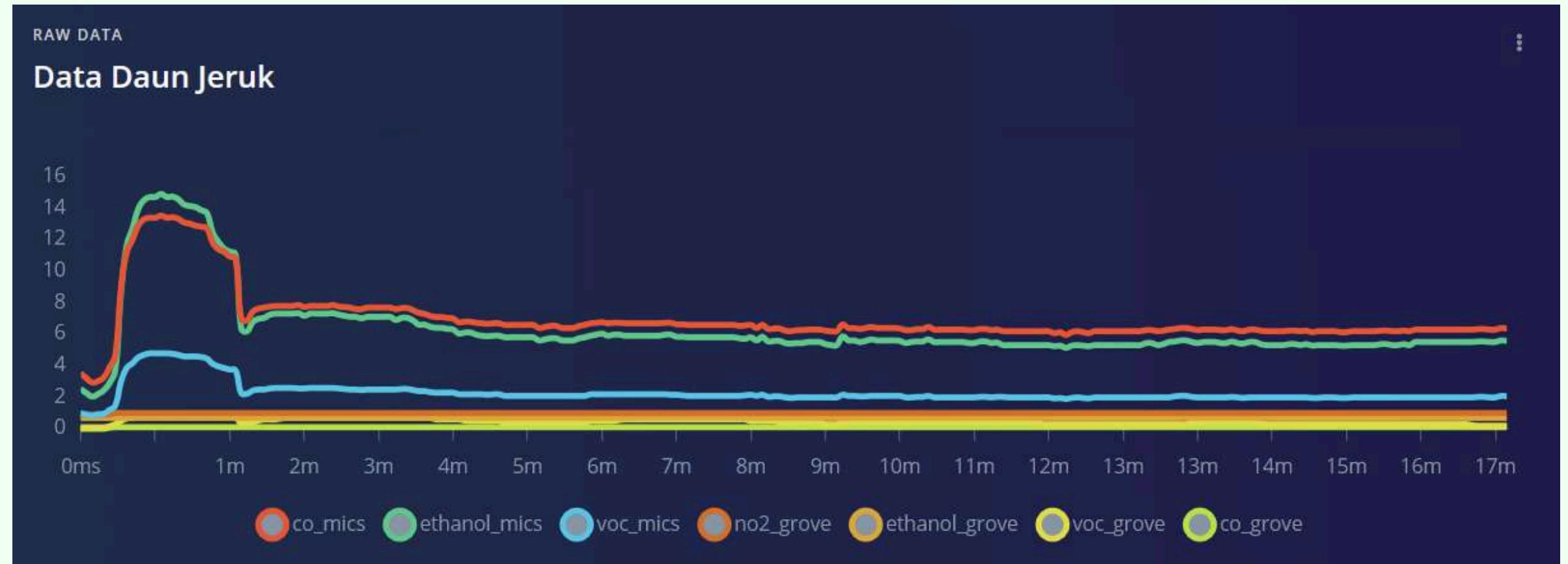
Daun Kari Pompa 1

Hasil Data



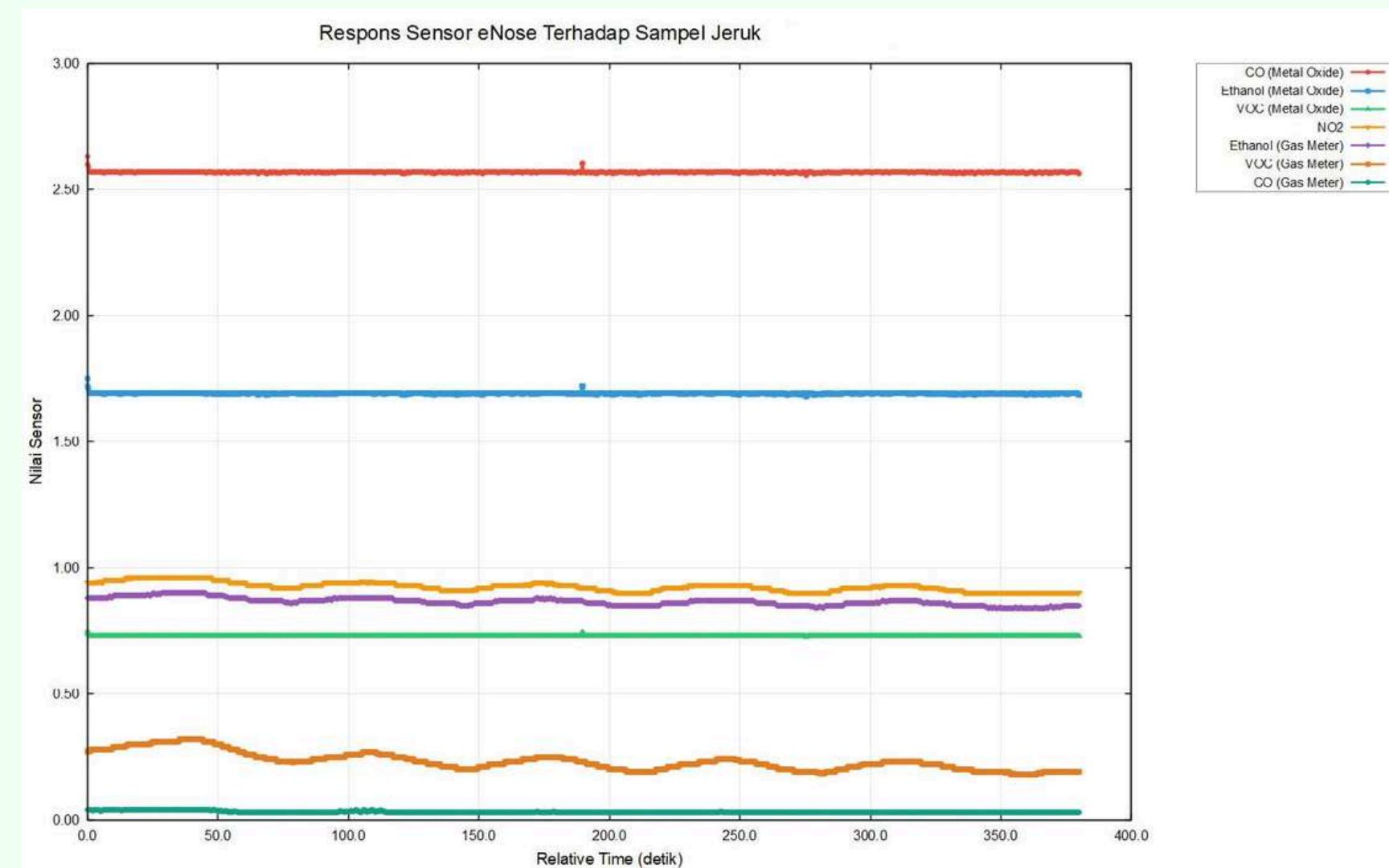
Daun Kari Pompa 2

Hasil Data



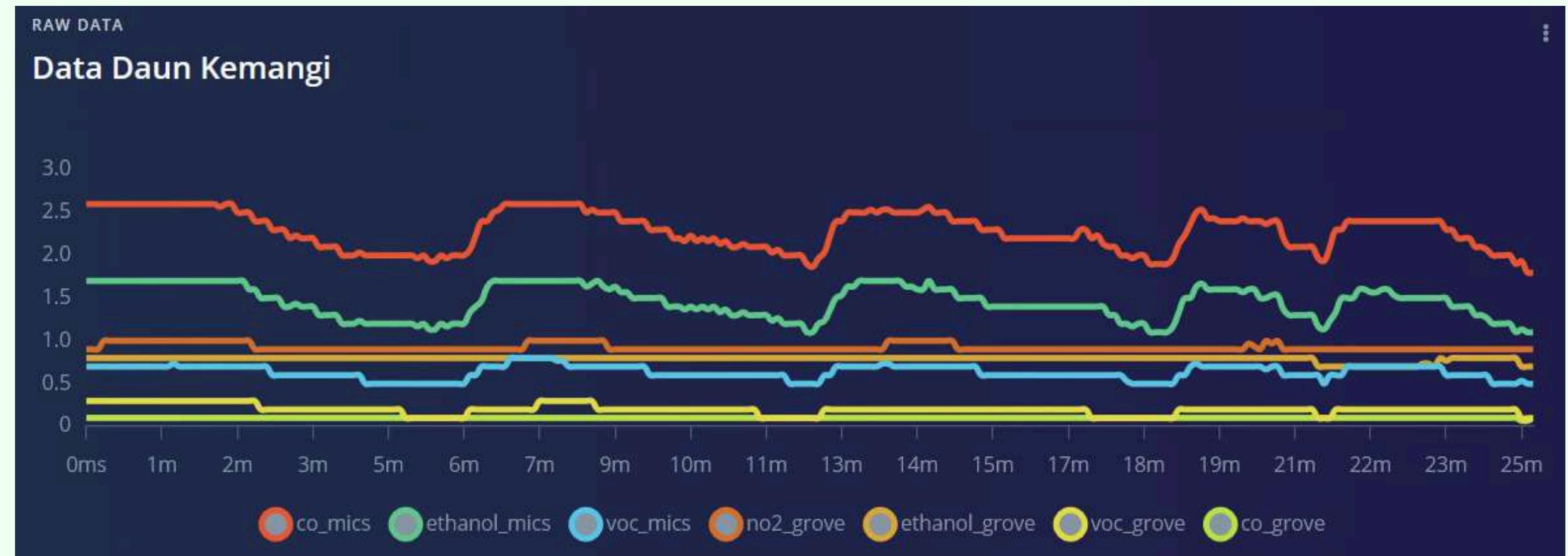
Daun Jeruk Pompa 1

Hasil Data



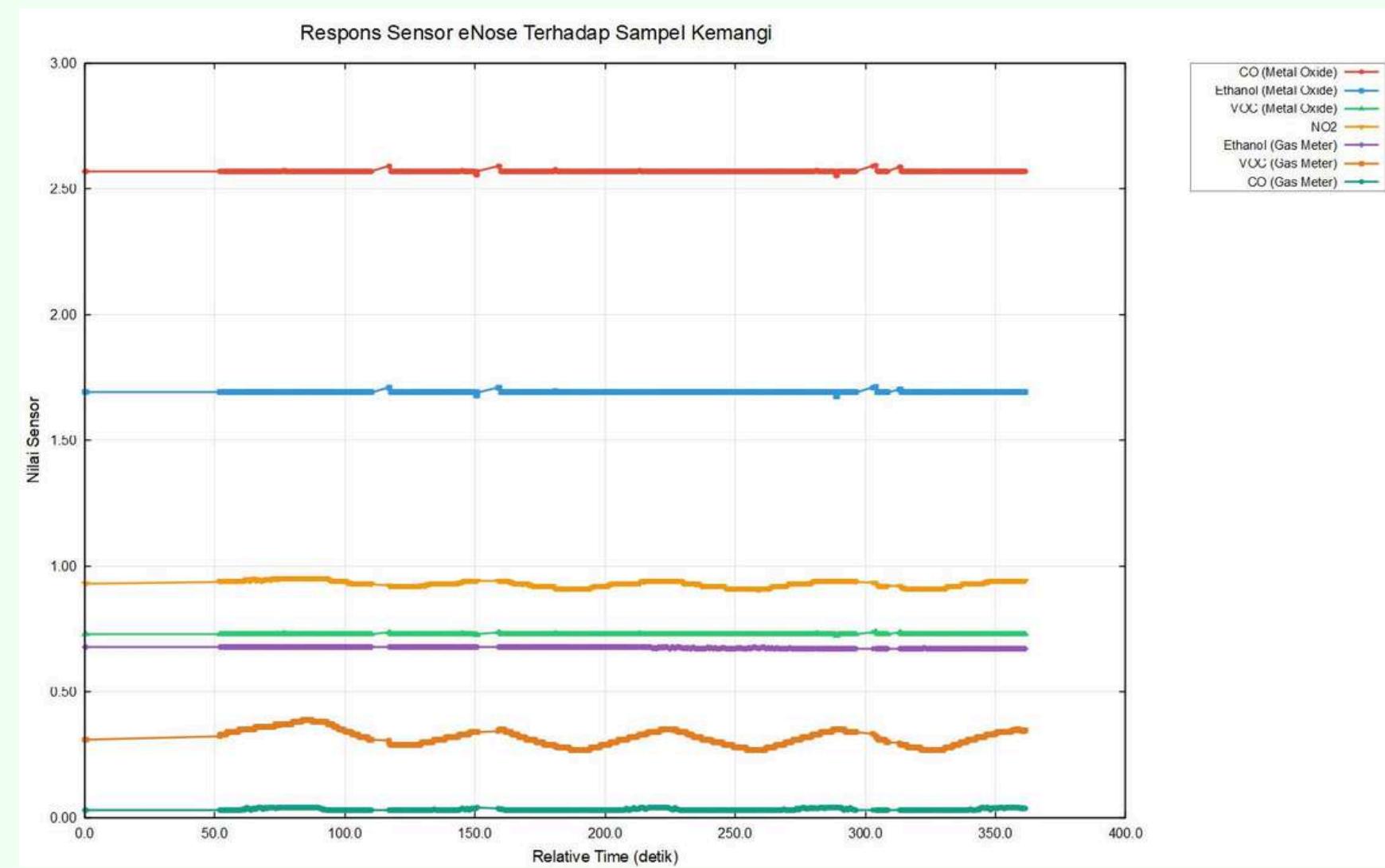
Daun Jeruk Pompa 2

Hasil Data



Daun Kemangi Pompa 1

Hasil Data



Daun Kemangi Pompa 2

Kesimpulan

Berdasarkan hasil percobaan, sistem eNose mampu mendeteksi gas yang dikeluarkan oleh keempat sampel daun. Namun, grafik keluaran menunjukkan bahwa penggunaan satu pompa dengan durasi pengukuran 30 menit menghasilkan pola sinusoidal yang lebih jelas dibandingkan konfigurasi dua pompa dengan durasi hanya 5 menit. Perbedaan kualitas grafik ini dapat disebabkan oleh beberapa faktor, antara lain perbedaan jumlah pompa, variasi temperatur lingkungan, tingkat kekedapan chamber uji, maupun karakteristik gas dari masing-masing sampel daun.