

LAPORAN AKHIR

SISTEM PENGOLAHAN SINYAL

Pengembangan Aplikasi GUI untuk Visualisasi Data Electronic Nose (eNose) dengan Backend Rust dan Frontend Qt Python

Dosen: Ahmad Radhy S.Si, M.Si



Disusun Oleh Kelompok 10:

1. Amalia Fitria Damaiyanti (2042241021)
2. Duta Permana Agung (2042241041)
3. M. Friza Aditya Pradana (2042241048)

PRODI D4 TEKNOLOGI REKAYASA INSTRUMENTASI
DEPARTEMEN TEKNIK INSTRUMENTASI
FAKULTAS VOKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER

2025

Contents

| | |
|------------------------------------------------------------|----------|
| 1 LATAR BELAKANG | 3 |
| 2 METODOLOGI | 3 |
| 2.1 Gambaran Umum Sistem | 3 |
| 2.2 Perangkat dan Bahan | 4 |
| 2.3 Desain Backend | 4 |
| 2.4 Desain Frontend | 4 |
| 2.5 Alur Data dan Proses Pengolahan | 4 |
| 2.6 Pengujian Sistem | 4 |
| 2.7 Evaluasi dan Dokumentasi Data | 5 |
| 3 PERANCANGAN SISTEM | 5 |
| 3.1 Gambaran Umum Sistem | 5 |
| 3.2 Desain Software | 5 |
| 3.2.1 Backend (Rust) | 6 |
| 3.2.2 Frontend (PyQt6) | 6 |
| 3.3 Desain Hardware | 7 |
| 3.3.1 Alat dan Bahan | 7 |
| 3.3.2 Rancangan Susunan Hardware | 7 |
| 3.3.3 Gambar Perancangan Hardware | 8 |
| 3.4 Alur Data Sistem | 9 |
| 3.4.1 Alur Data Secara Logis | 9 |
| 3.4.2 Alur Data Secara Teknis | 9 |
| 3.5 Perancangan Komunikasi Sistem | 10 |
| 3.5.1 Komunikasi Arduino ke Backend | 10 |
| 3.5.2 Komunikasi Frontend ke Backend (GUI) | 10 |
| 3.5.3 Sinkronisasi Status Sistem | 11 |
| 3.5.4 Desain Penanganan Kondisi Tidak Normal | 11 |
| 3.5.5 Ringkasan Rancangan Komunikasi | 11 |
| 3.6 Perancangan GUI | 11 |
| 3.6.1 Tujuan Perancangan GUI | 11 |
| 3.6.2 Struktur Tampilan dan Tata Letak | 11 |
| 3.6.3 Perancangan Interaksi | 12 |
| 3.6.4 Perancangan Grafik Real - Time | 12 |
| 3.6.5 Perancangan Penyimpanan dan Notifikasi | 12 |
| 3.6.6 Antisipasi Masalah pada GUI | 12 |
| 3.6.7 Ringkasan Perancangan GUI | 12 |
| 3.7 Perancangan Preprocessing Sinyal | 13 |
| 3.7.1 Tujuan Preprocessing | 13 |
| 3.7.2 Masalah Umum pada Sinyal Sensor | 13 |
| 3.7.3 Pemotongan Bagian Sinyal yang Tidak Stabil | 13 |
| 3.7.4 Normalisasi Sinyal | 13 |
| 3.7.5 Pengurangan Noise | 13 |
| 3.7.6 Ekstraksi Fitur | 14 |
| 3.7.7 Pembagian Window Waktu | 14 |
| 3.7.8 Verifikasi Visual Hasil Preprocessing | 14 |
| 3.8 Perancangan Penyimpanan dan Ekspor Data | 14 |
| 3.8.1 Tujuan Perancangan Penyimpanan | 14 |
| 3.8.2 Struktur Data yang Disimpan | 14 |
| 3.8.3 Mekanisme Penyimpanan Selama Sampling | 15 |
| 3.8.4 Proses Ekspor ke CSV | 15 |
| 3.8.5 Penanganan Kesalahan | 15 |
| 3.8.6 Kompatibilitas dengan Edge Impulse | 15 |
| 3.9 Perancangan Integrasi Edge Impulse | 15 |
| 3.9.1 Tujuan Integrasi | 15 |
| 3.9.2 Penyusunan Format Dataset | 15 |
| 3.9.3 Pemilihan Label untuk Dataset | 16 |

| | | |
|----------|-------------------------------------------------------|-----------|
| 3.9.4 | Alur Unggah Data ke Edge Impulse | 16 |
| 3.9.5 | Integrasi dengan Preprocessing Edge Impulse | 16 |
| 3.9.6 | Persiapan untuk Training dan Evaluasi | 16 |
| 3.9.7 | Verifikasi Setelah Integrasi | 16 |
| 4 | IMPLEMENTASI | 16 |
| 4.1 | Implementasi Hardware | 16 |
| 4.2 | Implementasi Backend | 17 |
| 4.3 | Implementasi Frontend (GUI) | 19 |
| 4.4 | Implementasi Komunikasi Sistem | 21 |
| 4.5 | Implementasi Penyimpanan Data | 21 |
| 4.6 | Implementasi Integrasi Edge Impulse | 22 |
| 5 | HASIL PENGUJIAN | 22 |
| 6 | VISUALISASI GRAFIK | 23 |
| 6.1 | Grafik Gnuplot | 24 |
| 6.2 | Grafik Edge Impulse | 26 |
| 6.3 | Grafik Tampilan GUI | 28 |
| 6.3.1 | Dengan Satu Pompa | 28 |
| 6.3.2 | Dengan Dua Pompa | 30 |
| 7 | DESAIN 3D | 31 |
| 7.1 | Gambar 3D | 32 |
| 8 | ANALISIS DATA | 33 |
| 8.1 | Analisa Grafik Gnuplot | 34 |
| 8.2 | Analisa Grafik Edge Impulse | 35 |
| 8.3 | Grafik Tampilan GUI | 35 |
| 9 | KESIMPULAN | 35 |
| 9.1 | Saran dan Penutup | 36 |

1 LATAR BELAKANG

Electronic nose atau *e - nose* adalah sistem sensor gas yang dirancang untuk mengenali aroma. Prinsipnya mirip hidung manusia, tetapi bekerja dengan rangkaian sensor dan pemrosesan data. *E - nose* banyak digunakan di industri makanan, pemantauan udara, hingga riset kesehatan karena dapat memberi hasil yang cepat dan konsisten.

Saat sensor mendekripsi gas, datanya berbentuk sinyal waktu yang berubah setiap detik. Sinyal ini perlu diolah terlebih dahulu karena dapat mengandung *noise*, *drift*, atau respon sensor yang tidak stabil. Itu sebabnya kami butuh sistem yang bukan hanya membaca data, tetapi juga menampilkannya dalam bentuk yang mudah dipahami.

Dalam perkembangan teknologi instrumentasi modern, digitalisasi indra penciuman melalui sistem *Electronic Nose* atau *e - nose* menjadi solusi vital untuk pengujian kualitas produk secara non - destruktif dan cepat. Namun, tantangan utama dalam pengembangan *e - nose* terletak pada integrasi sistem akuisisi data yang mampu menangani streaming sinyal multi - sensor secara *real - time* tanpa latensi yang signifikan.

Pada proyek ini, kami membuat aplikasi GUI untuk memvisualisasikan data dari *e - nose* secara *real - time*. Proyek ini menjadi latihan untuk menghubungkan perangkat keras, pemrosesan sinyal, dan tampilan data dalam satu sistem.

Berikut alur kerja yang kami kembangkan:

1. Arduino Uno R4 WiFi mengumpulkan data dari 8 sensor gas.
2. *Backend* berbasis *Rust* menangani koneksi, pemrosesan awal, dan pengiriman data.
3. *Frontend* berbasis *Qt Python* menampilkan data dalam grafik yang terus diperbarui.

Proyek ini bertujuan merancang bangun sistem *e - nose* yang mengintegrasikan ketangguhan *backend* berbasis *Rust* untuk pemrosesan data konkuren dan fleksibilitas *frontend* *Qt Python* untuk visualisasi antarmuka.

Sistem juga dapat menyimpan data ke file CSV lalu diunggah ke *Edge Impulse* untuk analisis lanjutan seperti pelatihan model pengenalan aroma.

Pada proyek ini, sampel yang kami gunakan adalah 4 sabun cair dengan aroma dan merk yang berbeda seperti Downy, So Klin Pewangi, Dove, dan Gantle Gen. Secara spesifik, industri barang konsumen (FMCG) seperti produksi sabun mandi memerlukan pemantauan konsistensi profil aroma untuk menjamin standar kualitas. Lewat visualisasi data, kami dapat melihat pola perubahan sinyal dari beberapa parameter gas seperti CO, VOC, dan etanol.

Sebagai studi kasus validasi sistem, proyek ini difokuskan pada identifikasi profil aroma dari sampel sabun cair komersial, dimana deteksi senyawa volatil (VOC) dan etanol menjadi parameter kunci dalam menentukan karakteristik sinyal yang dihasilkan.

Proyek ini membantu kami belajar banyak hal:

1. Mengelola data sensor yang dinamis.
2. Membuat alur komunikasi antara *hardware* dan *software*.
3. Menampilkan data dalam bentuk yang informatif.
4. Memahami bagaimana pemrosesan sinyal diterapkan langsung pada alat.

Dengan sistem ini, proses pemantauan respons sensor menjadi lebih praktis dan intuitif. Selain itu, sistem masih dapat dikembangkan untuk kebutuhan *real - world* yang lebih kompleks.

2 METODOLOGI

Bab ini menjelaskan bagaimana kami membangun sistem dan melakukan pengujian. Mulai dari alat yang digunakan, arsitektur perangkat lunak, sampai cara pengambilan data. Metodologi penelitian ini dilaksanakan melalui pendekatan rekayasa sistem terintegrasi yang terdiri dari beberapa tahap.

2.1 Gambaran Umum Sistem

Kami membuat sistem *e - nose* yang dapat:

1. Membaca data sensor gas secara *real - time*.
2. Mengirimkan data ke laptop.
3. Menampilkan grafik dari 8 parameter gas.
4. Menyimpan data untuk di analisis.

Sistem terbagi menjadi 2 bagian. Arduino yang berfungsi sebagai pengumpul data dari sensor, laptop menjalankan *backend* dan *frontend* untuk memproses dan menampilkan sinyal. Dengan pembagian ini, aplikasi terasa fleksibel dan mudah dikembangkan lagi.

2.2 Perangkat dan Bahan

Perangkat yang kami gunakan:

1. Arduino Uno R4 WiFi sebagai pusat akuisisi data.
2. Modul sensor gas yang mendeteksi CO, VOC, etanol, dan NO₂.
3. Sensor tambahan untuk suhu, tekanan, dan kelembapan.
4. Laptop untuk menjalankan *backend* dan GUI.

Perangkat lunak yang kami gunakan:

1. *Rust* untuk *backend*.
2. PyQt6 dan PyQtGraph untuk GUI.
3. Serial port *library* untuk komunikasi Arduino.
4. JSON untuk format data.
5. CSV sebagai format penyimpanan.

Semua dipilih karena mudah diintegrasikan dan mendukung komunikasi data yang konsisten.

2.3 Desain Backend

Backend mengelola komunikasi antara Arduino dan GUI. Cara kerjanya seperti ini:

1. *Backend* membaca data serial dari Arduino dalam bentuk JSON.
2. *Backend* menyimpan data terbaru dalam buffer agar tidak hilang.
3. *Backend* mengirimkan data ke GUI secara terus - menerus.
4. *Backend* menyediakan REST API untuk tombol kontrol seperti start, stop, dan menyimpan data.

Rust dipilih karena memiliki performa bagus dan stabil untuk pemrosesan *real - time*. Selain itu, *backend* tetap berjalan walaupun Arduino belum terhubung karena terdapat mode simulasi.

2.4 Desain Frontend

Frontend adalah aplikasi GUI yang digunakan untuk:

1. Mengatur koneksi Arduino dan merk sabun cair.
2. Memulai dan menghentikan sampling.
3. Melihat sinyal dalam grafik secara *real - time*.
4. Menyimpan data untuk dianalisa.

GUI memuat dua tampilan grafik. Dimana grafik pertama menampilkan tiga parameter MICS sensor, dan grafik kedua menampilkan empat parameter GM sensor. Setiap grafik hanya menampilkan sekitar 200 titik data terakhir supaya GUI tetap responsif.

2.5 Alur Data dan Proses Pengolahan

Urutan pemrosesan data dalam sistem:

1. Sensor mengukur gas di sekitar sampel.
2. Arduino mengubah hasil menjadi angka.
3. Arduino mengirimkan angka ke *backend* melalui USB.
4. *Backend* memproses dan meneruskan data ke GUI.
5. GUI memperbarui grafik secara *real - time* setiap kali data baru masuk.
6. Pengguna dapat menyimpan hasil ke dalam file.

Dengan alur singkat ini, kami dapat melihat langsung perubahan sinyal ketika aroma berganti.

2.6 Pengujian Sistem

Pengujian dilakukan menggunakan beberapa sampel sabun cair dengan aroma dan merk yang berbeda. Tujuannya untuk melihat dan menganalisa apakah sensor memproses aroma yang berbeda dengan pola sinyal yang juga berbeda.

Langkah - langkah pengujian sistem yang kami kerjakan:

1. Menghubungkan Arduino ke laptop dan membuka GUI.
2. Pilih port Arduino dan memasukkan nama sampel sabun cair dari aroma ataupun merk.

3. Klik tombol konfigurasi lalu memulai sampling.
 4. Mendekatkan sampel sabun cair ke sensor.
 5. Menunggu selama pengukuran setiap sampel.
 6. Stop sampling dan menyimpan data.
 7. Mengulangi langkah - langkah yang sama untuk sampel yang berbeda.
- Setiap data yang tersimpan digunakan untuk analisis sinyal dan dapat diunggah ke *Edge Impulse*.

2.7 Evaluasi dan Dokumentasi Data

Setelah data terkumpul:

1. CSV dievaluasi untuk melihat pola perubahan tiap sensor.
2. Pola tersebut dibandingkan antar aroma.
3. Hasil menjadi dasar untuk analisis pada bab selanjutnya.

3 PERANCANGAN SISTEM

Bab ini menjelaskan bagaimana sistem *e - nose* dirancang sebelum masuk ke tahap implementasi. Perancangan ini meliputi struktur perangkat keras, arsitektur perangkat lunak, alur data, desain backend dan frontend, sampai proses pengolahan data yang digunakan di *Edge Impulse*.

Arsitektur sistem dirancang dengan model pemisahan *concern* untuk menjamin stabilitas performa. Pada sisi perangkat keras, sistem menggunakan Arduino Uno R4 WiFi sebagai unit akuisisi data yang membaca tegangan analog dari delapan kanal sensor gas yang berbeda. Data tersebut dikirimkan dalam format paket JSON melalui komunikasi serial ke komputer. Pada sisi perangkat lunak, *backend* berbasis *Rust* bertindak sebagai "otak" sistem yang menjalankan *asynchronous task* (*Tokio*) untuk membaca *port* serial dan melakukan *broadcasting* data melalui protokol *WebSocket*. Pendekatan ini dipilih untuk meminimalisir *bottleneck* pemrosesan data. Di sisi pengguna, aplikasi GUI berbasis *Qt Python* (PyQt6) dirancang untuk menerima aliran data tersebut dan memetakkannya ke dalam grafik *time - series* secara *real - time*, memberikan respons visual yang instan terhadap perubahan konsentrasi gas di ruang uji.

3.1 Gambaran Umum Sistem

Sebelum mulai membangun sistem *e - nose*, kami perlu menyusun konsep utamanya terlebih dahulu. Tujuannya agar semua bagian sistem bekerja saling mendukung dan tidak ada yang tumpang tindih. Dari situ, kami memecah sistem ke beberapa bagian yang memiliki tugas berbeda.

Bagian pertama adalah perangkat sensor; di tahap perancangan, kami tidak hanya memerlukan sensor apa saja yang digunakan, tetapi juga bagaimana setiap sensor memberikan informasi yang berguna untuk membedakan aroma sabun cair (sampel). Kami ingin data yang masuk ke sistem itu stabil dan mudah diproses, jadi cara membaca dan mengatur interval pengambilan data juga turut serta direncanakan.

Bagian kedua adalah *backend*; disini kami merancang bagaimana data dari Arduino akan dikelola, kami perlu memutuskan hal - hal seperti:

1. Format data yang konsisten.
2. Struktur variabel yang menyimpan hasil pembacaan.
3. Cara *backend* menyampaikan data ke aplikasi GUI tanpa *delay*.
4. Bagaimana *backend* harus bersikap kalau Arduino tidak terdeteksi.

Intinya *backend* dirancang sebagai pusat yang mengatur alur data, semua ini dipikirkan terlebih dahulu sebelum kodennya dibuat agar proses pengembangan lebih rapi.

Bagian ketiga adalah *frontend* atau GUI; pada tahap perancangan, kami menentukan apa saja yang harus ditampilkan di layar, urutan tampilannya, dan bagaimana kami dapat mengontrol proses sampling. Kami ingin antarmukanya sederhana, tetapi tetap menampilkan semua informasi penting seperti nilai sensor dan grafik *real - time*.

Dengan membagi sistem ketiga bagian ini (sensor, *backend*, dan GUI) kami memiliki gambaran besar yang jelas sebelum masuk ke implementasi. Pendekatan ini membantu menjaga alur kerja tahap teratur dan memastikan setiap komponen memiliki peran yang jelas di dalam sistem.

3.2 Desain Software

Sebelum masuk *coding*, kami menyusun rencana bagaimana setiap bagian *software* saling berhubungan. Sistem ini terdiri dari dua bagian besar, yaitu *backend* dan *frontend*.

3.2.1 Backend (Rust)

Backend dirancang sebagai pusat pengatur data, ada beberapa hal yang kami pikirkan sejak awal:

1. Bagaimana data dari Arduino dipastikan memiliki format yang sama setiap waktu?
2. Bagaimana backend dapat membaca data terus - menerus tanpa mengganggu proses lainnya?
3. Bagaimana data dibagikan ke GUI secepat mungkin?
4. Bagaimana *backend* menangani kondisi error seperti Arduino terputus?

Untuk itu, kami membuat dua struktur data utama; sensor *packet* untuk menyimpan nilai sensor per sampel dan *AppState* untuk menyimpan status sistem seperti *port*, mode sampling, dan *streaming* cukup cepat.

Selain itu, kami merancang fitur '*fallback mode*', ini memungkinkan *backend* tetap berjalan meskipun tidak ada Arduino dengan menghasilkan data simulasi. Fitur ini sangat membantu ketika melakukan pengembangan GUI.

Backend sistem dikembangkan menggunakan bahasa *Rust* dengan memanfaatkan *framework Actix - web* untuk menangani komunikasi HTTP dan *WebSocket*. Arsitektur *backend* dirancang sebagai 'otak' sistem yang tangguh, berfokus pada performa tinggi dan keamanan *thread*. Pengelolaan status global ditangani untuk memastikan bahwa konfigurasi *port*, status *running*, dan *buffer* data dapat diakses dengan aman oleh banyak koneksi secara bersamaan. Logika akuisisi data dieksekusi dalam *task* asinkron terpisah saat *endpoint POST* atau *start* dipanggil. *Task* ini menggunakan *library serial port* untuk terhubung ke Arduino. Desain ini mencakup mekanisme *fallback* yang krusial; jika koneksi serial gagal, sistem secara otomatis beralih ke mode simulasi tanpa menghentikan server. Untuk *streaming* data *real - time*, kami mengimplementasikan pola *Publish - Subscribe* dimana *task* akuisisi bertindak sebagai *publisher*. Setiap klien *WebSocket* yang terhubung di - *spawn* sebagai aktor yang bertindak sebagai *subscriber*, menerima siaran data sensor secara efisien. Selain itu, *backend* menyediakan *endpoint REST API* untuk kontrol dan ekspor data.

Sistem *backend* mengimplementasikan mekanisme *non - blocking* dan *asynchronous programming* untuk memastikan performa *real - time* pada *streaming* data sinyal. Data konfigurasi dari *frontend* diterima melalui *endpoint POST*, sedangkan data sinyal hasil komputasi dikirimkan secara kontinu melalui koneksi *WebSocket*. Untuk kebutuhan analisis offline, sistem juga menyediakan *endpoint GET* atau CSV yang mengekspor data sinyal dalam format CSV. *Backend* dirancang dengan *fault tolerance* yang baik, dimana setiap perubahan parameter dapat dilakukan kapan saja tanpa mengganggu proses *streaming* yang sedang berjalan.

catatan: code backend terdapat pada (figure: 1 - 5).

3.2.2 Frontend (PyQt6)

Frontend dirancang agar kami dapat fokus pada data, bukan pada pengaturannya. Hal yang kami rancang sejak awal:

1. Posisi tombol dan menu agar mudah dipahami.
2. Grafik harus responsif tetapi tidak membuat aplikasi berat.
3. Tampilan nilai sensor harus jelas meskipun datanya cepat berubah.
4. Kami dapat menyimpan langsung CSV atau mengunggah data ke *Edge Impulse*.

Kami juga merancang cara *frontend* berkomunikasi dengan *backend* melalui *WebSocket*. Sistem *event loop* PyQt6 dikombinasikan dengan *asyncio* agar aplikasi tidak mengalami *freeze (debugging)* saat menerima data secara terus - menerus.

Frontend dikembangkan menggunakan PyQt6 dengan PyQtGraph untuk visualisasi *real - time*. Antarmuka dirancang dengan pendekatan modular yang terdiri dari beberapa bagian utama: panel konfigurasi Arduino dengan deteksi *port* otomatis, panel konfigurasi sampel, panel kontrol (tombol *start* dan *stop*), panel status sistem yang menampilkan status koneksi dan data, dan area visualisasi grafik. Kami menggunakan *qasync* untuk mengintegrasikan *event loop asyncio* untuk *WebSocket* dengan *event loop* PyQt6, sehingga GUI tetap responsif dan tidak *freeze (debugging)* saat menerima data *streaming*. Untuk komunikasi, *library requests* digunakan untuk mengirim perintah REST API, sementara *library WebSockets* digunakan untuk koneksi *real - time* ke *endpoint backend*. Data grafik dikelola dalam *buffer numpy* dan hanya 200 titik data terakhir yang ditampilkan untuk menjaga performa tetap tinggi. Desain ini juga mencakup logika *fallback* simulasi internal, yang aktif jika *frontend* gagal terhubung ke *backend* saat tombol 'mulai sampling' ditekan.

catatan: code frontend terdapat pada (figure: 6 - 9).

3.3 Desain Hardware

Di bagian ini, fokus kami adalah begaimana menyusun komponen fisik agar sistem bekerja stabil dan mudah diuji. Perancangannya dimulai dari pemilihan sensor, cara menghubungkannya, sampai bagaimana Arduino mengatur alur pengambilan data.

Sistem menggunakan beberapa sensor gas dan lingkungan yang memiliki karakteristik berbeda. Setiap sensor menghasilkan respons yang tidak selalu sama cepat atau sama stabil. Karena itu, perancangannya bukan hanya perihal 'pasang sensor ke Arduino' tetapi juga memastikan pembacaannya konsisten. Kami mengatur pin input, interval pembacaan, dan cara Arduino menggabungkan nilai - nilainya dalam satu paket data.

Arduino Uno R4 WiFi dipilih karena memiliki kemampuan membaca analog yang cukup, hubungan serial yang stabil, dan mudah diintegrasikan dengan laptop. Dalam perancangannya, Arduino bertugas sebagai titik awal sistem; membaca sensor, membuat JSON yang rapi, dan mengirimkannya ke laptop.

Susunan *hardware* dibuat sederhana agar proses *debugging* lebih jelas. Semua sensor berada pada papan yang sama, dan kabel diatur agar tidak mengganggu aliran udara ke sensor. Rancangan ini juga memudahkan kami menambahkan sensor lain jika dibutuhkan.

3.3.1 Alat dan Bahan

Pada tahap perancangan, kami menentukan alat dan komponen apa saja yang dibutuhkan. Tujuannya agar sistem dapat membaca aroma dengan konsisten dan mudah dirangkai. Berikut alat dan bahan yang kami gunakan:

1. Arduino Uno R4 WiFi; digunakan sebagai mikrokontroller utama untuk membaca sensor dan mengirimkan data ke laptop.
 2. Sensor gas; MICS (CO, VOC, etanol), GM (NO₂, VOC, CO, etanol, SG40 (VOC), SHT40 (suhu dan kelembapan), LPS22DF (tekanan udara).
 3. *Breadboard* dan kabel jumper; untuk menyambungkan sensor ke Arduino tanpa *soldering*.
 4. Catu daya dari USB; Arduino mendapatkan daya sekaligus berkomunikasi lewat USB.
 5. Laptop; menjalankan *backend*, *frontend*, dan tempat perekaman data.
- Seluruh alat ini dipilih karena mudah diperoleh, mudah dirakit, dan kompatibel satu sama lain, sehingga memudahkan proses pengujian.

3.3.2 Rancangan Susunan Hardware

Sebelum *hardware* dirangkai, kami membuat rancangan sederhana tentang bagaimana sensor ditempatkan dan bagaimana kabel diatur. Ada beberapa hal yang kami pertimbangkan:

1. Sensor membutuhkan ruang untuk menangkap udara; jadi, sensor ditempatkan berjauhan dan tidak saling menutupi.
2. Kabel dijaga tetap rapi; kabel yang berantakan dapat menghalangi aliran udara ke sensor, jadi kami mengatur jalurnya sedemikian rupa.
3. Beban kerja Arduino dijaga tetap ringan; pengaturan pin dan interval pembacaan dibuat agar Arduino tetap stabil meskipun membaca beberapa sensor sekaligus.
4. Sensor dikelompokkan menurut jenisnya; contohnya sensor VOC diletakkan berdekatan agar responnya dapat dibandingkan lebih mudah.

Dengan perancangan ini, proses pemasangan menjadi lebih terarah dan tidak memakan waktu lama saat *debugging*.

3.3.3 Gambar Perancangan Hardware

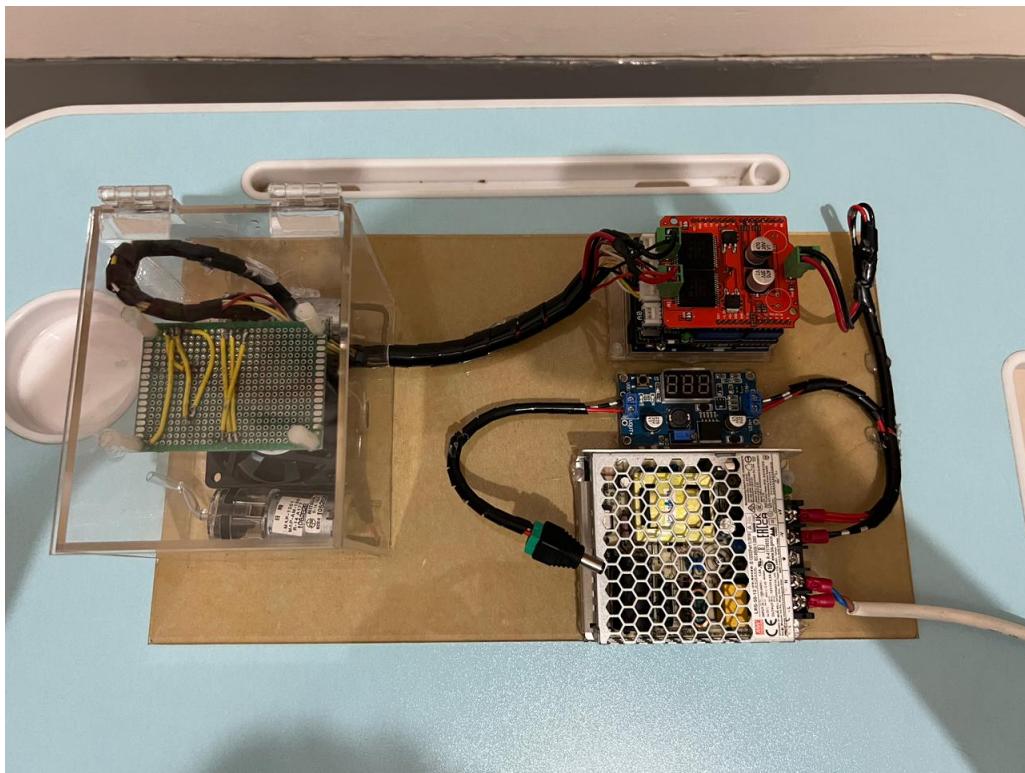


Figure 1: Hardware Tampak Atas

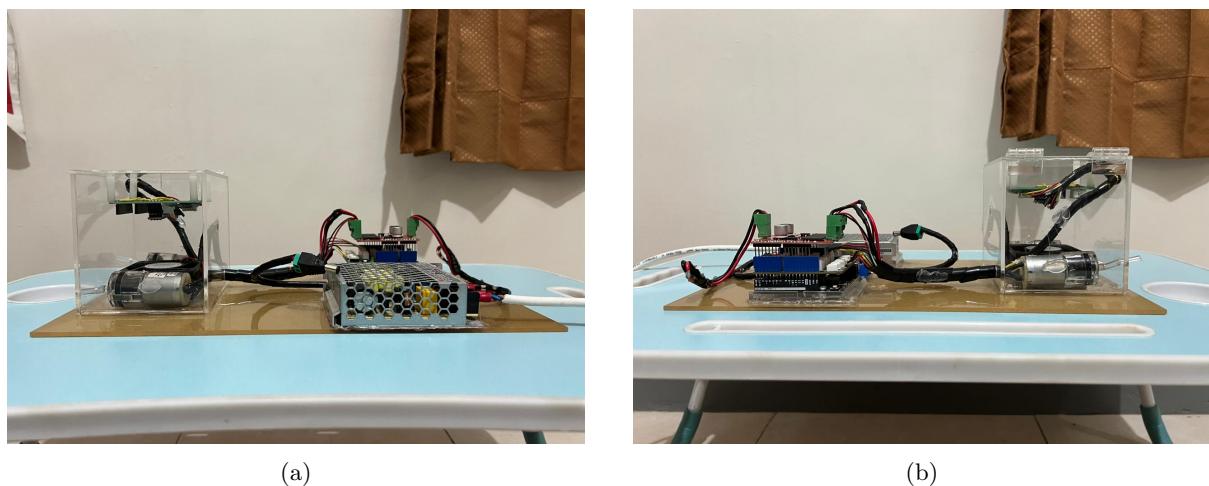
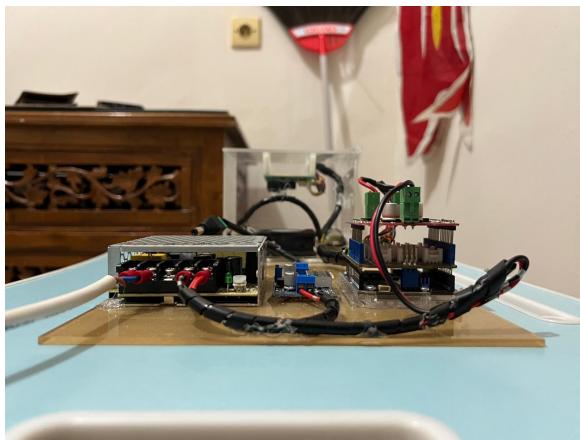


Figure 2: Hardware Tampak Samping



(a) Rangkaian Arduino Uno R4 WiFi



(b) Pompa dan Kipas

Figure 3

3.4 Alur Data Sistem

Alur data menjadi bagian penting dalam perancangan sistem karena semua komponen bergantung pada bagaimana data bergerak dari sensor sampai akhirnya ditampilkan atau disimpan. Agar sistem tetap stabil, kami merancang alur data yang jelas dan mudah diikuti. Dengan rancangan ini, setiap komponen tahu apa yang harus dilakukan dan kapan harus melakukannya.

Untuk menjelaskan alur data secara lebih rapi, bagian ini memiliki dua pembagian yaitu; alur data secara logis dan alur data secara teknis.

3.4.1 Alur Data Secara Logis

Pada level konsep, alur data dirancang sederhana agar mudah dipahami sebelum masuk ke detail teknis. Alurnya kami buat seperti ini:

1. Sensor membaca kondisi lingkungan; sensor gas dan sensor lingkungan menghasilkan nilai mentah sesuai kondisi udara di sekitarnya.
2. Arduino mengumpulkan semua nilai sensor; arduino bertugas mengambil nilai dari setiap sensor secara periodik.
3. Arduino membuat satu paket data; semua nilai digabung dalam satu struktur JSON agar *backend* tidak perlu menebak format datanya.
4. *Backend* menerima data dan menyimpannya; *backend* membaca JSON yang ada, lalu menyimpan nilai terbarunya di buffer internal agar dapat diakses kapan saja.
5. *Backend* mengirim data ke GUI; semua data terbaru dikirim ke GUI melalui WebSocket agar tampilannya selalu *real - time*.
6. GUI menampilkan data; GUI menampilkan data dalam bentuk angka dan grafik yang terus bergerak sesuai data terbaru.
7. Data direkam jika sampling aktif; ketika kami sedang mengambil sampel untuk dataset, data akan disimpan dalam list internal.
8. Data disimpan sebagai CSV; setelah sampling selesai, data dapat dieksport menjadi file CSV agar mudah dianalisis atau digunakan di *Edge Impulse*.

Alur logis ini memastikan semua bagian bekerja dalam satu jalur yang jelas dari awal sampai akhir.

3.4.2 Alur Data Secara Teknis

Pada tahap implementasi, alur data harus dirancang lebih rinci agar setiap komponen dapat bekerja dengan benar. Perancangan teknisnya kami buat seperti ini:

1. Interval pembacaan sensor di Arduino diatur tetap; interval ini menentukan seberapa cepat data masuk ke *backend*. Interval yang terlalu cepat membuat grafik berat, sedangkan terlalu lambat membuat respons sensor tidak terlihat. Karena itu, interval pembacaan dibuat seimbang.
2. Arduino mengirim JSON dengan struktur tetap; kami mendesain JSON berisi *timestamp*, nama sampel, nilai tiap sensor, dan status sampling. Dengan struktur tetap, *backend* dapat memproses data tanpa *parsing* rumit.

3. *Backend* menerima data menggunakan *thread* terpisah; *thread* khusus serial dibuat agar *backend* tetap responsif walaupun data masuk terus - menerus, data yang diterima langsung masuk ke *buffer*.
 4. *Backend* menggunakan sistem *broadcast* channel; setiap data baru dikirim ke semua klien *WebSocket* tanpa harus membaca *port* serial berkali - kali.
 5. GUI menerima data melalui *WebSocket*; GUI menjalankan *event loop* sendiri agar dapat menerima data tanpa membuat aplikasi *debugging*, data yang masuk langsung diperbarui ke grafik.
 6. *Buffer* di GUI menyimpan hanya data terbaru; untuk mencegah penggunaan memori berlebihan, GUI hanya menyimpan sekitar 200 sampel terakhir di grafik *real - time*, namun untuk keperluan sampling semua data tetap direkam.
 7. Ekspor CSV mengikuti standar dataset; kami merancang format CSV agar langsung cocok untuk *Edge Impulse*, termasuk kolom sensor dan *timestamp*.
 8. Jika ada gangguan, data tetap aman; jika Arduino terputus, *backend* tetap berjalan dan mengirimkan status ke GUI dan mode simulasi otomatis aktif sehingga sistem tetap dapat digunakan tanpa *hardware*.
- Dengan alur teknis seperti ini, seluruh sistem dapat mengalirkan data secara konsisten dan stabil dari sensor hingga penyimpanan akhir.

3.5 Perancangan Komunikasi Sistem

Komunikasi antara Arduino, *backend*, dan GUI adalah inti dari sistem ini. Jika alur komunikasinya tidak dirancang dengan baik, data dapat telat muncul, tidak sinkron, atau bahkan hilang. Karena itu, sebelum tahap implementasi, kami membuat rancangan bagaimana setiap bagian saling bertukar data dengan cara yang stabil dan mudah dipahami.

3.5.1 Komunikasi Arduino ke Backend

Arduino mengirimkan data sensor ke *backend* melalui koneksi serial. Agar *backend* dapat membaca data tanpa bingung, kami merancang beberapa aturan:

1. Format data harus konsisten; semua data yang dikirim berbentuk JSON. Dengan begitu, *backend* tinggal melakukan *parsing* tanpa harus menebak struktur datanya.
2. Setiap paket data berisi satu sampel lengkap; satu JSON mencakup *timestamp*, nilai seluruh sensor, status langkah pengukuran, nama sampel. Ini memudahkan *backend* untuk menyimpan dan mengirim data apa adanya.
3. Arduino mengirim data secara periodik; interval pengambilan ditetapkan pada jarak yang cukup cepat untuk *real-time*, tetapi tidak membuat Arduino atau *backend* kewalahan.
4. Penanganan error; kalau data yang diterima *backend* rusak atau tidak lengkap, *backend* mengabaikannya agar tidak mengganggu grafik di GUI.

Dengan rancangan ini, jalur Arduino menuju *backend* jadi stabil dan mudah diperiksa selama pengembangan.

3.5.2 Komunikasi Frontend ke Backend (GUI)

Agar GUI bisa menampilkan grafik secara *real - time*, *backend* butuh cara untuk mengirim data secepat mungkin tanpa membuat aplikasi berat. Karena itu, kami memakai *WebSocket*. Perancangannya seperti ini:

1. *WebSocket* sebagai jalur *streaming*; setiap kali *backend* menerima data baru dari Arduino, data itu langsung dikirim ke semua klien GUI yang sedang terhubung.
2. Pengiriman data bersifat *push*; GUI tidak perlu meminta data berkali - kali. *Backend* yang otomatis mengirim setiap ada update. Cara ini membuat tampilan grafik terasa lebih responsif.
3. *Broadcast* channel; *backend* menggunakan sistem *broadcast* internal agar satu data dapat dikirim ke banyak GUI sekaligus tanpa harus membaca serial berulang kali.
4. Format data tetap JSON; dengan cara ini, GUI tinggal memecah JSON itu menjadi nilai masing - masing sensor tanpa perlakuan khusus.
5. Penanganan koneksi putus; jika GUI terputus, *backend* tidak terganggu. GUI harus menyambungkan ulang *WebSocket* ketika siap.

Desain ini membuat komunikasi *backend* menuju GUI tetap lancar walaupun data masuk terus setiap beberapa ratus milidetik.

3.5.3 Sinkronisasi Status Sistem

Selain mengirimkan data sensor, *backend* juga perlu memberitahu GUI tentang status tertentu, misalnya:

1. Apakah Arduino sudah terhubung.
2. Apakah *backend* sedang membaca data.
3. Apakah mode sampling sedang berjalan.

Untuk itu, kami menyertakan beberapa *field* tambahan dalam data yang dikirim seperti *current level*, *current state*, atau *flag* koneksi Arduino.

Rancangannya dibuat agar GUI dapat menampilkan status sistem tanpa harus bertanya ulang ke *backend* melalui request terpisah. Ini membantu GUI tetap terasa ringan dan cepat.

3.5.4 Desain Penanganan Kondisi Tidak Normal

Dalam komunikasi sistem, hal yang sering terjadi adalah Arduino tiba - tiba terlepas atau data sensor bermasalah. Agar sistem tidak ikut macet, kami membuat beberapa rancangan antisipasi:

1. *Fallback mode* (simulasi); jika *backend* tidak menemukan Arduino, sistem otomatis masuk ke mode simulasi. GUI tetap bisa bekerja, jadi pengguna bisa mengecek grafik atau antarmuka tanpa harus memasang semua perangkat.

2. *Timeout* pembacaan serial; kalau Arduino berhenti mengirim data, *backend* tetap hidup dan memberi sinyal ke GUI bahwa perangkat terputus.

3. *Restart* koneksi otomatis; *backend* dapat membuka kembali *port* serial ketika kami memilih ulang *port* yang benar.

Rancangan ini membuat sistem lebih tahan gangguan dan lebih nyaman digunakan saat pengembangan.

3.5.5 Ringkasan Rancangan Komunikasi

Secara keseluruhan, komunikasi dalam sistem dirancang dengan tiga prinsip utama:

1. Format data konsisten, sehingga mudah diproses oleh *backend* dan GUI.
2. Transmisi *real - time*, agar grafik tidak lag.
3. Penanganan error yang jelas, agar aplikasi tidak macet ketika perangkat bermasalah.

Dengan rancangan seperti ini, alur data menjadi lebih stabil dan sistem lebih mudah dipelihara ataupun dikembangkan ke fitur baru.

3.6 Perancangan GUI

GUI (*Graphical User Interface*) dirancang sebagai cara utama kami berinteraksi dengan sistem. Karena itu, desainnya harus jelas, sederhana, dan tidak membuat kami bingung saat melakukan proses sampling. Pada tahap ini, fokus kami adalah menentukan apa saja yang perlu ditampilkan, bagaimana posisi komponennya, dan bagaimana GUI merespons data yang terus bergerak secara *real - time*. Untuk memudahkan penjelasan, perancangan GUI dibagi menjadi beberapa bagian.

3.6.1 Tujuan Perancangan GUI

Sebelum menentukan desain visualnya, kami menetapkan dua tujuan utama:

1. GUI harus mudah dipahami tanpa penjelasan panjang; kami cukup melihat komponen yang tersedia, lalu langsung mengerti alur kerjanya.
 2. GUI harus menampilkan data *real - time* tanpa terasa berat; sensor mengirim data cukup cepat, sehingga tampilan harus bisa mengikuti tanpa lag atau *freeze*.
- Dengan dua tujuan ini, kami merancang GUI supaya fungsional tapi tetap ringan.

3.6.2 Struktur Tampilan dan Tata Letak

Di tahap perancangan, kami membagi tampilan GUI menjadi beberapa bagian agar lebih teratur.

1. Panel konfigurasi Arduino, berisi; pilihan *port*, tombol *connect*, indikator apakah Arduino terdeteksi. Bagian ini dirancang sederhana agar kami dapat langsung memulai tanpa pengaturan yang rumit.

2. Panel kontrol sampling, terdiri dari; input nama sampel, tombol *start*, tombol stop, tombol simpan CSV, tombol unggah ke *Edge Impulse*. Tombol - tombol ini dikelompokkan dalam satu area agar lebih mudah digunakan saat proses sampling berlangsung.

3. Tampilan nilai sensor (angka); Bagian ini menampilkan nilai sensor secara langsung dalam bentuk angka, tujuannya agar kami dapat cepat melihat perubahan kecil di setiap sensor.
4. Grafik *real - time*; grafik dibagi menjadi dua kelompok yaitu grafik sensor MICS dan grafik sensor GMXXX. Pemisahan ini dilakukan agar grafik tetap mudah dibaca meskipun menampilkan banyak kurva sekaligus.
5. Area status sistem, berisi; status koneksi, status sampling, jumlah data yang sudah direkam, mode sistem (normal atau simulasi). Dengan pembagian seperti ini, kami dapat melihat semua informasi penting tanpa harus membuka menu tambahan.

3.6.3 Perancangan Interaksi

Agar pengalaman kami lebih lancar, kami juga merancang bagaimana setiap tombol atau aksi akan bekerja. Alur interaksi yang dirancang:

1. Memilih *port* Arduino.
2. *Backend* dihubungkan; status koneksi muncul di GUI.
3. Memberi nama pada sampel.
4. Menekan tombol *start*.
5. Data mengalir ke grafik dan direkam di list.
6. Setelah selesai, selanjutnya menekan stop.
7. Data dapat disimpan sebagai CSV atau langsung diunggah ke *Edge Impulse*.

Setiap langkah dibuat agar terasa natural dan tidak membingungkan, terutama untuk pengguna baru.

3.6.4 Perancangan Grafik Real - Time

Bagian grafik adalah salah satu bagian yang paling sering dilihat, sehingga perancangannya dibuat hati-hati. Kami mempertimbangkan beberapa hal:

1. Pembatas jumlah data di grafik; grafik hanya menampilkan sekitar 200 sampel terakhir. Ini menjaga GUI tetap ringan meskipun *streaming* data berjalan terus - menerus.
2. Warna grafik dibedakan antar sensor; warna dipilih agar mudah dibedakan tetapi tidak terlalu mencolok.
3. Skala grafik menyesuaikan otomatis; dengan begitu, kami tidak perlu mengatur zoom manual.
4. Grafik memperbarui setiap kali ada data baru; bukan pada interval tertentu, agar perubahan sensor terasa lebih nyata.

Tujuan akhirnya adalah membuat grafik yang informatif tanpa membebani sistem.

3.6.5 Perancangan Penyimpanan dan Notifikasi

GUI tidak hanya menampilkan data, tetapi juga mengatur proses penyimpanan. Perancangannya mencakup; indikator jumlah data yang terus diperbarui, pemberitahuan ketika sampling dimulai atau dihentikan, konfirmasi saat data berhasil disimpan, penanganan error, misalnya jika kami lupa mengisi nama sampel. Semua ini dibuat agar kami tidak perlu menebak - nebak apakah aksi mereka berhasil atau tidak.

3.6.6 Antisipasi Masalah pada GUI

Pada tahap perancangan, kami juga mempertimbangkan skenario yang mungkin terjadi seperti; koneksi *backend* terputus, data tidak masuk, mode simulasi aktif, *port* Arduino berubah. Jika hal ini terjadi, GUI menampilkan status yang jelas, misalnya '*Backend* tidak terhubung' atau "Arduino tidak ditemukan". Dengan begitu, kami dapat lebih cepat mengetahui apa yang salah tanpa harus mencari tahu secara manual.

3.6.7 Ringkasan Perancangan GUI

Secara keseluruhan, GUI dirancang untuk; mudah digunakan, menampilkan data *real - time* dengan lancar, memberikan kontrol penuh kepada kami, memberikan informasi status yang jelas, menyederhanakan proses pengumpulan data dan ekspor dataset. Dengan perancangan seperti ini, GUI tidak hanya menjadi alat visualisasi, tetapi juga pusat pengendali seluruh proses sampling.

3.7 Perancangan Preprocessing Sinyal

Preprocessing sinyal adalah langkah penting sebelum data digunakan untuk analisis atau *machine learning*. Sinyal dari sensor gas biasanya tidak langsung bersih. Respons awal dapat tidak stabil, beberapa sensor cenderung berisik, dan bentuk sinyal tiap sensor tidak selalu selaras. Karena itu, *preprocessing* perlu dirancang agar hasil akhirnya benar - benar siap digunakan. Bagian ini membahas bagaimana *preprocessing* direncanakan sejak awal sebelum data dikirim ke *Edge Impulse*.

3.7.1 Tujuan Preprocessing

Preprocessing dirancang dengan tiga tujuan utama:

1. Menstabilkan sinyal agar nilai yang dipakai benar - benar mencerminkan kondisi aroma.
2. Menyamakan skala antar sensor, karena tiap sensor memiliki rentang pembacaan berbeda.
3. Mengambil fitur yang paling menggambarkan pola aroma agar proses analisis atau klasifikasi lebih akurat.

Dengan tujuan ini, sistem *preprocessing* menjadi lebih terarah dan tidak hanya sekadar 'membersihkan data'.

3.7.2 Masalah Umum pada Sinyal Sensor

Sebelum menentukan *preprocessing* apa yang digunakan, kami memetakan masalah umum yang muncul pada sinyal:

1. Bagian awal sinyal sering melonjak atau belum stabil.
2. Beberapa sensor mengalami *noise* tinggi terutama saat kondisi lingkungan berubah.
3. Waktu respons sensor berbeda - beda, ada yang cepat, ada yang lambat.
4. Nilai antar sensor tidak berada pada rentang yang sama (contoh: ada sensor yang normalnya 5–50, tapi sensor lain 200–1500).
5. Kurva sinyal untuk aroma tertentu memiliki bentuk ciri khas, dan ciri tersebut perlu ditangkap sebagai fitur.

Rancangan *preprocessing* ini dibuat dengan mempertimbangkan masalah - masalah tersebut.

3.7.3 Pemotongan Bagian Sinyal yang Tidak Stabil

Bagian awal sinyal biasanya tidak mewakili respon sensor yang sebenarnya, ini dapat disebabkan oleh; sensor yang belum 'panas', aliran udara yang belum merata, perubahan mendadak ketika sampel mulai masuk.

Karena itu, *preprocessing* dirancang untuk memotong 10 - 20 persen bagian awal sebelum sinyal masuk ke proses berikutnya. Pemotongan ini dilakukan otomatis oleh *Edge Impulse* berdasarkan parameter *windowing* yang sudah dirancang.

3.7.4 Normalisasi Sinyal

Setiap sensor memiliki rentang nilai yang berbeda, sehingga sinyal perlu dinormalisasi. Tujuan normalisasi adalah; membuat pola antar sensor lebih mudah dibandingkan, menghindari sensor dengan nilai besar mendominasi model, memastikan seluruh data memiliki skala yang setara.

Metode normalisasi yang dirancang untuk digunakan:

1. *Min - Max Scaling* untuk membuat nilai berada di rentang 0 sampai 1.
2. *Standard Scaling* untuk mengubah sinyal menjadi rata - rata 0 dan deviasi standar 1.

Normalisasi ini dilakukan langsung oleh modul *preprocessing Edge Impulse*.

3.7.5 Pengurangan Noise

Beberapa sensor gas menghasilkan sinyal dengan fluktuasi kecil yang tidak menggambarkan perubahan aroma sebenarnya. Untuk itu, kami merancang penggunaan *moving average* atau median filter. Tergantung konfigurasi *Edge Impulse* yang digunakan. Tujuannya agar sinyal lebih halus tanpa menghilangkan pola penting seperti puncak respons sensor.

3.7.6 Ekstraksi Fitur

Setelah sinyal dibersihkan, kami merancang fitur - fitur yang akan diambil dari data. Fitur ini yang nanti digunakan untuk membedakan aroma. Fitur yang direncanakan mencakup:

1. Nilai rata-rata, menunjukkan tingkat konsentrasi gas secara umum.
 2. Nilai maksimum, menggambarkan puncak respons sensor saat aroma masuk.
 3. Waktu menuju puncak (*time - to - peak*), beberapa sensor lebih cepat mencapai puncak dibanding yang lain.
 4. Kenaikan dari baseline, selisih antara nilai awal dan nilai puncak.
 5. Area di bawah kurva (AUC), digunakan untuk menangkap kekuatan sinyal secara keseluruhan.
 6. Bentuk kurva, *Edge Impulse* dapat menggunakan *windowing* untuk menangkap pola sinyal keseluruhan, seperti kurva naik - turun khas aroma tertentu.
- Fitur - fitur ini dipilih karena terbukti sering digunakan untuk mengidentifikasi pola aroma pada sistem *e - nose*.

3.7.7 Pembagian Window Waktu

Edge Impulse menggunakan konsep *window* untuk memotong sinyal menjadi bagian yang lebih kecil agar bisa dianalisis. Pada tahap perancangan, kami menentukan; ukuran *window,overlap* antar - *window*, jumlah *frame* agar sinyal yang masuk tetap mencerminkan bentuk aslinya. *Window* yang terlalu kecil akan kehilangan pola, sedangkan *window* yang terlalu besar membuat model lambat. Karena itu, ukuran *window* dirancang agar cukup panjang untuk menangkap respons sensor, tetapi tetap efisien.

3.7.8 Verifikasi Visual Hasil Preprocessing

Sebelum data digunakan, kami melakukan pemeriksaan visual; apakah sinyal sudah halus, apakah pola aroma terlihat jelas, apakah pemotongan awal sudah tepat, apakah *noise* berkurang. Jika masih terlihat aneh, konfigurasi *preprocessing* diperbaiki lagi. Verifikasi visual ini menjadi bagian penting karena sinyal sensor lebih mudah dianalisis lewat grafik daripada angka mentah.

3.8 Perancangan Penyimpanan dan Ekspor Data

Sistem *e - nose* tidak hanya menampilkan data secara *real - time*, tetapi juga menyimpannya untuk keperluan analisis lebih lanjut. Karena itu, penyimpanan data dan proses ekspor menjadi bagian penting dalam perancangan sistem. Desainnya harus konsisten, mudah digunakan, dan kompatibel dengan *platform* analisis seperti *Edge Impulse*.

Bagian ini menjelaskan bagaimana proses penyimpanan dirancang sejak awal, termasuk format file, struktur data, dan cara sistem menangani penyimpanan selama sesi sampling.

3.8.1 Tujuan Perancangan Penyimpanan

Kami merancang mekanisme penyimpanan dengan beberapa tujuan utama:

1. Data dapat disimpan dengan format yang universal (CSV) dan mudah dibuka oleh siapa saja.
2. Struktur kolom harus konsisten, agar tidak menyulitkan proses analisis.
3. Penyimpanan dilakukan terpisah dari grafik *real - time*, agar GUI tetap ringan.
4. Data harus siap digunakan untuk *Edge Impulse* tanpa perlu banyak modifikasi.

Dengan tujuan ini, penyimpanan data menjadi lebih terarah dan tidak menimbulkan masalah saat proses training model.

3.8.2 Struktur Data yang Disimpan

Sebelum implementasi dimulai, kami menetapkan struktur data yang akan disimpan di file CSV. Kolom - kolom ini dirancang untuk mencakup semua informasi yang dibutuhkan seperti; *timestamp*, nama sampel, nilai sensor MICS (CO, VOC, ethanol), nilai sensor GMXXX (NO₂, VOC, CO, ethanol), nilai sensor lingkungan (suhu, kelembapan, dan tekanan), status langkah sampling (misalnya level atau step). Struktur ini memastikan bahwa semua data penting tercatat dan dapat diproses ulang tanpa kehilangan konteks.

3.8.3 Mekanisme Penyimpanan Selama Sampling

Selama proses sampling berlangsung, GUI tidak menyimpan data langsung ke file. Sebagai gantinya, kami merancang mekanisme penyimpanan sementara:

1. Setiap data yang masuk dari *backend* dimasukkan ke list atau *buffer* internal.
2. *Buffer* ini hanya menyimpan data ketika tombol *start* ditekan.
3. Ketika sampling stop, data dihentikan dan *buffer* siap dieksport.

Desain ini digunakan agar sistem tetap ringan selama proses berjalan dan hanya memproses ekspor sekali di akhir.

3.8.4 Proses Ekspor ke CSV

Setelah sesi sampling selesai, kami dapat menyimpan data ke CSV. Di tahap perancangan, kami membuat aturan:

1. File CSV menggunakan nama sampel sebagai nama file.
2. Setiap baris berisi satu sampel lengkap sesuai struktur kolom.
3. Tidak ada kolom kosong atau format campur yang dapat membingungkan *platform* analisis.
4. File CSV dapat langsung diunggah ke *Edge Impulse* tanpa proses tambahan.

Dengan rancangan seperti ini, proses ekspor menjadi cepat dan bebas error.

3.8.5 Penanganan Kesalahan

Beberapa hal yang perlu diantisipasi saat menyimpan data:

1. Nama sampel kosong, GUI memberi peringatan sebelum file dibuat.
2. File gagal ditulis, sistem menampilkan pesan error secara jelas.
3. *Buffer* kosong, kami diberi peringatan bahwa belum ada data yang dapat disimpan.

Dengan adanya perancangan ini, penyimpanan menjadi lebih aman dan kecil kemungkinan terjadi kesalahan yang merusak dataset.

3.8.6 Kompatibilitas dengan Edge Impulse

Karena data juga akan dipakai untuk *machine learning*, CSV dirancang agar cocok dengan *Edge Impulse*. Desainnya mencakup; format waktu yang sederhana, nilai sensor per sampel berada di kolom terpisah, tidak ada karakter yang membingungkan parser, kolom yang konsisten di setiap file CSV.

Dengan format ini, data langsung dapat digunakan untuk; training model, evaluasi respons sensor per aroma, perbandingan antar dataset.

3.9 Perancangan Integrasi Edge Impulse

Integrasi dengan *Edge Impulse* dirancang supaya data dari sistem *e - nose* dapat langsung digunakan untuk proses *machine learning*, seperti pengenalan aroma atau klasifikasi sampel. Agar prosesnya berjalan mulus, kami perlu menyiapkan format data, cara ekspor, serta alur kerja yang sesuai dengan kebutuhan *Edge Impulse*. Perancangan ini dilakukan sejak awal agar tidak ada penyesuaian besar ketika dataset mulai terkumpul.

3.9.1 Tujuan Integrasi

Sebelum membuat alur integrasi, kami menetapkan beberapa tujuan utama:

1. Data dapat langsung diunggah ke *Edge Impulse* tanpa perlu ubah format secara manual.
2. *Preprocessing* dapat berjalan otomatis berdasarkan konfigurasi yang sudah dirancang.
3. Setiap sampel memiliki identitas yang jelas, seperti nama aroma dan waktu pengambilan.
4. Proses integrasi efisien untuk pengguna, terutama ketika mengumpulkan banyak dataset.

Dengan tujuan ini, alur integrasi terasa lebih natural dan tidak memakan banyak waktu.

3.9.2 Penyusunan Format Dataset

Dataset dari sistem *e - nose* harus sesuai dengan format standar *Edge Impulse*. Karena itu, kami merancang struktur CSV yang berisi; *timestamp*, nilai masing-masing sensor, mendata sampel, urutan langkah sampling, nilai lingkungan bila diperlukan (suhu, kelembapan, tekanan). Struktur kolom ini disusun agar *parser* *Edge Impulse* dapat langsung mengenali dataset sebagai sinyal *multivariate*.

3.9.3 Pemilihan Label untuk Dataset

Setiap file CSV harus memiliki label yang jelas agar proses training lebih mudah. Saat perancangan, kami membuat beberapa aturan; nama file mengikuti pola (nama - sampel -tanggal.csv), label di *Edge Impulse* mengambil nama sampel dari file atau folder, nama label dibuat konsisten, misalnya kopi, teh, udara kosong, dan sebagainya.

Konsistensi label ini sangat penting karena *Edge Impulse* menggunakan label sebagai dasar pengelompokan data.

3.9.4 Alur Unggah Data ke Edge Impulse

Kami merancang alur integrasi agar sesederhana mungkin:

1. Mengumpulkan data melalui GUI.
2. Setelah sampling selesai, data disimpan sebagai CSV.
3. File CSV dipilih dan diunggah lewat menu 'Data Acquisition' di *Edge Impulse*.
4. Pada saat unggah, memilih label sesuai nama sampel.
5. *Edge Impulse* menjalankan validasi dan memproses file.

Dengan alur ini, tidak ada pembersihan data tambahan di luar sistem.

3.9.5 Integrasi dengan Preprocessing Edge Impulse

Edge Impulse menyediakan modul *preprocessing* bawaan, sehingga kami merancang data agar sesuai dengan *pipeline* tersebut. *Preprocessing* yang digunakan meliputi; *flattening* atau *windowing* untuk memotong sinyal ke beberapa *frame*, normalisasi otomatis, *smoothing* ringan, ekstraksi fitur dasar (*mean*, *max*, *peak*, *energy*), pembuatan fitur berbasis kurva respons sensor.

Kami menyesuaikan struktur CSV dan panjang sinyal agar modul - modul ini bisa bekerja optimal tanpa error.

3.9.6 Persiapan untuk Training dan Evaluasi

Selain tahap unggah data, integrasi juga mencakup perancangan bagaimana dataset akan dipakai untuk training dan evaluasi model. Kami membuat beberapa pedoman; dataset harus memiliki ukuran *window* yang seragam, nilai sensor tidak boleh hilang atau kosong, setiap kelas aroma mempunyai jumlah sampel yang seimbang, sinyal harus mengandung respons penuh, bukan hanya *baseline*.

Rancangan ini dibuat supaya model tidak bias atau kesulitan mempelajari pola aroma.

3.9.7 Verifikasi Setelah Integrasi

Setelah data masuk ke *Edge Impulse*, kami melakukan beberapa pengecekan:

1. Apakah sinyal terlihat normal di tampilan grafik.
2. Apakah preprocessing bekerja sesuai pengaturan.
3. Apakah label sudah terbaca dengan benar.
4. Apakah dataset dapat digunakan untuk training tanpa error.

Langkah verifikasi ini membantu memastikan bahwa integrasi berjalan dengan baik dan dataset siap digunakan.

4 IMPLEMENTASI

Bab ini membahas bagaimana rancangan yang dijelaskan pada Bab 3 diterapkan menjadi sistem yang benar - benar berjalan. Implementasi dilakukan bertahap, mulai dari merakit *hardware*, membuat *backend* untuk membaca data sensor, mengembangkan GUI sebagai tampilan utama, hingga menyiapkan proses penyimpanan data dan integrasi ke *Edge Impulse*.

4.1 Implementasi Hardware

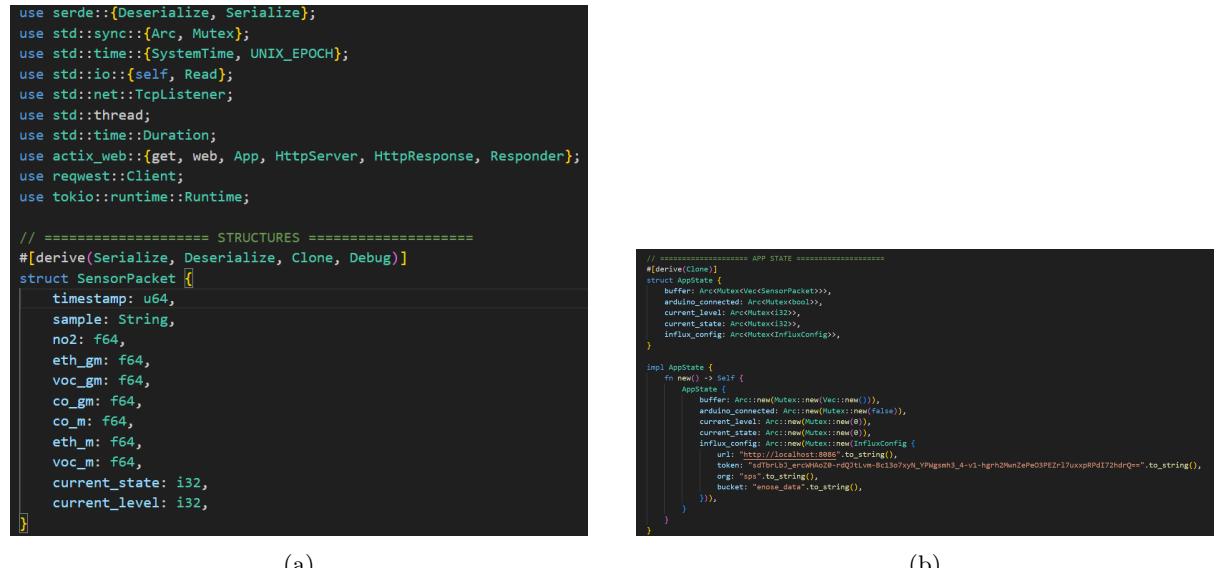
Tahap pertama adalah merakit perangkat *e - nose* sesuai rancangan pada bab sebelumnya. Semua sensor ditempatkan pada satu *board* agar mudah diuji dan tetap mendapat aliran udara yang baik.

- Penyambungan sensor ke Arduino; setiap sensor dihubungkan ke Arduino Uno R4 WiFi melalui kombinasi pin analog, digital, dan I2C. Pengkabelan dibuat serapi mungkin agar tidak mengganggu aliran udara sekitar sensor.
- Konfigurasi pin dan interval pembacaan; Arduino diprogram untuk membaca seluruh sensor pada interval tetap. Interval dipilih agar sinyal masih terlihat *real - time* tetapi Arduino tidak kelebihan beban.
- Pengujian tahap awal, setelah selesai dirakit perangkat diuji dengan: mengecek apakah setiap sensor mengeluarkan nilai, memastikan pembacaan tidak *delay*, memastikan tidak ada kabel yang longgar.

4.2 Implementasi Backend

Backend dibuat menggunakan bahasa *Rust*. Bagian ini menjadi pusat pengolah data sebelum diteruskan ke *GUI*.

- Membaca data dari Arduino; *backend* membuka koneksi serial ke Arduino, lalu membaca *string JSON* yang dikirim setiap interval. JSON ini langsung di *parse* menggunakan *library deserialization* agar nilai sensor mudah digunakan.
- Penggunaan struktur data, dua struktur utama diimplementasikan; *SensorPacket* untuk menampung nilai - nilai sensor per sampel. Sedangkan, *AppState* untuk menyimpan status sistem seperti *port* yang digunakan, *buffer* data, dan mode sampling.
- Sistem *thread* dan *broadcast* channel untuk menghindari *bottleneck*; satu *thread* khusus membaca data serial, *thread* lain mengurus *WebSocket*, data dikirim melalui channel *broadcast*, sehingga semua klien *GUI* bisa mendapat data yang sama tanpa membaca serial ulang.
- WebSocket* server; *backend* menyalurkan data sensor melalui *WebSocket*. Tiap ada sampel baru, *backend* mengirim JSON ke *GUI* dalam hitungan milidetik.
- Mode simulasi; jika Arduino tidak terdeteksi, *backend* secara otomatis menghasilkan data *dummy*. Fitur ini memudahkan pengembangan *GUI* tanpa harus menyalaikan seluruh *hardware*.



```

use serde::Deserialize;
use std::sync::{Arc, Mutex};
use std::time::{SystemTime, UNIX_EPOCH};
use std::io::{self, Read};
use std::net::TcpListener;
use std::thread;
use std::time::Duration;
use actix_web::{get, web, App, HttpServer, HttpResponse, Responder};
use request::Client;
use tokio::runtime::Runtime;

// ===== STRUCTURES =====
#[derive(Deserialize, Serialize, Clone, Debug)]
struct SensorPacket {
    timestamp: u64,
    sample: String,
    no2: f64,
    eth_gm: f64,
    voc_gm: f64,
    co_gm: f64,
    co_m: f64,
    eth_m: f64,
    voc_m: f64,
    current_state: i32,
    current_level: i32,
}

// ===== APP STATE =====
struct AppState {
    buffer: Arc<Mutex<Vec<SensorPacket>>>,
    arduino_connected: Arc<Mutex<i32>>,
    current_level: Arc<Mutex<i32>>,
    current_state: Arc<Mutex<i32>>,
    influx_config: Arc<Mutex<InfluxConfig>>,
}

impl AppState {
    fn new() -> Self {
        AppState {
            buffer: Arc::new(Mutex::new(Vec::new())),
            arduino_connected: Arc::new(Mutex::new(false)),
            current_level: Arc::new(Mutex::new(0)),
            current_state: Arc::new(Mutex::new(0)),
            influx_config: Arc::new(Mutex::new(InfluxConfig {
                token: "sdibrubj_ercd04020-ndQ0t1vm-Bc13o7xyM_lYPhgsmh3_4-v1-hgrh2MnZePeOPEZ=17uxpRdI72hdRQ==".to_string(),
                org: "ps".to_string(),
                bucket: "enose_data".to_string(),
            })),
        }
    }
}

```

(a)

(b)

Figure 4: Kode Backend

```
// [ ] FORMAT 1: JSON
if cleaned_line.starts_with('{') {
    match serde_json::from_str::<serde_json::Value>(cleaned_line) {
        Ok(parsed) => {
            let no2 = parsed["no2_gm"].as_f64().unwrap_or(-1.0);
            let eth_gm = parsed["eth50h_gm"].as_f64().unwrap_or(-1.0);
            let voc_gm = parsed["voc_gm"].as_f64().unwrap_or(-1.0);
            let co_gm = parsed["co_gm"].as_f64().unwrap_or(-1.0);
            let co_m = parsed["co_mics"].as_f64().unwrap_or(-1.0);
            let eth_m = parsed["eth_mics"].as_f64().unwrap_or(-1.0);
            let voc_m = parsed["voc_mics"].as_f64().unwrap_or(-1.0);

            let state_str = parsed["state"].as_str().unwrap_or("IDLE");
            let current_state = match state_str {
                "IDLE" => 0,
                "PRE_COND" => 1,
                "RAMP_UP" => 2,
                "HOLD" => 3,
                "PURGE" => 4,
                "RECOVERY" => 5,
                "DONE" => 6,
                _ => 0
            };

            let current_level = parsed["currentLevel"].as_i64().unwrap_or(0) as i32;
        }
    }
}

// [ ] FORMAT 2: SENSOR: CSV
if cleaned_line.starts_with("SENSOR:") {
    let data_part = &cleaned_line[7..];
    let parts: Vec<&str> = data_part.split(',').collect();

    if parts.len() == 9 {
        let no2 = parts[0].parse().unwrap_or(-1.0);
        let eth_gm = parts[1].parse().unwrap_or(-1.0);
        let voc_gm = parts[2].parse().unwrap_or(-1.0);
        let co_gm = parts[3].parse().unwrap_or(-1.0);
        let co_m = parts[4].parse().unwrap_or(-1.0);
        let eth_m = parts[5].parse().unwrap_or(-1.0);
        let voc_m = parts[6].parse().unwrap_or(-1.0);
        let current_state = parts[7].parse().unwrap_or(0);
        let current_level = parts[8].parse().unwrap_or(0);
    }
}
```

(a)

(b)

Figure 5: JSON dan CSV

```
// [ ] FORMAT 3: Direct CSV tanpa prefix
let parts: Vec<&str> = cleaned_line.split(',').collect();
if parts.len() == 9 {
    let no2 = parts[0].parse().unwrap_or(-1.0);
    let eth_gm = parts[1].parse().unwrap_or(-1.0);
    let voc_gm = parts[2].parse().unwrap_or(-1.0);
    let co_gm = parts[3].parse().unwrap_or(-1.0);
    let co_m = parts[4].parse().unwrap_or(-1.0);
    let eth_m = parts[5].parse().unwrap_or(-1.0);
    let voc_m = parts[6].parse().unwrap_or(-1.0);
    let current_state = parts[7].parse().unwrap_or(0);
    let current_level = parts[8].parse().unwrap_or(0);

    return Some(SensorPacket {
        timestamp: SystemTime::now().duration_since(UNIX_EPOCH).unwrap().as_millis() as u64,
        sample: "Active".to_string(),
        no2,
        eth_gm,
        voc_gm,
        co_gm,
        co_m,
        eth_m,
        voc_m,
        current_state,
        current_level,
    });
}
```

Figure 6: Direct CSV Tanpa Prefix

```
// ===== TCP SERVER =====
fn tcp_server_worker(state: AppState) {
    println!("[+] Starting TCP server on 0.0.0.0:8081");

    match TcpListener::bind("0.0.0.0:8081") {
        Ok(listener) => {
            println!("[+] TCP Server started on 0.0.0.0:8081");

            listener.set_nonblocking(true).expect("Cannot set non-blocking");

            loop {
                match listener.accept() {
                    Ok((stream, addr)) => {
                        println!("[+] Arduino connected from: {}", addr);

                        let state_for_thread = state.clone();
                        thread::spawn(move || {
                            handle_arduino_connection(stream, state_for_thread);
                        });
                    }
                    Err(e) => {
                        if e.kind() == io::ErrorKind::WouldBlock {
                            thread::sleep(Duration::from_millis(50));
                            continue;
                        } else {
                            eprintln!("[!] TCP accept error: {}", e);
                        }
                    }
                }
            }
        }
    }
}

// ===== HANDLE ARDUINO CONNECTION =====
fn handle_arduino_connection(mut stream: std::net::TcpStream, state: AppState) {
    println!("[+] Handling Arduino connection...");

    // Set timeout for reading
    stream.set_read_timeout(Some(Duration::from_secs(10))).ok();

    let mut buffer = [0; 1024];
    let mut data_buffer = String::new();

    // Mark as connected
    {
        let mut connected = state.arduino_connected.lock().unwrap();
        *connected = true;
        println!("[+] Arduino marked as connected");
    }
}
```

(a) TCP Server

(b) Arduino Connection

Figure 7

```
// ===== HTTP ENDPOINTS =====
#[get("/api/status")]
async fn get_status(data: web::Data) -> impl Responder {
    let connected = data.arduino_connected.lock().unwrap();
    let level = data.current_level.lock().unwrap();
    let current_state = data.current_state.lock().unwrap();
    let buffer = data.buffer.lock().unwrap();

    HttpResponse::Ok().json(&serde_json::json!({
        "arduino_connected": *connected,
        "current_level": *level,
        "current_state": *current_state,
        "data_points": buffer.len()
    }));
}

// ===== HEALTH CHECK ENDPOINT =====
#[get("/api/health")]
async fn health_check(data: web::Data) -> impl Responder {
    let connected = data.arduino_connected.lock().unwrap();
    let buffer = data.buffer.lock().unwrap();

    HttpResponse::Ok().json(&serde_json::json!({
        "status": "ok",
        "service": "e-nose-backend",
        "version": "1.0.0",
        "arduino_connected": *connected,
        "data_points": *buffer.len(),
        "timestamp": SystemTime::now().duration_since(UNIX_EPOCH).unwrap().as_millis()
    }));
}

// ===== MAIN =====
#[actix_web::main]
async fn main() -> std::io::Result<()> {
    println!("[+] Electronic Nose Backend - Multi Service");
    println!("=====");
    println!("[+] TCP Server on port 8081 (Arduino)");
    println!("[+] HTTP API on port 8080 (PyQt6 Frontend)");
    println!("[+] InfluxDB integration ready");
    println!("[+] Waiting for connections..");
    println!("[+] Available HTTP Endpoints:");
    println!("- http://0.0.0.0:8080 - Status page");
    println!("- http://0.0.0.0:8080/api/health - Health check");
    println!("- http://0.0.0.0:8080/api/status - System status");
    println!("- http://0.0.0.0:8080/api/data - Sensor data (JSON)");
    println!("- http://0.0.0.0:8080/api/csv - Export CSV");
    println!("=====");
    let state = AppState::new();
}
```

(a) HTTP Endpoints

(b) Health Check Endpoint

(c) Main

Figure 8

4.3 Implementasi Frontend (GUI)

Frontend dibangun menggunakan PyQt6. Bagian ini menjadi tempat untuk mengatur sampling, melihat grafik, dan menyimpan data.

- Pembuatan *layout* GUI, komponen GUI meliputi; menu pemilihan *port* Arduino, tombol *connect*, *start*, *stop*, *save*, dan *upload*, tampilan angka nilai sensor, dua grafik *real - time* (MICS dan GMXXX), status sistem seperti jumlah data dan mode *backend*. Semua komponen disusun agar tidak membungkungkan pengguna baru.

- Implementasi grafik *real - time*; grafik dibuat menggunakan *pyqtgraph* karena lebih cepat dibanding *Matplotlib*. GUI hanya menyimpan sekitar 200 data terbaru untuk grafik, sehingga tampilan tetap halus meskipun data masuk cepat.

- Integrasi *WebSocket*; GUI memakai *asyncio* untuk menerima data dari *backend* tanpa membuat aplikasi *freeze*. Setiap data baru langsung diupdate ke grafik dan panel angka.

- Logging* dan notifikasi; GUI menampilkan status seperti *backend* terhubung atau tidak, sampling aktif atau berhenti, jumlah data terkumpul, penyimpanan CSV berhasil.

```

import sys
import requests
import json
import socket
import time
from datetime import datetime
from PyQt6 import QtWidgets, QtCore, QtGui
from PyQt6.QtCore import QTimer
import pyqtgraph as pg

# ===== KONFIGURASI =====
RUST_IP = "192.168.1.13" # IP backend Rust
RUST_PORT = 8080 # Port HTTP backend Rust
|
# BARU: Arduino Direct Control
ARDUINO_IP = "192.168.1.21" # Ganti dengan IP Arduino
ARDUINO_PORT = 8082

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("E-NOSE Monitor - Direct Arduino Control")
        self.resize(1400, 900)

```

(a)

```

# Set style untuk UI
self.setStyleSheet("""
QMainWindow { background-color: #f0f2f5; }
QGroupBox {
    font-weight: bold; border: 2px solid #d1d5db; border-radius: 8px;
    margin-top: 1ex; padding-top: 10px; background-color: white;
}
QGroupBox::title {
    subcontrol-origin: margin; left: 10px; padding: 0 8px 0 8px;
    color: #374151; font-weight: bold; font-size: 11px;
}
QPushButton {
    background-color: #3b82f6; border: none; color: white;
    padding: 6px 12px; font-size: 11px; font-weight: bold;
    margin: 2px; border-radius: 4px;
}
QPushButton:hover { background-color: #2563eb; }
QPushButton:pressed { background-color: #1d4ed8; color: #6b7280; }
QLineEdit { padding: 6px; border: 1px solid #d1d5db; border-radius: 4px; }
QLabel { color: #374151; font-size: 11px; }
""")

```

(b)

Figure 9: Kode Frontend

```

# BARU: Sampling Control Section
sampling_group = QtWidgets.QGroupBox("Sampling Control - DIRECT to Arduino")
sampling_layout = QtWidgets.QGridLayout(sampling_group)

self.btn_start_sampling = QtWidgets.QPushButton("START")
self.btn_stop_sampling = QtWidgets.QPushButton("STOP")
self.sampling_status = QtWidgets.QLabel("Status: IDLE")
self.actuator_status = QtWidgets.QLabel("Actuator: -")

self.btn_start_sampling.setStyleSheet("background-color: #10b981; font-weight: bold;")
self.btn_stop_sampling.setStyleSheet("background-color: #f4444; font-weight: bold;")
self.sampling_status.setStyleSheet("padding: 4px; background-color: #f3f4f6; border-radius: 3px; font-weight: bold;")
self.actuator_status.setStyleSheet("padding: 4px; background-color: #f3f4f6; border-radius: 3px; font-weight: bold;")

self.btn_start_sampling.clicked.connect(self.start_sampling_direct)
self.btn_stop_sampling.clicked.connect(self.stop_sampling_direct)

sampling_layout.addWidget(self.btn_start_sampling, 0, 0)
sampling_layout.addWidget(self.btn_stop_sampling, 0, 1)
sampling_layout.addWidget(self.sampling_status, 1, 0)
sampling_layout.addWidget(self.actuator_status, 1, 1)
controls_layout.addWidget(sampling_group)

```

(a) Sampling Control Section

```

# System Status
status_group = QtWidgets.QGroupBox("System Status")
status_layout = QtWidgets.QGridLayout(status_group)
self.state_label = QtWidgets.QLabel("State: IDLE")
self.level_label = QtWidgets.QLabel("Level: 0")
self.data_count_label = QtWidgets.QLabel("Data Points: 0")
self.session_label = QtWidgets.QLabel("Session: None")

status_labels = [self.state_label, self.level_label, self.data_count_label, self.session_label]
for label in status_labels:
    label.setStyleSheet("padding: 4px; background-color: #f3f4f6; border-radius: 3px;")

status_layout.addWidget(self.state_label, 0, 0)
status_layout.addWidget(self.level_label, 0, 1)
status_layout.addWidget(self.data_count_label, 1, 0)
status_layout.addWidget(self.session_label, 1, 1)
controls_layout.addWidget(status_group)

# Sensor Values Display
values_layout = QtWidgets.QHBoxLayout()

```

(b) System Status

Figure 10

```

# BARU: Arduino Direct Control
self.arduino_socket = None

# Test connection on startup
QtCore.QTimer.singleShot(1000, self.test_connection)

def get_backend_url(self, endpoint=""):
    """Get backend URL for HTTP requests"""
    ip = self.backend_ip.text().strip() or RUST_IP
    port = self.backend_port.text().strip() or str(RUST_PORT)
    return f"http://(ip):(port){endpoint}"

def test_connection(self):
    """Test connection to Rust backend via HTTP"""
    try:
        response = requests.get(self.get_backend_url("/api/status"), timeout=5)
        if response.status_code == 200:
            data = response.json()
            self.connection_status.setText("Connected")

```

(a) Arduino Direct Control

```

# BARU: DIRECT ARDUINO CONTROL METHODS
def send_direct_to_arduino(self, command):
    """Kirim command langsung ke Arduino via TCP"""
    try:
        if self.arduino_socket is None:
            self.arduino_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.arduino_socket.settimeout(5)
            self.arduino_socket.connect((ARDUINO_IP, ARDUINO_PORT))
            print(f"Connected to Arduino [{ARDUINO_IP}]:{ARDUINO_PORT}")

        self.arduino_socket.sendall(f"{command}\n".encode())
        print(f"Direct to Arduino: {command}")
        return True

    except Exception as e:
        print(f"Direct Arduino failed: {e}")
        self.arduino_socket = None
        QtWidgets.QMessageBox.warning(self, "Direct Control Error",
                                      f"Failed to send command to Arduino:\n{str(e)}")
        return False

```

(b) Arduino Control Methods

Figure 11

```

# [✓] BARU: Update sampling status dari Arduino data
sampling_active = data.get('current_level', 0) # Arduino kirim samplingActive di current_level
sampling_state = data.get('current_state', 0) # Arduino kirim state di current_state

if sampling_active == 1: # Jika sampling aktif
    if sampling_state == 1:
        self.actuator_status.setText("Actuator: [🌀] KIPAS (2 menit)")
    elif sampling_state == 2:
        self.actuator_status.setText("Actuator: [💧] POMPA (4 menit)")
    else:
        self.actuator_status.setText("Actuator: -")

```

Figure 12: Update Sampling Status dari Arduino Data

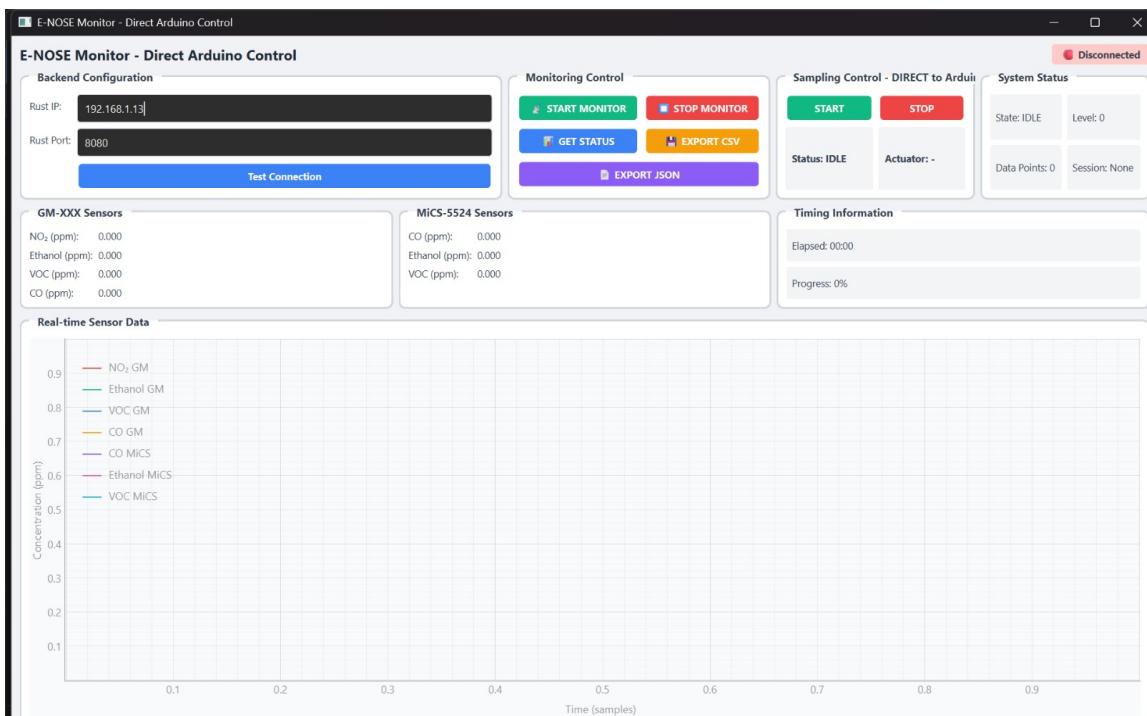


Figure 13: Tampilan GUI

4.4 Implementasi Komunikasi Sistem

Bagian komunikasi menjadi penghubung antara *hardware* dan GUI.

1. Serial menuju *backend*, Arduino mengirim JSON tiap interval. *Backend* memeriksa: apakah JSON valid, apakah semua *field* sensor lengkap, apakah *timestamp* masuk akal.
2. *Backend* menuju GUI; setiap JSON yang valid dikirim ke GUI melalui *WebSocket*. Formatnya tetap sama supaya GUI tidak perlu *parsing* ulang yang rumit.
3. Penanganan kondisi error, jika ada kondisi seperti: data tidak lengkap, Arduino terputus, *port* salah, *backend* mengirim status error ke GUI sehingga kami langsung mengerti apa yang bermasalah.

4.5 Implementasi Penyimpanan Data

Mekanisme penyimpanan dibuat sederhana tapi aman.

1. *Buffer* penyimpanan di GUI; selama sampling aktif, GUI menyimpan setiap sampel ke list internal.
2. Ekspor ke CSV, ketika sampling dihentikan; GUI membuat file CSV berdasarkan nama sampel, menulis seluruh data sesuai urutan waktu, memastikan kolom konsisten (sensor, *timestamp*, label).
3. Struktur kolom konsisten; kolom - kolom disusun agar kompatibel dengan *Edge Impulse*.
4. Error handling, GUI menampilkan pesan jika: nama sampel kosong, data tidak ada, file tidak dapat disimpan.

4.6 Implementasi Integrasi Edge Impulse

Bagian ini menghubungkan sistem *e - nose* dengan *platform Edge Impulse*.

1. Format CSV sesuai standar *Edge Impulse*, format data diuji untuk memastikan *Edge Impulse* dapat membaca; *timestamp*, nilai sensor, label sampel.
2. Unggah dataset ke *Edge Impulse*; kami dapat mengunggah file CSV yang sudah dibuat melalui halaman Data *Acquisition*.
3. *Preprocessing* otomatis; *Edge Impulse* melakukan *preprocessing* seperti normalisasi dan *smoothing* berdasarkan pengaturan yang sudah disiapkan sebelumnya.
4. Verifikasi dataset, setelah data masuk, dicek; apakah grafik sinyal terlihat normal, apakah jumlah fitur masuk akal, apakah label terbaca benar. Integrasi dianggap berhasil jika semua data dapat masuk tanpa error.

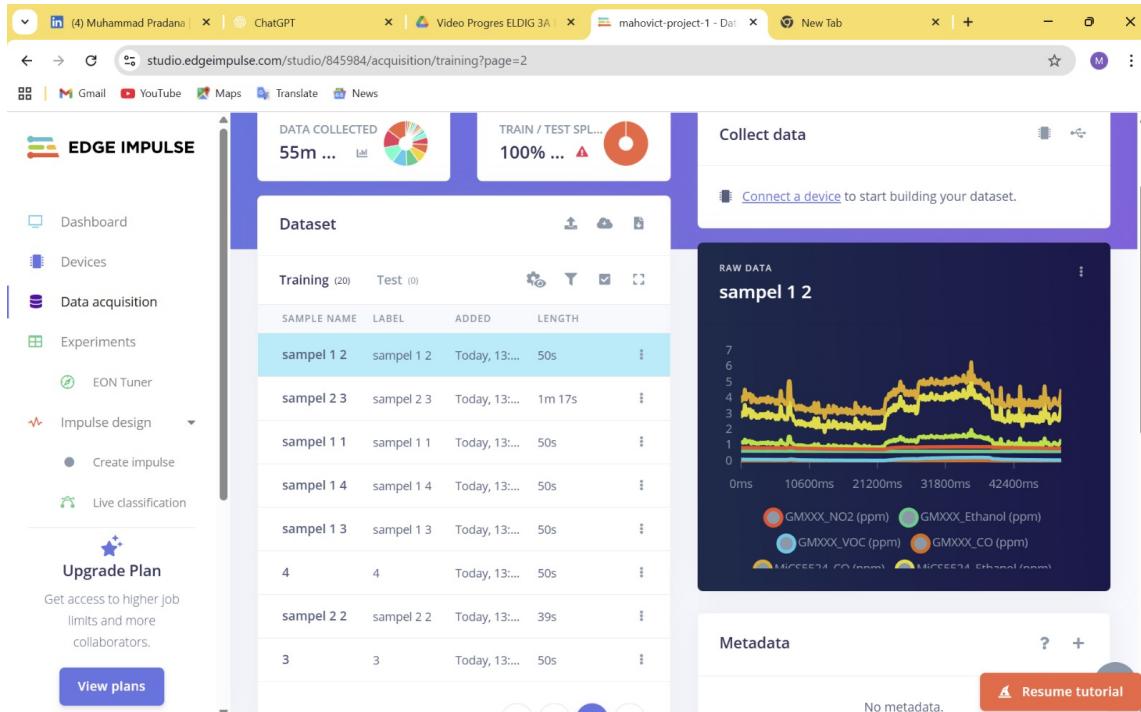


Figure 14: Tampilan Edge Impulse

5 HASIL PENGUJIAN

Bab ini membahas hasil pengujian yang dilakukan untuk memastikan bahwa sistem *e - nose* bekerja sesuai dengan rancangan pada bab sebelumnya. Pengujian dilakukan mulai dari *hardware*, *backend*, proses komunikasi, GUI, sampai integrasi dataset ke *Edge Impulse*. Setiap bagian diuji secara keseluruhan untuk memastikan sistem berjalan sebagai satu kesatuan.

Pengujian fungsional dilakukan dengan memaparkan *array* sensor terhadap sampel sabun cair di dalam ruang uji terkondisi. Berdasarkan tampilan GUI yang telah dikembangkan, sistem berhasil mendeteksi keberadaan perangkat Arduino dan memulai proses *streaming* data tanpa jeda yang terlihat. Indikator numerik pada panel 'GM-XXX Sensors' dan 'MiCS-5524 Sensors' menunjukkan perubahan nilai yang dinamis segera setelah sampel sabun cair didekatkan. Tombol kontrol seperti 'Start Monitor' dan 'Stop Monitor' berfungsi responsif dalam mengendalikan aliran data, dan fitur 'Export CSV' berhasil menyimpan dataset hasil pengujian untuk keperluan analisis lanjutan. Stabilitas koneksi *WebSocket* terbukti handal dalam mengirimkan ratusan titik data per detik tanpa membebani memori tampilan pengguna.

Pengujian sistem dilakukan untuk memastikan bahwa seluruh komponen *e - nose* bekerja sesuai dengan rancangan. Pengujian mencakup *hardware*, *backend*, tampilan GUI, alur komunikasi data, proses penyimpanan, dan integrasi ke *Edge Impulse*. Setiap bagian diuji keseluruhan sebagai satu kesatuan untuk memastikan bahwa sistem bisa dipakai secara stabil saat pengambilan data aroma.

Pengujian dimulai dari *hardware*. Setiap sensor dicek satu per satu untuk memastikan bahwa nilai yang muncul masuk akal dan berubah ketika diberikan aroma tertentu. Sensor MICS dan GMXXX langsung menunjukkan perubahan respons ketika didekatkan ke aroma kuat seperti pada sampel kami (sabun cair). Ketika tidak ada aroma, nilai sensor kembali ke *baseline* setelah beberapa detik. Selama pengujian, *hardware* juga dibiarkan tanpa sampel untuk melihat apakah *baseline* stabil dan tidak ada lonjakan nilai yang tidak wajar. Hasilnya, *hardware* bekerja dengan baik dan sensor memberikan pola respons yang konsisten.

Setelah itu, pengujian berlanjut ke *backend*. Bagian ini memastikan bahwa data dari Arduino dapat terbaca dan diproses tanpa error. *Backend* berhasil membuka *port* serial, membaca JSON yang dikirim Arduino, dan memproses data tersebut secara *real - time*. Pengujian dilakukan menggunakan data asli maupun data simulasi, dan *backend* tetap stabil. Proses *broadcast* melalui *WebSocket* juga tidak menimbulkan *delay* berarti; GUI dapat menerima data dari *backend* dengan lancar, bahkan ketika ada lebih dari satu klien yang terhubung.

Pengujian berikutnya dilakukan pada GUI. Grafik *real - time* berjalan halus dan tidak mengalami *freeze* ketika data masuk secara cepat. Semua tombol seperti *Connect*, *Start*, Stop, *Save*, dan *Upload* bekerja sesuai fungsinya. Nilai sensor yang tampil di GUI cocok dengan data yang dikirim oleh *backend*, dan status sistem seperti jumlah data dan mode sampling juga muncul dengan benar. Secara keseluruhan, GUI responsif dan nyaman digunakan saat proses pengambilan data.

Pengujian komunikasi antara Arduino, *backend*, dan GUI menunjukkan bahwa alur data berjalan stabil. *Delay* dari sensor sampai grafik tampil di GUI hanya berada pada kisaran puluhan milidetik sehingga data terasa *real - time*. Tidak ditemukan data hilang selama proses *streaming* beberapa menit. Ketika Arduino dilepas atau *port* berubah, sistem dapat mendeteksi kondisi tersebut dan memberi pesan status sehingga pengguna dapat melakukan *reconnect* tanpa harus menutup aplikasi.

Pengujian penyimpanan dilakukan untuk memastikan bahwa data yang dikumpulkan dapat disimpan dalam format CSV secara rapi. GUI berhasil membuat file CSV dengan struktur kolom yang konsisten dan mudah dibaca oleh aplikasi lain. File CSV yang dihasilkan bisa dibuka di Excel maupun diproses menggunakan Python tanpa error. GUI juga memberikan peringatan saat terjadi kondisi yang tidak valid, seperti ketika nama sampel belum diisi atau data masih kosong.

Tahap terakhir adalah pengujian integrasi ke *Edge Impulse*. Dataset CSV yang dihasilkan dapat diunggah tanpa perlu konversi tambahan. Sinyal yang muncul di tampilan *Edge Impulse* sesuai dengan bentuk asli dari grafik *real - time*. Proses *preprocessing* seperti *windowing*, normalisasi, dan ekstraksi fitur berjalan lancar. Hal ini menunjukkan bahwa dataset dari sistem *e - nose* sudah kompatibel dan bisa dipakai untuk pelatihan model *machine learning*.

Secara keseluruhan, hasil pengujian menunjukkan bahwa sistem *e - nose* berjalan stabil dan dapat digunakan untuk proses akuisisi data aroma. *Hardware*, *backend*, GUI, dan alur komunikasi bekerja dengan baik, sedangkan proses penyimpanan dan integrasi data sudah sesuai kebutuhan analisis lebih lanjut. Sistem ini telah memenuhi tujuan perancangan dan siap digunakan sebagai alat pengumpul data maupun penelitian lanjutan.

6 VISUALISASI GRAFIK

Bab ini membahas visualisasi dan analisis sinyal yang dihasilkan oleh sistem *e - nose* selama proses pengambilan data aroma. Setelah sistem dipastikan berfungsi dengan baik pada bab sebelumnya, tahap berikutnya adalah melihat bentuk sinyal yang muncul pada setiap sensor ketika mendeteksi sampel sabun cair. Visualisasi ini membantu memahami pola respons sensor, membandingkan karakteristik aroma, dan menilai apakah data yang terkumpul cukup jelas dan konsisten untuk digunakan pada tahap pemrosesan lebih lanjut, termasuk pelatihan model di *Edge Impulse*. Melalui grafik - grafik yang ditampilkan, dapat terlihat bagaimana perubahan konsentrasi aroma tercermin dalam bentuk kurva, seberapa stabil *baseline* sensor, serta bagaimana perbedaan pola muncul di antara berbagai jenis sampel.

Visualisasi data pada GUI menampilkan respons sinyal yang signifikan dari sampel sabun cair, sebagaimana terlihat pada grafik *real - time* Sensor Data. Grafik terbagi menjadi dua plot utama untuk membedakan karakteristik sensor. Pada pengujian sabun cair ini, terlihat jelas adanya kenaikan amplitudo pada kurva sensor yang sensitif terhadap alkohol dan senyawa organik yang mudah menguap (VOC), seperti sensor GM - Ethanol (kurva hijau) dan MiCS- Ethanol (kurva ungu kemerah). Grafik menunjukkan pola *time - series* yang khas; fase *baseline* yang stabil sebelum paparan, diikuti oleh fase *response* berupa kenaikan eksponensial saat aroma sabun cair dideteksi, dan fase *recovery* (peluruhan) saat sampel dijauhkan atau sistem dihentikan. Garis - garis grafik yang halus menunjukkan bahwa

frekuensi sampling yang diterapkan sudah memadai untuk menangkap fenomena transien dari sensor gas.

6.1 Grafik Gnuplot

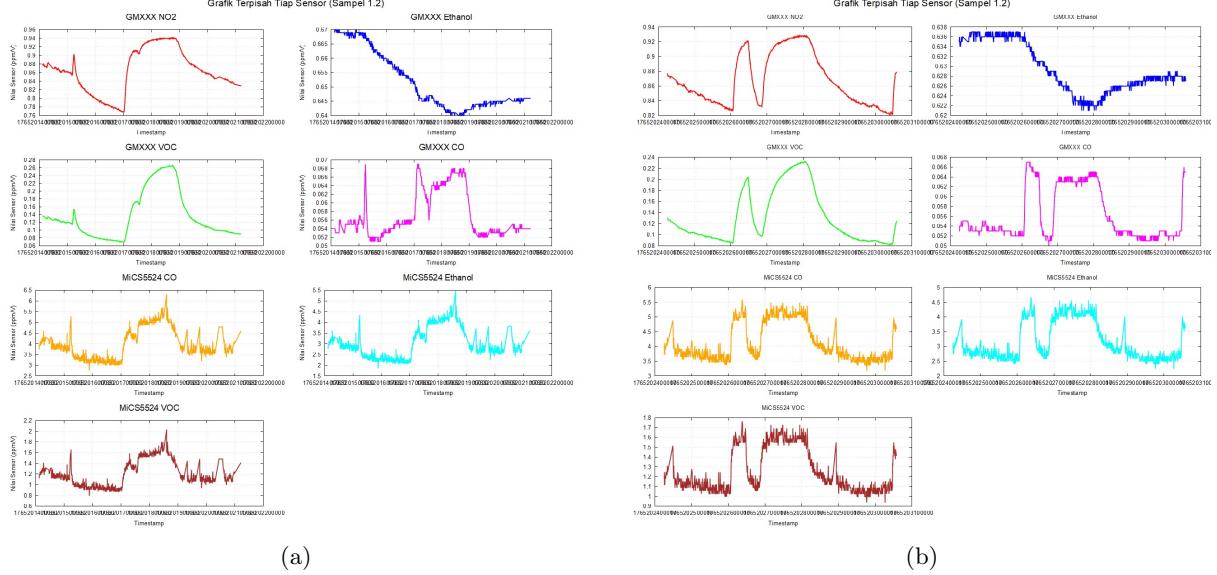


Figure 15: Grafik Hasil Sampel 1

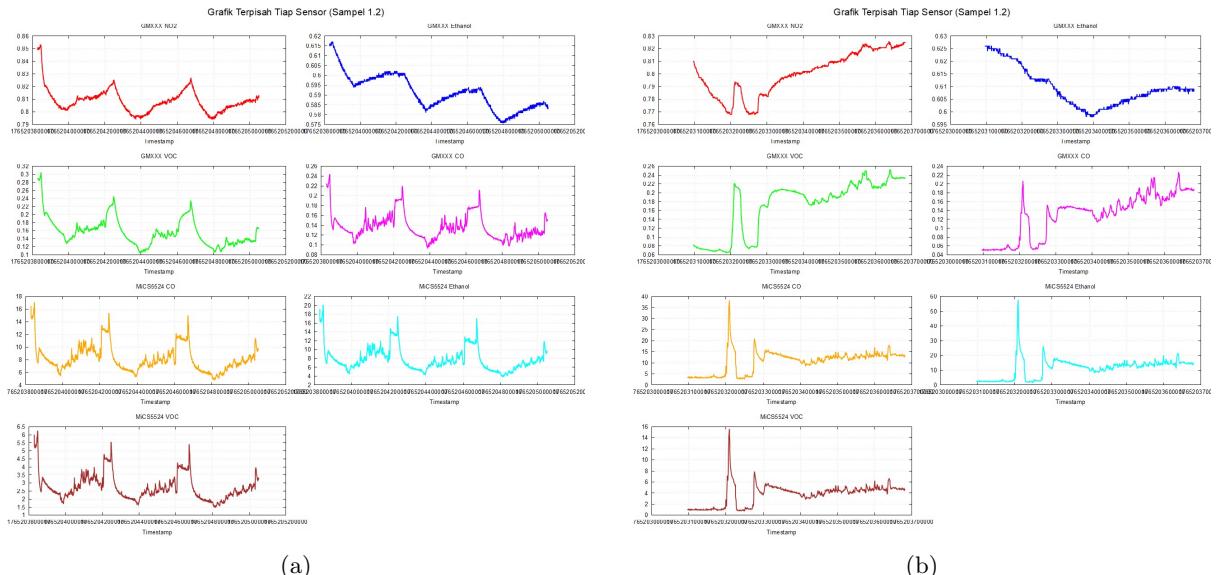
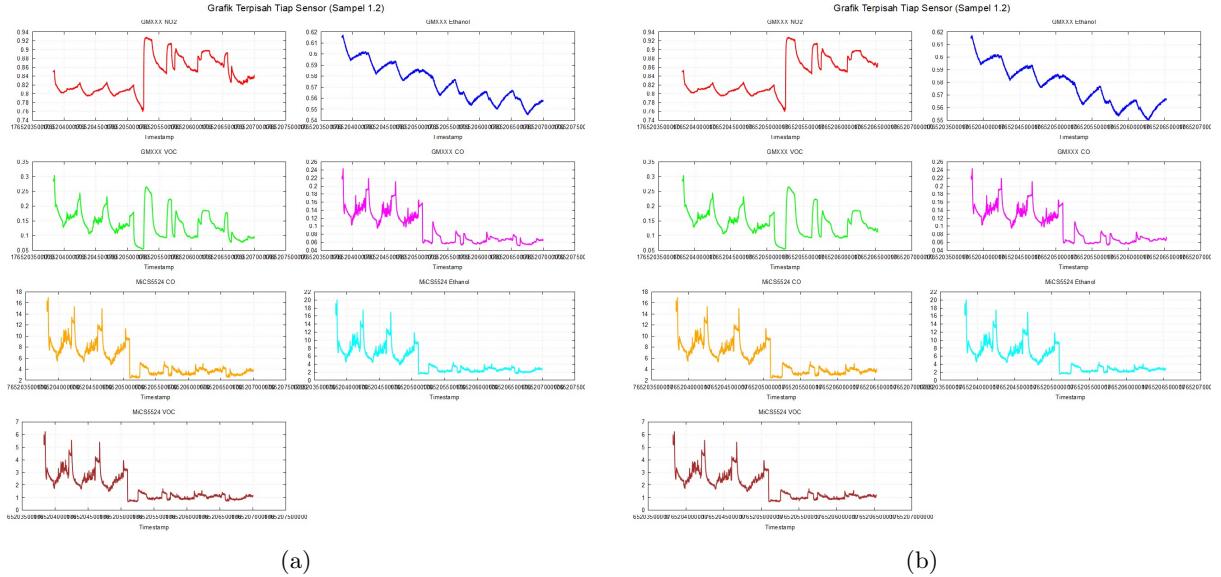
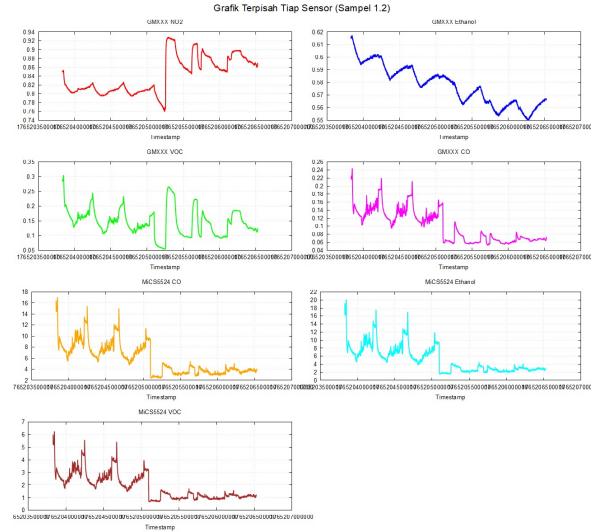


Figure 16: Grafik Hasil Sampel 2

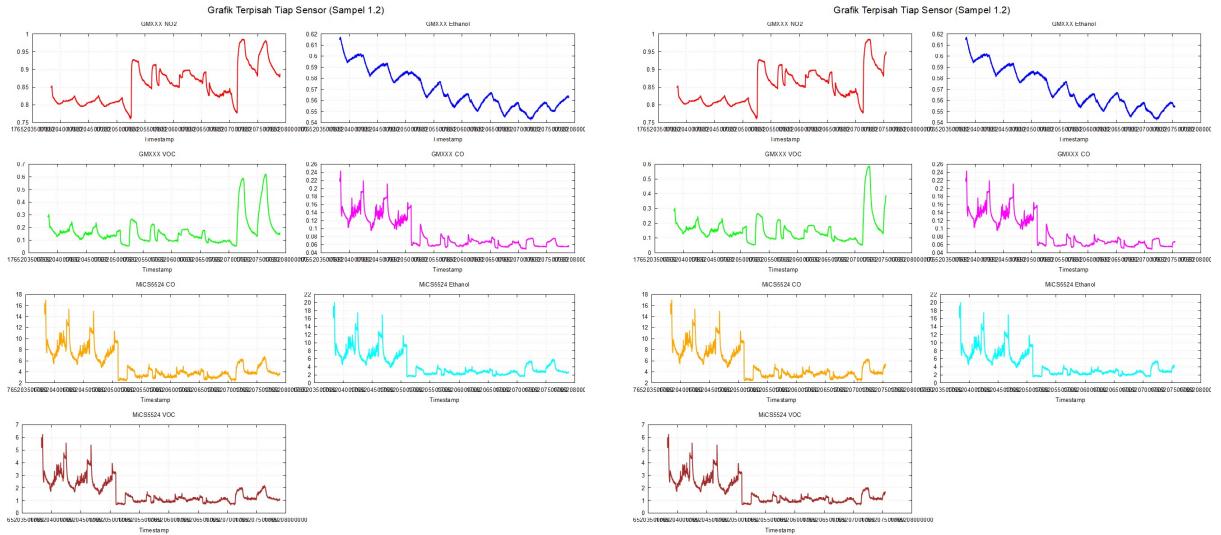


(a)

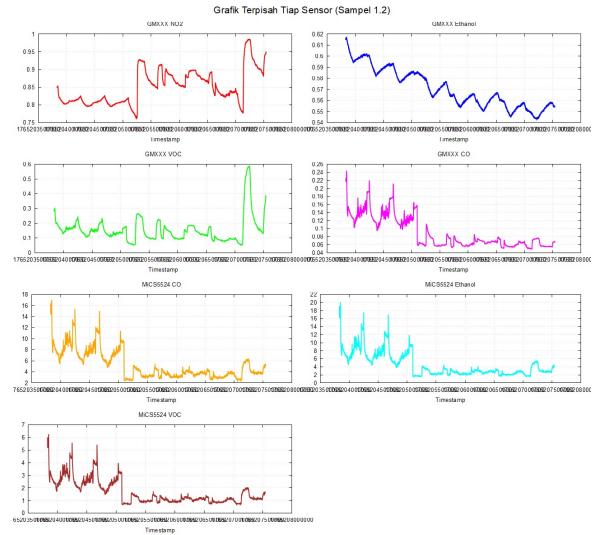


(b)

Figure 17: Grafik Hasil Sampel 3



(a)



(b)

Figure 18: Grafik Hasil Sampel 4

6.2 Grafik Edge Impulse

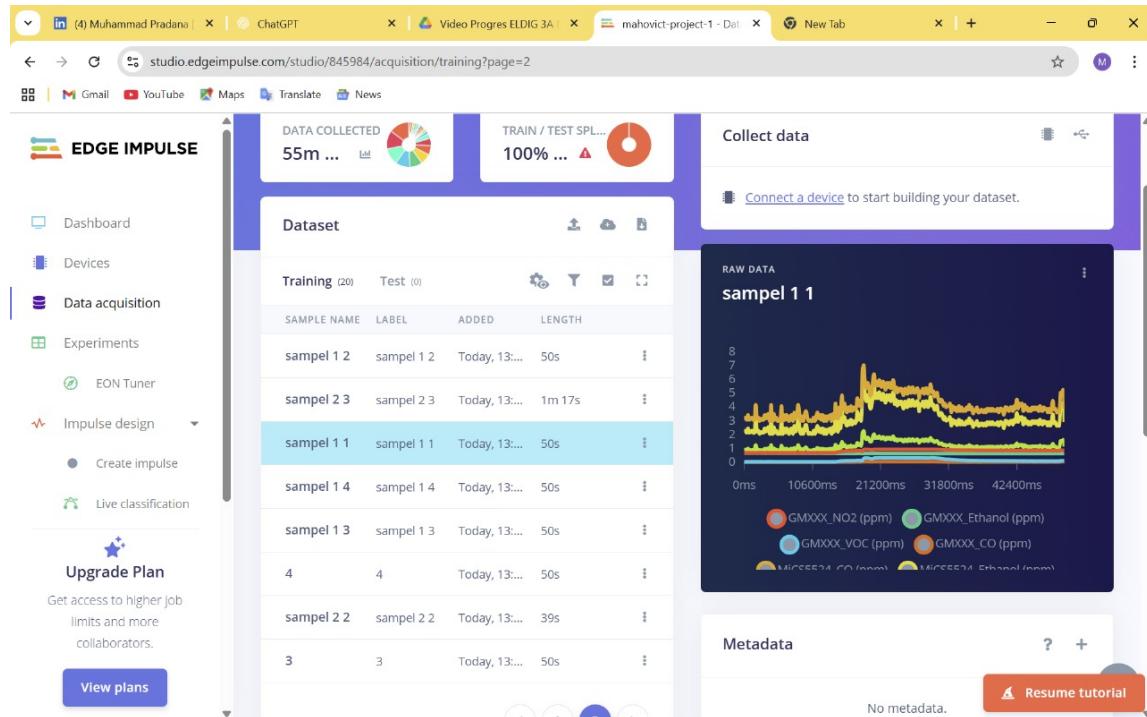


Figure 19: Grafik Hasil Sampel 1.1

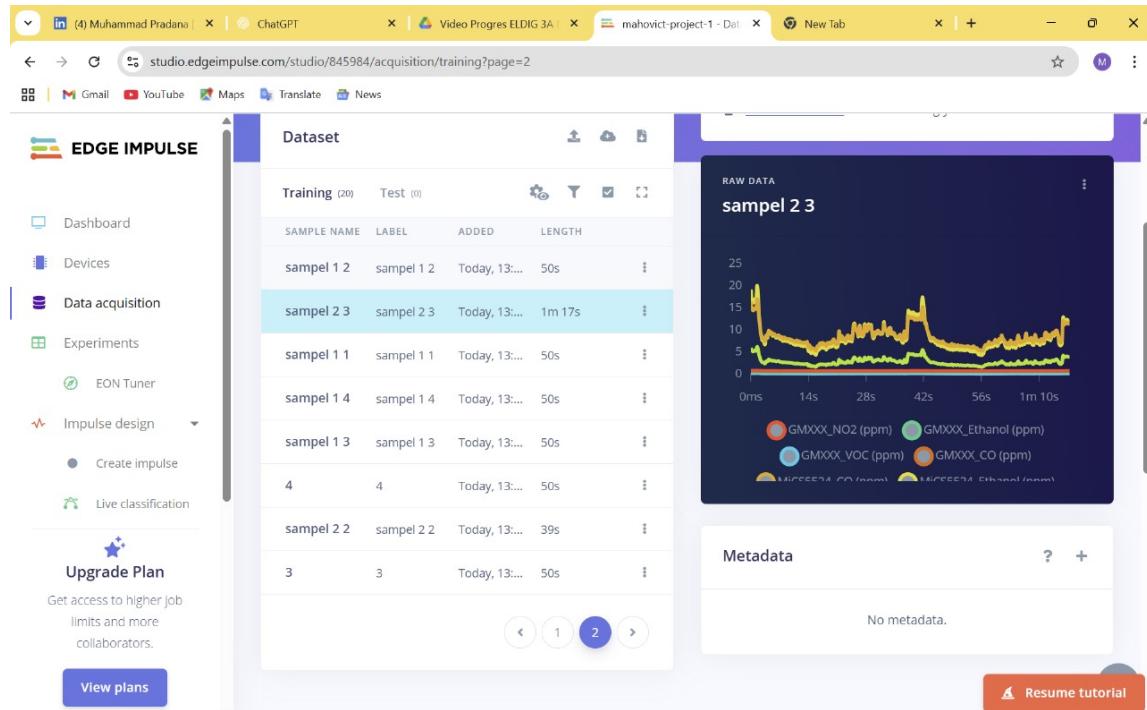


Figure 20: Grafik Hasil Sampel 2.3

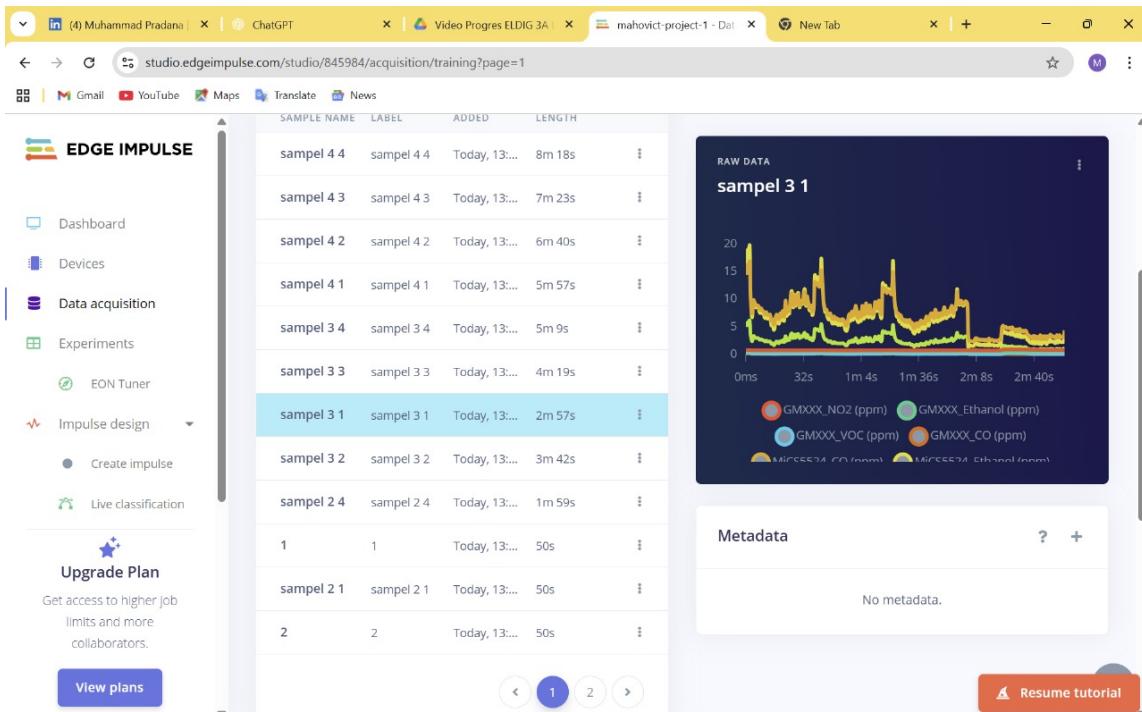


Figure 21: Grafik Hasil Sampel 3.1

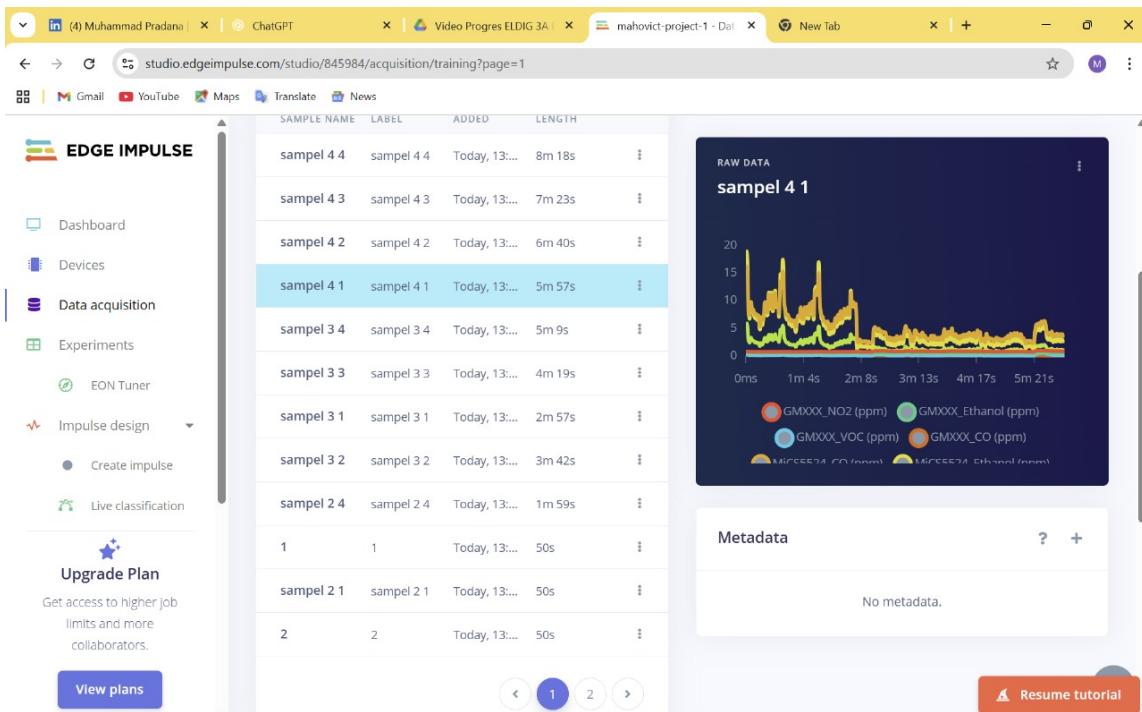


Figure 22: Grafik Hasil Sampel 4.1

6.3 Grafik Tampilan GUI

6.3.1 Dengan Satu Pompa



Figure 23: Grafik Hasil Sampling 1.1



Figure 24: Grafik Hasil Sampling 2.1



Figure 25: Grafik Hasil Sampling 3.1



Figure 26: Grafik Hasil Sampling 4.1

6.3.2 Dengan Dua Pompa

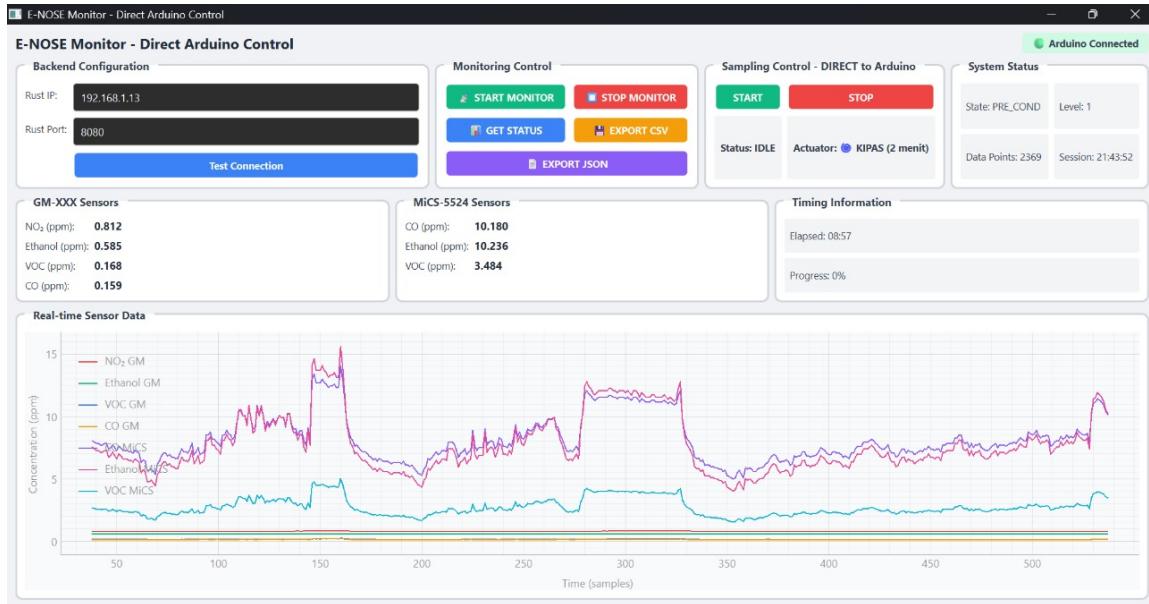


Figure 27: Grafik Hasil Sampling 1.2

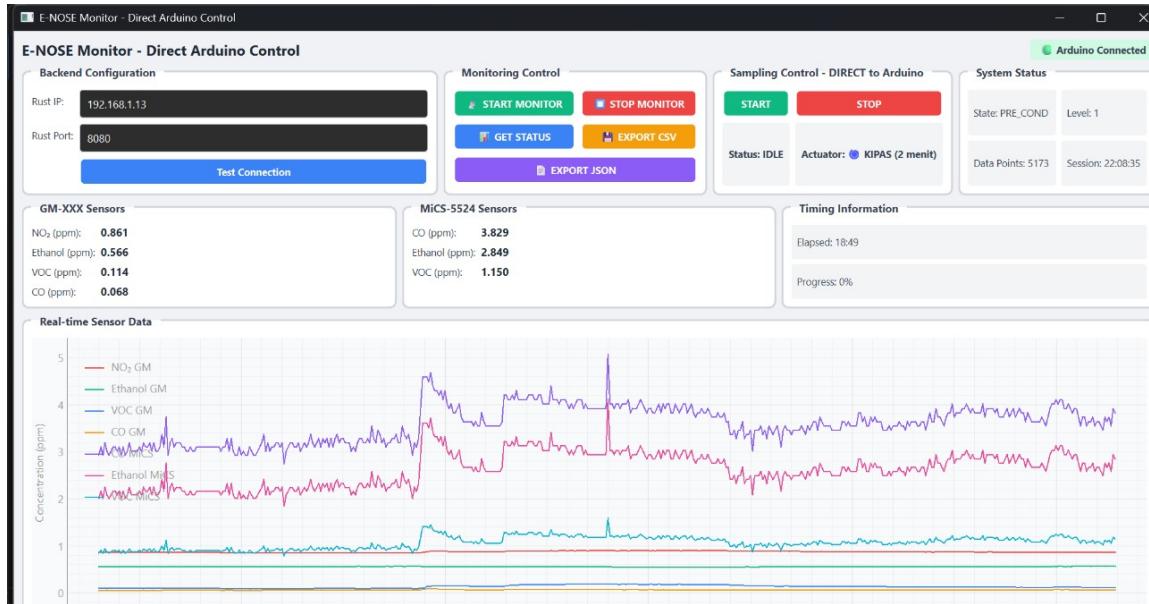


Figure 28: Grafik Hasil Sampling 2.2

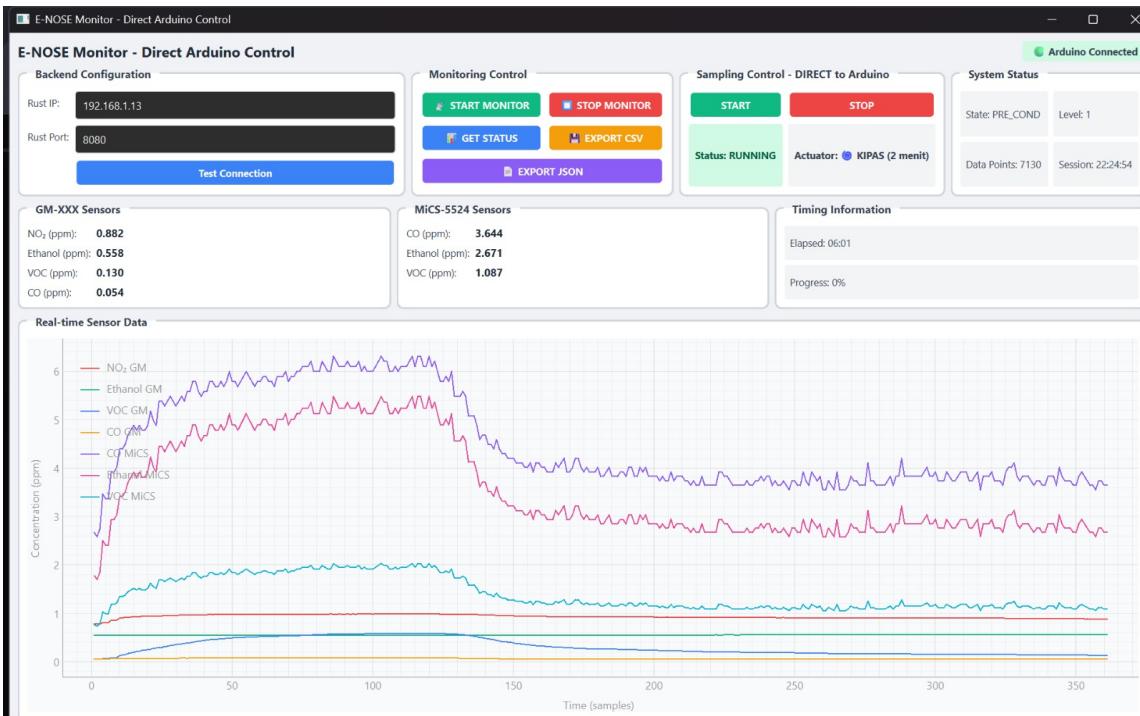


Figure 29: Grafik Hasil Sampling 3.2



Figure 30: Grafik Hasil Sampling 4.2

7 DESAIN 3D

Bab ini menjelaskan bagaimana desain 3D dibuat untuk menampung seluruh komponen sistem *e - nose*, mulai dari sensor, Arduino, hingga modul pendukung lainnya. Desain 3D diperlukan agar perangkat lebih rapi, mudah dibawa, dan menjaga posisi sensor tetap konsisten selama proses pengambilan data aroma. Dengan wadah yang terstruktur, aliran udara ke sensor juga bisa diatur lebih baik, sehingga hasil pengukuran menjadi lebih stabil.

Desain dibuat menggunakan *software* permodelan 3D yang memungkinkan pengaturan ukuran secara presisi. Bentuk *casing* dirancang sederhana agar mudah dicetak menggunakan printer 3D. Bagian dalam *casing* diberi ruang untuk masing - masing sensor, jalur kabel, serta dudukan Arduino. Selain itu,

dibuat lubang udara pada sisi bawah sebagai jalur masuk aroma. Lubang ini memastikan udara mengalir langsung ke permukaan sensor tanpa terhalang komponen lain. Bagian luar *casing* juga dilengkapi lubang kecil untuk *port USB* Arduino agar perangkat bisa dihubungkan ke laptop tanpa membuka penutupnya.

Beberapa versi desain biasanya dibuat sebelum menentukan bentuk akhir. Proses ini dilakukan agar semua komponen muat dengan baik dan tidak saling bertumpukan. Model 3D juga diperiksa apakah kuat saat dicetak dan apakah *casing* tidak terlalu tipis.

Desain 3D ini menjadi wadah utama yang menyatukan seluruh sistem. Dengan adanya *casing*, perangkat terlihat lebih bersih, kokoh, dan lebih nyaman digunakan saat proses pengambilan data aroma.

Untuk mendukung proses akuisisi data yang optimal, dirancang sebuah *chamber* (ruang sampel) 3D menggunakan perangkat lunak CAD (Fusion 360). Desain mekanik ini bertujuan untuk mengisolasi sensor dari gangguan udara ambien dan memusatkan konsentrasi aroma sabun cair ke permukaan sensor. *Casing* dirancang dengan mempertimbangkan aliran udara (*airflow*) yang laminar agar gas dapat menyebar merata ke seluruh *array* sensor dan dapat dibersihkan (*di-purge*) dengan cepat setelah pengambilan sampel selesai. Struktur desain juga mengakomodasi dudukan untuk Arduino Uno R4 dan modul sensor agar terpasang kokoh, meminimalisir *noise* sinyal akibat getaran mekanik. Implementasi fisik dari desain ini sangat krusial untuk memastikan reproduksibilitas data pada setiap pengambilan sampel sabun cair.

7.1 Gambar 3D

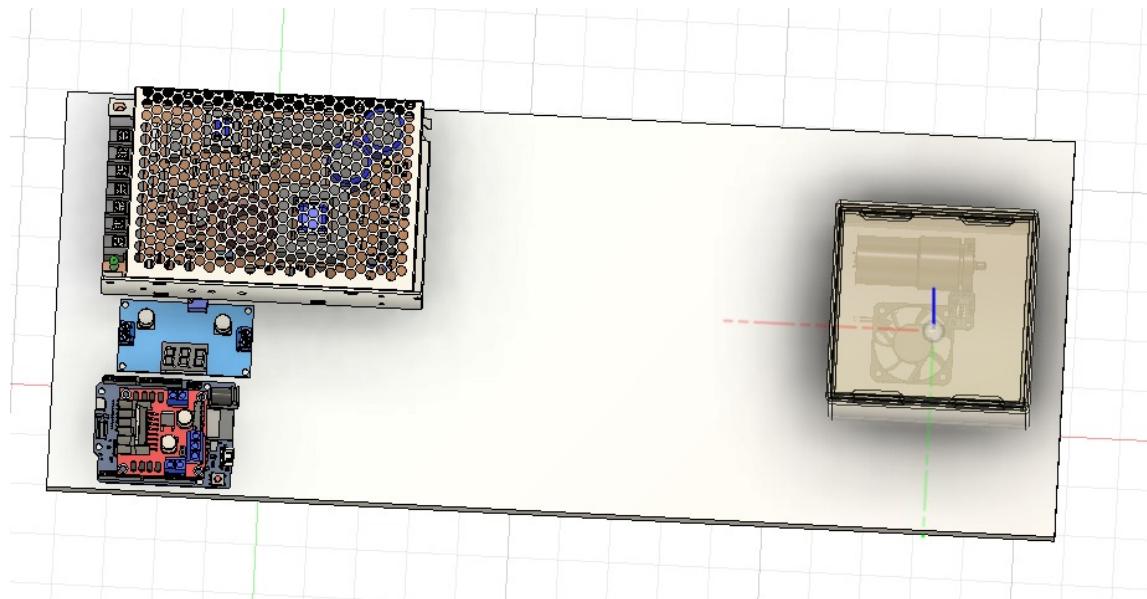


Figure 31: Tampak Atas Dengan Satu Pompa

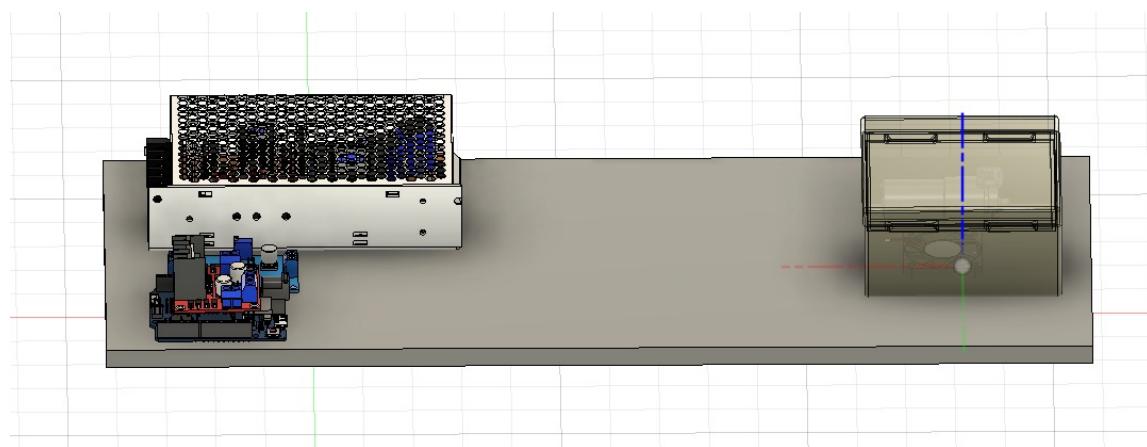


Figure 32: Tampak Depan Dengan Satu Pompa

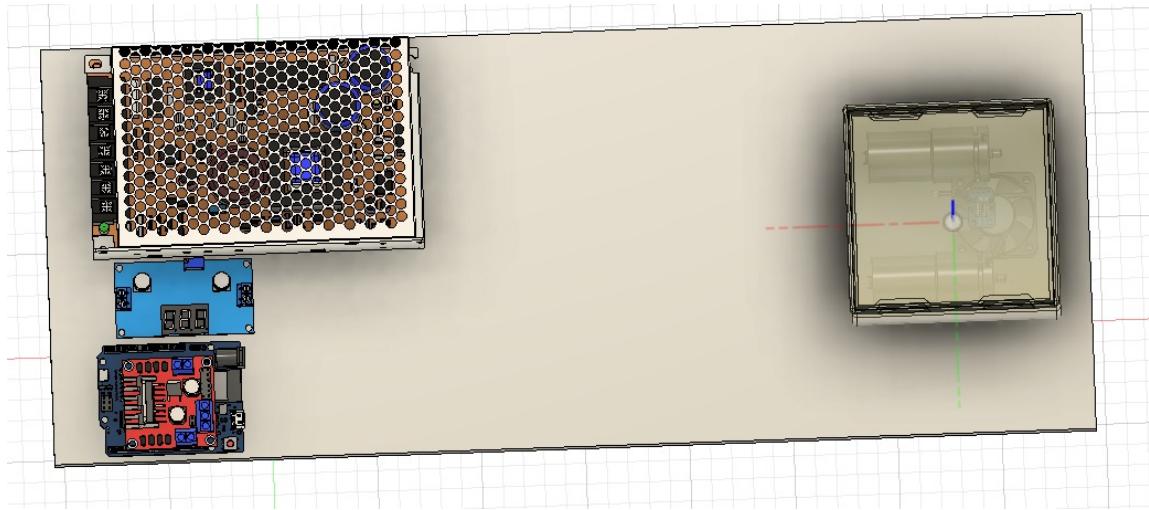


Figure 33: Tampak Atas Dengan Dua Pompa

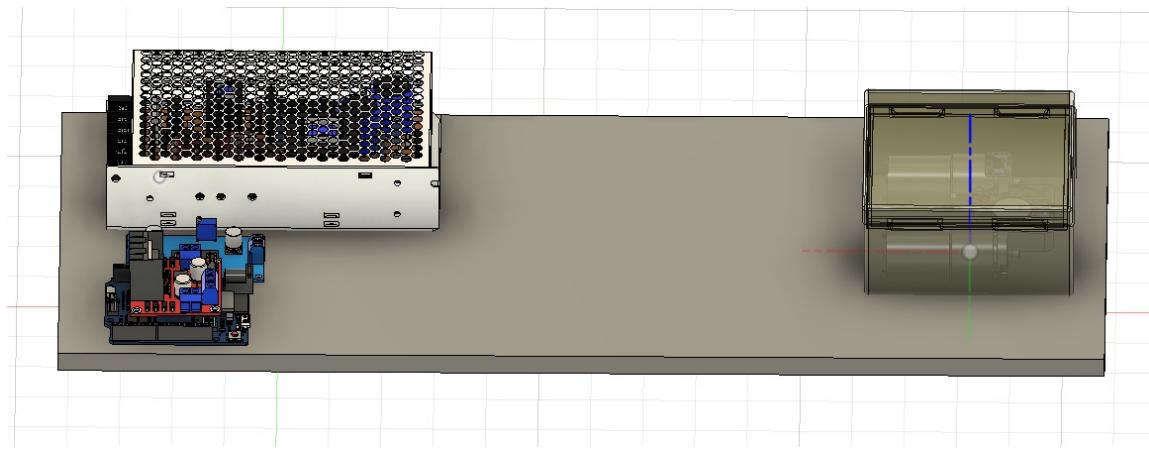


Figure 34: Tampak Depan Dengan Dua Pompa

8 ANALISIS DATA

Bab ini membahas bagaimana data sensor yang terkumpul diolah dan dianalisis untuk memahami pola respons aroma. Setelah proses pengambilan data selesai, setiap sampel dibersihkan, diperiksa, dan dibandingkan untuk melihat karakteristik masing - masing sensor. Analisis ini penting untuk mengetahui apakah pola sinyal cukup konsisten, apakah perbedaan antar aroma terlihat jelas, dan apakah data yang dihasilkan layak digunakan untuk pelatihan model *machine learning*.

Langkah pertama yang dilakukan adalah mengecek kondisi data mentah. File CSV yang telah dieksplor dari GUI diperiksa untuk memastikan tidak ada nilai kosong, *timestamp* tidak meloncat, dan struktur kolom sudah sesuai. Data kemudian divisualisasikan ulang menggunakan grafik garis sederhana untuk memastikan bentuk sinyalnya konsisten dengan grafik *real - time* pada GUI. Dari proses ini, terlihat bahwa setiap aroma menghasilkan pola respons yang berbeda, baik dari durasi peningkatan sinyal maupun ketinggian puncaknya.

Setelah data dinyatakan bersih, analisis dilanjutkan dengan melihat pola setiap sensor. Sensor MICS cenderung menunjukkan respons yang lebih tajam terhadap aroma VOC yang kuat, sedangkan sensor GMXXX memberikan respons yang lebih stabil tetapi naik perlahan. Perbedaan pola ini membantu mengenali karakter aroma, karena beberapa sampel menunjukkan puncak respons yang lebih cepat, sedangkan yang lainnya menghasilkan kurva yang lebih datar. Analisis ini juga membantu menentukan sensor mana yang paling berkontribusi terhadap identifikasi aroma.

Untuk melengkapi evaluasi, data kemudian diperiksa menggunakan *tools* di *Edge Impulse*. Pada tahap ini, bentuk sinyal yang masuk ke jendela *preprocessing* terlihat serupa dengan sinyal asli di CSV, yang menandakan bahwa format dataset sudah sesuai. Ekstraksi fitur memberikan gambaran mengenai

seberapa 'berbeda' sinyal antar aroma secara numerik. Jika perbedaan fitur cukup jelas, artinya dataset potensial untuk dilatih menjadi model klasifikasi aroma.

Berdasarkan data sinyal yang terekam pada GUI, sampel sabun cair menghasilkan jejak aroma (*smell print*) yang didominasi oleh respons sensor VOC dan Etanol. Hal ini logis mengingat sabun cair mengandung zat pewangi (*fragrance*) yang bersifat volatil. Pada grafik, terlihat bahwa sensor MiCS - 5524 memberikan respons yang lebih sensitif terhadap perubahan konsentrasi mendadak dibandingkan beberapa kanal pada GM - XXX, ditandai dengan slope (kemiringan) grafik yang lebih curam pada fase awal deteksi. Variasi sinyal antar sensor ini (*cross-sensitivity*) adalah data fundamental yang diperlukan untuk ekstraksi fitur. Data *time - series* yang diperoleh dari eksperimen ini menunjukkan pola deterministik yang konsisten, sehingga sangat layak untuk digunakan sebagai dataset pelatihan model *machine learning* pada platform *Edge Impulse* guna klasifikasi jenis atau merek sabun cair secara otomatis.

Berdasarkan keseluruhan analisis, data yang dihasilkan sistem *e - nose* memiliki kualitas yang baik dan menunjukkan pola yang konsisten. Setiap aroma menghasilkan bentuk sinyal yang bisa dibedakan secara visual maupun numerik. Hal ini menunjukkan bahwa perangkat, proses sampling, dan *pipeline* data sudah bekerja dengan benar sehingga dataset siap digunakan untuk tahap pemodelan atau pengembangan sistem lebih lanjut.

8.1 Analisa Grafik Gnuplot

Berdasarkan grafik yang dihasilkan dari keempat sampel, terlihat bahwa masing - masing sampel memberikan pola respons yang berbeda pada hampir semua sensor. Pada Sampel 1, respons sensor cenderung menunjukkan pola peningkatan yang jelas di awal pengukuran. Kanal MICS VOC dan etanol naik dengan cepat, mencapai puncak yang cukup tinggi sebelum perlahan menurun menuju *baseline*. Hal ini menggambarkan bahwa Sampel 1 memiliki komponen volatil yang cukup kuat dan mudah terdeteksi. Sensor GMXXX juga menunjukkan kenaikan yang stabil, meskipun dengan amplitudo yang lebih rendah dibandingkan sensor MICS. Secara umum, Sampel 1 memiliki pola respons yang paling bersih dan terstruktur, dengan puncak yang tajam namun mudah diidentifikasi.

Berbeda dengan itu, Sampel 2 memperlihatkan pola yang lebih bervariasi. Beberapa kanal menunjukkan kenaikan pendek lalu cepat turun, sedangkan kanal lainnya memiliki puncak yang lebih panjang. Hal ini menandakan bahwa kecepatan difusi dan konsentrasi aroma pada Sampel 2 kemungkinan tidak setinggi Sampel 1, atau mengandung senyawa volatil yang bereaksi lebih lambat pada sensor tertentu. Pada grafik GMXXX, respons terlihat lebih datar di awal kemudian meningkat berkala, yang menunjukkan perubahan konsentrasi aroma secara bertahap.

Sementara itu, Sampel 3 memiliki karakteristik yang cukup berbeda, terutama pada sensor MICS. Pada sampel ini, grafik menunjukkan pola kenaikan yang lebih rendah dan lebih 'bergerigi', menandakan bahwa aroma yang dideteksi mungkin memiliki konsentrasi volatil yang lebih rendah atau tidak stabil. Tren yang muncul adalah kenaikan kecil, diikuti penurunan bertahap, sehingga bentuk sinyalnya terlihat lebih datar dibandingkan Sampel 1 dan 2. Sensor GMXXX memperlihatkan respons yang serupa dengan pola '*step - down*', yaitu puncak yang kecil namun konsisten turun seiring waktu. Hal ini memberi kesan bahwa Sampel 3 memiliki intensitas aroma paling lemah atau paling cepat menguap sehingga sensor tidak mempertahankan puncak respons lama.

Untuk Sampel 4, pola respons terlihat lebih kompleks dibandingkan tiga sampel sebelumnya. Pada banyak kanal, terutama MICS VOC dan GMXXX CO, muncul beberapa puncak berturut - turut yang menandakan perubahan aroma yang tidak stabil atau adanya beberapa komponen volatil yang terdeteksi secara bergantian. Grafik juga menunjukkan bahwa sensor membutuhkan waktu lebih lama untuk kembali menuju *baseline*, yang bisa terjadi ketika aroma memiliki komponen yang lebih 'berat' atau cenderung menempel lebih lama pada permukaan sensor. Pola puncak bertingkat yang muncul pada Sampel 4 menjadi indikator bahwa sampel ini mungkin mengandung campuran aroma atau konsentrasi yang lebih pekat.

Jika dibandingkan secara keseluruhan, Sampel 1 cenderung memiliki respons paling tajam dan konsisten, Sampel 2 memiliki respons sedang dengan pola bertahap, Sampel 3 menunjukkan respons paling lemah dan cepat mereda, sedangkan Sampel 4 menampilkan pola yang lebih kompleks dan berkali-kali naik turun. Perbedaan pola ini menunjukkan bahwa setiap sampel memiliki karakter volatil yang berbeda, dan sistem *e - nose* berhasil menangkap variasi tersebut melalui kombinasi sensor yang digunakan.

8.2 Analisa Grafik Edge Impulse

Secara umum, keempat sampel ini menunjukkan data sensor mentah yang berasal dari sensor gas, kemungkinan besar mengukur kadar NO₂ (nitrogen dioksida), VOC (*Volatile Organic Compounds*), CO (karbon monoksida), dan Etanol (*Ethanol*) dalam satuan ppm (*parts per million*) atau nilai sensor yang sebanding. Perbedaan utama terletak pada pola, amplitudo, dan jenis gas yang paling dominan dalam setiap sampel, yang mengindikasikan kondisi lingkungan atau peristiwa yang berbeda saat data diambil.

Sampel 1.1, grafik menunjukkan pola yang relatif stabil dan berulang, dengan beberapa puncak yang jelas. Amplitudo untuk NO₂ dan Etanol tampak cukup tinggi dan fluktuatif, sedangkan VOC dan CO juga hadir tetapi dengan puncak yang lebih rendah. Sampel ini mungkin mewakili kondisi lingkungan yang mengalami paparan polutan gas tertentu secara intermiten atau kondisi yang bervariasi.

Sampel 2.3, dibandingkan dengan sampel 1.1, sampel 2.3 menunjukkan pola yang lebih terkonsentrasi atau terdistribusi merata di waktu pengamatan. Amplitudo gas - gas utama, terutama NO₂ dan Etanol, terlihat cukup tinggi dan mungkin sedikit lebih stabil daripada sampel 1.1. Ini bisa mengindikasikan paparan gas yang berkelanjutan atau kondisi lingkungan yang lebih konstan.

Sampel 3.1, grafik ini tampak menunjukkan amplitudo yang paling rendah di antara keempat sampel, terutama untuk gas seperti NO₂ dan Etanol. Garis data untuk semua gas cenderung berada di bagian bawah grafik, mendekati nol, dengan puncak yang minimal atau tidak setinggi sampel lainnya. Sampel 3.1 kemungkinan besar mewakili kondisi lingkungan yang paling bersih atau minim paparan gas-gas tersebut.

Sampel 4.1, sampel ini menunjukkan pola yang sangat tinggi dan fluktuatif untuk semua gas yang diukur, terutama NO₂ dan Etanol, mencapai puncak yang signifikan dan konsisten. Puncak - puncak tersebut tampak lebih tajam dan lebih sering. Sampel 4.1 kemungkinan mewakili kondisi lingkungan yang mengalami paparan gas yang paling intens atau terjadi peristiwa signifikan yang melepaskan konsentrasi gas tinggi. Misalnya, aktivitas pembakaran, kebocoran, atau paparan langsung ke sumber polutan.

Jadi sampel 3.1 menunjukkan lingkungan paling bersih, sampel 4.1 menunjukkan paparan gas paling intens, sementara sampel 1.1 dan 2.3 berada di tengah, dengan sedikit perbedaan dalam pola dan konsentrasi puncaknya. Perbedaan ini sangat penting dalam konteks pelatihan model *machine learning* di *Edge Impulse* karena akan menjadi dasar bagi model untuk membedakan antara kondisi 'normal' dan 'tercemar' atau kondisi spesifik lainnya.

8.3 Grafik Tampilan GUI

Perbedaan antara hasil sampling menggunakan Satu Pompa dan Dua Pompa terletak pada efisiensi pengumpulan gas dan responsivitas sensor. Pada kondisi Satu Pompa, grafik cenderung menunjukkan pola yang lebih halus dan perubahan konsentrasi yang lebih landai (respons yang lebih lambat), serta konsentrasi puncak yang terukur berada pada tingkat yang relatif lebih rendah. Sebaliknya, penggunaan Dua Pompa menghasilkan grafik dengan puncak konsentrasi yang jauh lebih tinggi dan tajam. Misalnya, konsentrasi CO melonjak dari sekitar 2.119 ppm menjadi 10.103 ppm serta menunjukkan respons sensor yang lebih cepat terhadap perubahan lingkungan, ditandai dengan perubahan garis yang lebih vertikal dan mendadak. Hal ini mengindikasikan bahwa dua pompa bekerja lebih efektif dalam menarik dan mengumpulkan sampel gas, membuat sensor menjadi lebih sensitif dan mampu menangkap tingkat konsentrasi gas yang sesungguhnya dengan lebih intensif.

9 KESIMPULAN

Proyek ini berhasil mengembangkan sistem *e - nose* yang terintegrasi penuh, mulai dari akuisisi data *hardware* hingga visualisasi *software*, sesuai dengan spesifikasi tugas PBL. Kombinasi *backend Rust* dan *frontend Qt Python* terbukti menghasilkan sistem instrumentasi yang responsif, stabil, dan mampu menangani aliran data sensor *real - time* dengan efisien. Pengujian menggunakan sampel sabun cair menunjukkan bahwa sistem mampu membedakan respons sensor terhadap senyawa volatil spesifik, yang divisualisasikan dengan jelas melalui grafik dinamis pada GUI. Secara keseluruhan, sistem ini tidak hanya memenuhi persyaratan akademis Mata Kuliah Sistem Pengolahan Sinyal, tetapi juga memiliki potensi aplikatif sebagai prototipe alat uji kualitas produk berbasis aroma.

Dari seluruh proses perancangan, implementasi, pengujian, dan analisis data, beberapa kesimpulan yang dapat diambil adalah:

1. Sistem *e - nose* berhasil dibangun menggunakan kombinasi sensor MICS dan GMXXX, Arduino Uno R4 WiFi, *backend Rust*, dan GUI berbasis PyQt6.

2. *Hardware* bekerja stabil dan mampu mendeteksi perubahan aroma melalui respons sensor yang konsisten.
3. *Backend* dapat membaca data serial, memproses JSON, dan mengirimkan data ke GUI secara *real-time* tanpa kehilangan paket.
4. GUI mampu menampilkan grafik sensor dengan halus, menunjukkan nilai sensor secara langsung, dan menyediakan fitur penyimpanan data yang mudah digunakan.
5. Alur komunikasi antara Arduino, *backend*, dan GUI berjalan lancar dengan *delay* yang kecil, sehingga proses pengambilan data terasa responsif.
6. Data yang disimpan dalam bentuk CSV memiliki struktur yang rapi dan kompatibel dengan *Edge Impulse* tanpa perlu konversi tambahan.
7. Visualisasi grafik menunjukkan bahwa setiap aroma menghasilkan pola sinyal yang dapat dibedakan, baik dari tinggi puncak, bentuk kurva, maupun waktu respons.
8. Hasil analisis fitur di *Edge Impulse* mendukung bahwa dataset cukup baik untuk digunakan dalam proses klasifikasi atau pembuatan model *machine learning*.
9. Secara keseluruhan, sistem memenuhi tujuan awal penelitian, yaitu menyediakan perangkat *e-nose* yang dapat digunakan untuk pengambilan data aroma secara stabil, mudah dioperasikan, dan siap untuk dikembangkan lebih lanjut.

9.1 Saran dan Penutup

Sistem *e-nose* yang dibuat masih memiliki banyak ruang untuk dikembangkan. Ke depannya, penambahan jenis sensor gas dapat membantu memperkaya pola aroma sehingga proses identifikasi menjadi lebih akurat. *Casing* 3D juga bisa diperbaiki agar aliran udara lebih terkontrol dan posisi sensor tetap stabil saat pengukuran. Dari sisi *software*, GUI dapat dilengkapi fitur analisis sederhana seperti *smoothing* atau *filtering*, sehingga grafik terlihat lebih bersih sejak awal. Integrasi dengan *Edge Impulse* juga bisa dibuat lebih otomatis, misalnya dengan menambahkan fitur unggah atau pelatihan model langsung melalui aplikasi. Selain itu, pengambilan data sebaiknya dilakukan dengan lebih banyak sampel aroma untuk membangun dataset yang lebih kuat dan bervariasi. Penggunaan mikrokontroler yang lebih kuat atau modul komunikasi nirkabel juga dapat dipertimbangkan agar perangkat menjadi lebih portabel dan mudah digunakan di berbagai kondisi.

Kami mengucapkan terima kasih kepada semua pihak yang telah membantu selama proses pengembangan proyek dan penyusunan laporan ini. Terima kasih kepada dosen pengampu yang telah memberikan bimbingan dan kesempatan untuk mengembangkan sistem *e-nose* ini. Kami juga berterima kasih kepada teman-teman yang telah membantu dalam pengujian perangkat, memberikan masukan, dan berdiskusi selama proses pengembangan berlangsung. Dukungan dari keluarga dan lingkungan sekitar turut memberikan motivasi sehingga proyek ini dapat diselesaikan dengan baik. Tanpa bantuan berbagai pihak, laporan dan sistem ini tidak dapat terselesaikan sebagaimana mestinya.

Daftar Pustaka

References

- [1] Z. Ye, Y. Liu, and Q. Li, “Recent progress in smart electronic nose technologies enabled with machine learning methods,” 2021. doi: 10.3390/s21227620.
- [2] J. Yan et al., “Electronic nose feature extraction methods: A review,” 2015. doi: 10.3390/s151127804.