Etilotest Digital cu Stocare Cloud

Introducere

Proiectul constă în realizarea unui Etilotest Digital inteligent, capabil să măsoare concentrația de alcool din aerul expirat și să transmită automat aceste date către o platformă Cloud pentru monitorizare.

- **Ce face**: Sistemul detectează vaporii de etanol folosind senzorul MQ-3, procesează semnalul analogic cu ajutorul unui microcontroller ATmega328P și oferă feedback instantaneu (vizual pe LCD și sonor prin buzzer), trimițând simultan datele prin Wi-Fi către un API extern.
- **Care este scopul lui**: Scopul principal este furnizarea unei metode rapide și accesibile de testare a alcoolemiei, asigurând în același timp o evidență digitală a măsurătorilor prin integrarea IoT.
- **Ideea de la care am pornit**: Am pornit de la dorința mea de a-mi proteja prietenii, creând un dispozitiv care să elimine incertitudinea la volan. Am conceput acest etilotest ca un instrument de prevenție responsabil, menit să reducă riscurile rutiere și să ofere o metodă rapidă de verificare a alcoolemiei înainte de plecarea la drum.
- **De ce este util**: Pentru mine, acest proiect este o oportunitate de a învăța cum să conectez hardware-ul analogic la puterea Cloud-ului. Pentru ceilalți, consider că este un instrument vital care poate salva vieți, oferindu-le prietenilor mei și comunității o metodă simplă de a lua decizii corecte și responsabile înainte de a urca la volan.

Descriere generală

Modul Hardware	Descriere Tehnică	Interacțiune / Protocol
Arduino UNO R3 (ATmega328p + ATmega16u2)	Unitatea centrală de procesare. Gestionează logica locală și calculele.	Master-ul sistemului; coordonează toate perifericele și trimite date către gateway.
Senzor MQ-3	Senzor chimic pentru detectarea vaporilor de etanol (alcool).	Analog (ADC): Trimite semnal variabil (0-5V) către pinul A0 al ATmega328P.
RPi Pico 2WH	Gateway Wi-Fi bazat pe arhitectură ARM.	UART: Primește date seriale de la ATmega și le transmite în Cloud prin Wi-Fi.
LCD 1602	Ecran pentru afișarea în timp real a rezultatelor.	I2C: Primește comenzi de afișare pe 2 fire (SDA/SCL) de la ATmega328P.
Buzzer Pasiv	Componentă pentru avertizare sonoră.	PWM: Generare tonuri de frecvențe diferite în funcție de concentrație (Pin D9).

Hardware Design

Descriere componente

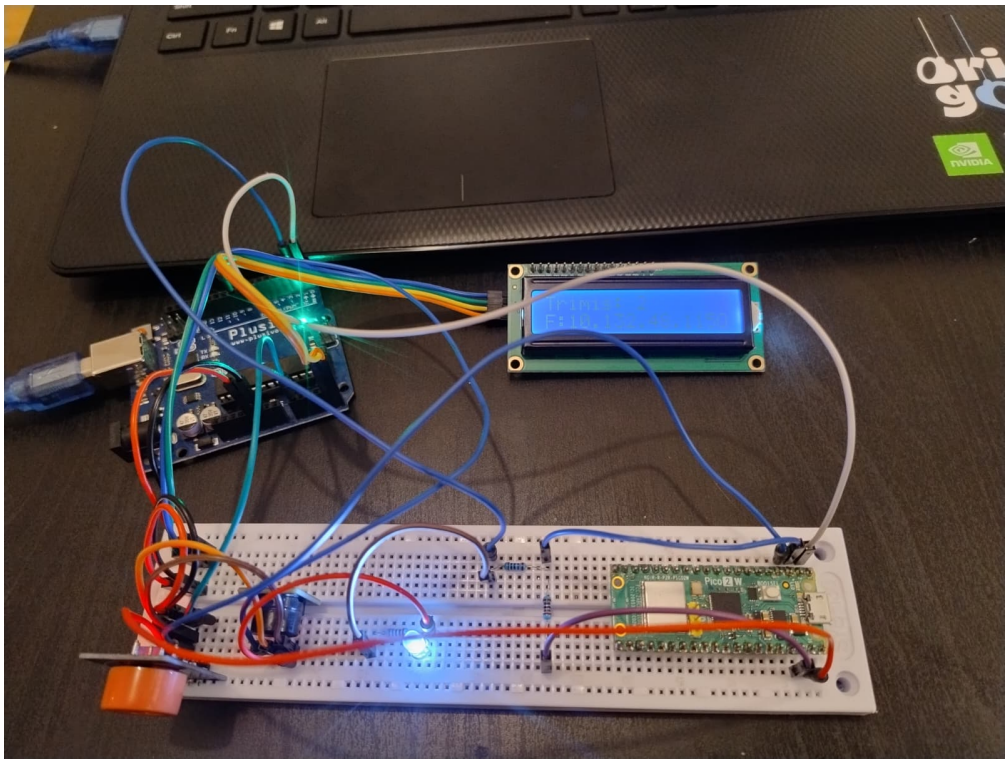
- Arduino UNO R3
- Senzor MQ-3
- RPi Pico 2WH
- LCD 1602
- Buzzer Pasiv
- Set Jumper Wires
- Breadboard
- Set Rezistențe

Tabel Conexiuni Hardware

Componentă	Pin Componentă	Destinație (Placă/Pin)	Detalii
Senzor MQ-3	VCC	Arduino 5V	Necesită preîncălzire pentru precizie
	GND	Arduino GND	
	AO	Arduino A0	Citire concentrație alcool
LCD I2C	VCC	Arduino 5V	
	GND	Arduino GND	
	SDA	Arduino A4	Conexiune Date I2C
	SCL	Arduino A5	Conexiune Ceas I2C
Buzzer Pasiv	VCC	Arduino 5V	
	GND	Arduino GND	
	S	Arduino D9	Generare alerte sonore
LED Debug	Anod (A)	Arduino D13	Serie cu rezistență 220Ω
	Catod (C)	Arduino GND	
Pico 2WH	VSYS	Arduino 5V	Intrare alimentare stabilizată
	GND	Arduino GND	Masa comună
	GP0 (TX)	Arduino D0 (RX)	Transmisie date către Arduino
	GP1 (RX)	Divizor Tensiune	Recepție date protejată (3.3V)
Divizor Tensiune	In (R 1k)	Arduino D1 (TX)	Punct intrare semnal 5V
	Out (1k/2.2k)	Pico GP1 (RX)	Punct ieșire semnal ~3.4V
	GND (R 2.2k)	Pico GND	Masă comună

Schema electrica

Rezultate



Software Design

Mediul de Dezvoltare

Pentru realizarea sistemului etilotest s-au utilizat trei medii de dezvoltare complementare:

- **Arduino IDE:** Utilizat pentru scrierea, compilarea și încărcarea firmware-ului pe plăcuța Arduino Uno R3. A fost ales pentru suportul hardware excelent și rapiditatea în gestionarea modulelor analogice și I2C.
- **Thonny IDE:** Utilizat pentru dezvoltarea și depanarea scripturilor MicroPython rulate pe Raspberry Pi Pico 2WH, oferind un control în timp real asupra execuției codului și a stării conexiunii la rețea.
- **PythonAnywhere Web Editor:** Utilizat pentru editarea directă în cloud a scriptului Flask și vizualizarea fișierelor de log-uri.

Librării și Surse 3rd-Party

Sistemul se bazează pe următoarele biblioteci software externe pentru asigurarea comunicării și abstractizarea hardware-ului:

Librărie / Modul	Platformă	Rol în Proiect
Wire.h	Arduino	Permite comunicarea pe magistrala I2C (protocol nativ).
LiquidCrystal_I2C.h	Arduino	Gestionarea ecranului LCD 16×2 prin modulul adaptor I2C PCF8574.
network	Pico	Modul nativ MicroPython pentru controlul cipului Wi-Fi integrat pe Pico 2WH.
urequests	Pico	Versiunea optimizată de MicroPython a librăriei requests, folosită pentru trimiterea cererilor HTTP POST către server.
json	Pico / Flask	Serializarea datelor sub formă de pachete JSON transmise prin rețea.
flask	Server	Framework web lightweight în Python utilizat pentru ridicarea endpoint-ului HTTP și rutarea paginilor.

Algoritmi și Structuri de Date Implementate

A. Algoritmul de Calibrare Dinamică (Baseline Calibration)

La pornirea sistemului, senzorul MQ-3 necesită o fază de stabilizare. Algoritmul citește valoarea analogică de 20 ori la un interval de 100 ms (timp total de 2 secunde) și calculează o medie aritmetică stocată în variabila **baseline**. Acest „zero matematic” este scăzut din toate citirile ulterioare pentru a elimina variațiile de temperatură și zgomotul de fond din cameră.

B. Algoritmul de Detecție a Vârfului (Peak Detection)

Pentru a asigura o măsurătoare corectă, sistemul nu trimite prima valoare citită, ci așteaptă ca utilizatorul să termine de suflat. Algoritmul monitorizează în timp real valoarea BAC. Cât timp valoarea crește, ea este salvată în **maxBAC**. În momentul în care respirația se oprește, iar valoarea curentă scade cu un prag fixat de safety ($\Delta = 0.05\text{ g/L}$) sub **maxBAC**, algoritmul marchează starea **peakReached = true** și blochează valoarea maximă pentru procesare.

C. Structura de Date pentru Istoric (Buffer Circular Virtual)

Pe server, datele transmise sunt salvate într-un fișier persistent (**istoric.txt**) sub formă de linii CSV **(Timestamp, Valoare)**. La fiecare accesare a paginii **ghindea.pythoneverywhere.com/grafic**, serverul citește fișierul de la coadă la capăt și păstrează doar ultimele 20 de înregistrări într-o listă de dicționare Python (implementare similară unui *Queue* cu dimensiune fixă), asigurând fluiditatea randării graficului din Chart.js.

Fluxul Programului

Sistemul funcționează pe baza unei mașini de stări distribuite între cele două plăcuțe și server. Mai jos este descris flow-ul logic cap-la-coadă:

1. **Inițializare & Calibrare (Arduino + Pico):**

- Arduino își configurează pinii, inițializează ecranul LCD și rulează algoritmul de calibrare (afișând “Calibrare...”).
- În paralel, Pico pornește cipul radio și încearcă asocierea cu router-ul Wi-Fi. Dacă reușește, trimite IP-ul alocat către Arduino prin UART, confirmând starea de “Online”.

2. **Starea de Repaus (Idle):**

- Arduino afișează “Suflati...” pe ecran și ascultă pinul **A0**.

3. **Faza de Testare (Sampling):**

- Utilizatorul suflă în senzor. Arduino procesează semnalul brut prin funcțiile **max()** și **map()**.
- Se rulează algoritmul de Preak Detection.

4. **Procesare Locală (Alerte):**

- Imediat ce vârful este detectat, Arduino blochează valoarea și activează Buzzer-ul selectând melodia în funcție de gravitate ($<0.4\text{ g/L}$: bipuri scurte, $0.4 - 0.8\text{ g/L}$: Mario Death, $>0.8\text{ g/L}$: Apocalypse).

5. **Sincronizare Cloud (UART → HTTP):**

- Arduino trimite valoarea numerică curată către Pico prin linia serială hardware (pinii D1 TX $\rightarrow$ GP1 RX).
- Arduino își schimbă ecranul în “Trimite in Cloud - Asteptare...” și pornește un cronometru de timeout de 10 secunde.
- Pico recepționează numărul, îl împachetează într-un obiect JSON **{“valoare”: “X.XX”}** și face o cerere **HTTP POST** către PythonAnywhere.

6. **Validare și Confirmare (Server → Hardware):**

- Serverul Flask primește JSON-ul, extrage valoarea prin conversie **float**, generează un timestamp curent și o salvează în fișierul **istoric.txt**.
- Serverul răspunde cu **Status 201 Created** și textul brut **“SALVAT”**.
- Pico recepționează răspunsul HTTP de succes și trimite instant prin UART mesajul **“SALVAT\n”** înapoi către Arduino.

7. **Finalizare Ciclu:**

- Arduino primește **“SALVAT”**, oprește cronometrul, confirmă pe LCD: “Cloud: SUCCES! BAC:X.XX salvat.” timp de 5 secunde.

Rezultate

Concluzii

Proiectul a demonstrat realizarea cu succes a unui sistem IoT complet de monitorizare și avertizare a alcoolemiei (Etilotest Cloud), acoperind toate cele trei straturi fundamentale: hardware/embedded local, comunicație inter-plăcuțe și infrastructură server/web.

Prin implementarea acestui sistem, am obținut următoarele rezultate și competențe:

- **Sincronizare Multi-Arhitectură:** S-a realizat o punte de comunicație stabilă între o arhitectură pe 8 biți (ATmega328P - Arduino) și una modernă pe 32 de biți (RP2350 - Raspberry Pi Pico 2WH), utilizând protocolul serial UART hardware.
- **Procesare Robustă de Semnal:** Algoritmul de **Calibrare Dinamică** și cel de **Peak Detection** au asigurat izolarea zgomotului de fond al senzorului MQ-3 și capturarea cu precizie a valorii maxime din respirație, fără a bloca resursele microcontrolerului.
- **Securitate Hardware:** Integrarea unui divizor de tensiune pe linia RX a plăcuței Pico a asigurat adaptarea nivelurilor logice ($5\text{V} \rightarrow 3.3\text{V}$), protejând pinii sensibili ai microcontrolerului.
- **Persistența Datelor în Cloud:** Serverul Flask dezvoltat pe PythonAnywhere oferă o metodă fiabilă de stocare asincronă a istoricului într-un fișier persistent și randarea dinamică a datelor prin Chart.js, facilitând monitorizarea de la distanță.

Dificultăți întâmpinate și soluții

1. **Saturația și Calibrarea Senzorului:** Inițial, senzorul MQ-3 genera valori reziduale mari în aer curat. Problema a fost rezolvată prin scrierea unei rutine de eșantionare în `setup()` care calculează un `baseline` la fiecare pornire.

2. **Erori de Parsare pe Server (Eroarea HTTP 400):** Criptarea URL-encoded realizată automat de anumite tipuri de request-uri din MicroPython bloca split-ul de caractere pe server. Trecerea la un pachet de date formatat explicit ca JSON standardizat (`application/json`) a rezolvat definitiv problemele de compatibilitate dintre Pico și endpoint-ul Flask.

Source

Github repo

Export to PDF