Sistem de Monitorizare si Protectie Industriala

Introducere

Proiectul implementeaza un sistem de monitorizare si protectie pentru o linie de productie industriala simulata. In centrul sistemului se afla un motor stepper 28BYJ-48 care reprezinta echipamentul principal de productie - de exemplu o masa rotativa de asamblare sau un conveyor. In jurul lui sunt dispusi mai multi senzori care detecteaza situatiile periculoase: prezenta personalului in zona de operare, apropierea unor obiecte, infiltratii de apa, zgomote anormale si conditii de mediu nepotrivite.

Punctul de plecare a fost o intrebare concreta: cum se opreste in siguranta un echipament industrial cand apare o anomalie? In aplicatiile reale, timpul de raspuns trebuie sa fie de ordinul milisecundelor, iar sistemul trebuie sa functioneze fiabil chiar si atunci cand o parte din componente cedeaza. Aceste cerinte motiveaza arhitectura aleasa, cu doua microcontrollere avand roluri distincte.

Proiectul acopera conceptele de curs - senzori analogici si digitali, comunicatie seriala, WiFi, interfata cu utilizatorul - iar elementul principal de interes este arhitectural: separarea hard real-time (siguranta, control motor) de procesare si conectivitate pe microcontrollere distincte este o abordare standard in sistemele industriale.

Descriere generala

Sistemul este construit pe doua microcontrollere cu roluri distincte:

- **Arduino MEGA 2560** indeplineste functia de controler hard real-time. Citeste toti senzorii la fiecare 10 ms, ruleaza state machine-ul de siguranta, controleaza direct motorul si buzzer-ul si calculeaza scorul de sanatate. Bucla principala este non-blocking si bounded ca durata.
- **ESP32 WROOM-32D** functioneaza ca punte intre Arduino si infrastructura externa. Singura sa intrare proprie este citirea senzorului DHT11 (conectat fizic la el), retransmisa catre Arduino printr-un cadru UART de retur. In rest, primeste cadrele de la Arduino, le afiseaza pe LCD si le retransmite catre server prin WebSocket. Nu efectueaza procesare locala.

Arhitectura cu Arduino ca unica sursa de adevar aduce doua avantaje: redundanta (la caderea WiFi sau ESP32, Arduino continua sa opreasca motorul corect) si eliminarea variabilitatii de timing introduse de retea. Temperatura ambianta nu se modifica in milisecunde, asadar este suficient ca ESP32 sa transmita valoarea DHT11 catre Arduino la interval de aproximativ 2 secunde.

Fluxul de date implementat:

1. Arduino citeste senzorii rapizi (PIR, sunet, apa, buton) la fiecare 10 ms
2. Senzorul ultrasonic ruleaza asincron: declanseaza un puls la fiecare 60 ms, iar ecoul e cronometrat printr-o intrerupere pin-change (fara blocare cu pulseIn)
3. State machine-ul evalueaza prioritatile cu histereza si seteaza starea
4. La fiecare 50 ms Arduino trimite un cadru UART de 27 bytes catre ESP32 (20 Hz): 2 bytes header + 1 byte LEN + 23 bytes payload + 1 byte XOR checksum
5. ESP32 valideaza checksum-ul, trimite ACK/NACK si imediat retransmite cadrul catre server prin WebSocket
6. La fiecare ~2 secunde ESP32 trimite spre Arduino temperatura si umiditatea citite de la DHT11
7. LCD-ul se reimprospateaza la fiecare 500 ms, cu paginile rotite la 3 secunde
8. Dashboard-ul web primeste actualizari instant prin WebSocket si se redeseneaza la fiecare cadru

Logica de siguranta e organizata ca o cascada cu prioritati. Starile cu prioritate mai mare le suprascriu pe cele inferioare:

```
EMERGENCY      - buton apasat (latched, eliberat doar prin triple-press)
SOUND_SHUTDOWN - 5 secunde de pauza dupa un spike de zgomot
MAINTENANCE    - 10 secunde dupa detectie PIR (cu refresh la fiecare
detectie noua)
PROXIMITY      - obiect mai apropiat de 40 cm (live, cu histereza la 50 cm)
WATER_HIGH    - senzor de apa peste 512/1023 (live, cu histereza la 450)
TEMP_HIGH      - temperatura peste 35 °C (live, cu histereza la 33 °C)
NORMAL         - functionare nominala
```

Motorul functioneaza exclusiv in starea NORMAL. WARNING exista in enum (valoarea 1) dar nu este produsa de nicio conditie - orice anomalie declanseaza o stare de oprire, conform cerintei unui controler de siguranta industriala.

Hardware Design

Lista de piese

#	Componenta	Cant.	Link
1	Arduino MEGA 2560 R3 (clona, ATmega2560 + CH340)	1	Optimus Digital
2	Placa de dezvoltare ESP32 cu WiFi si Bluetooth (ESP-WROOM-32)	1	Optimus Digital
3	Modul senzor PIR HC-SR501	1	Optimus Digital
4	Senzor ultrasonic HC-SR04	1	Optimus Digital
5	Modul senzor de sunet (iesire A0 + D0)	1	Optimus Digital
6	Senzor de nivel al apei (analogic)	1	Optimus Digital
7	Modul senzor DHT11 (temperatura + umiditate)	1	Optimus Digital
8	Set motor pas cu pas 28BYJ-48 + driver ULN2003	1	Optimus Digital
9	LCD 1602 HD44780 cu backlight verde, 5 V (paralel, 16 pini)	1	Optimus Digital
10	Buzzer activ 5 V	1	Optimus Digital
11	Buton tactil 6x6x6 mm (emergency stop)	1	Optimus Digital
12	Mini potentiometru 10k (contrast LCD)	1	Optimus Digital
13	Set rezistoare 0.25 W (220 ohm pt. backlight + 1k si 2k pt. divizorul UART)	1	Optimus Digital

14	Breadboard HQ 830 puncte	2	Optimus Digital
15	Set fire tata-tata 40p / 20 cm	1	Optimus Digital
16	Set fire mama-tata 40p / 10 cm	1	Optimus Digital
17	Alimentator 9 V / 1 A cu mufa DC	1	Optimus Digital

Aproape toate componentele provin din acest kit AliExpress. Singura exceptie este placa ESP32 WROOM-32, achizitionata separat. Link-urile Optimus Digital de mai sus sunt echivalente individuale, pentru referinta.

Organizarea fizica

Componentele sunt distribuite pe doua breadboard-uri pentru a separa responsabilitatile sistemului si pentru a facilita mentenanta.

Breadboard principal (Arduino):

- Arduino MEGA pozitionat central
- Senzorii de intrare in partea stanga (PIR, ultrasonic, sound, water)
- Actuatorii in partea dreapta (driver stepper, buzzer)
- Distributia de alimentare (5V/GND) pe rail-urile superioare

Breadboard secundar (ESP32):

- ESP32 pozitionat central
- LCD1602 conectat prin 6 fire de control
- DHT11 pentru monitorizarea ambientala
- Conexiunile UART catre Arduino prin fire dedicate

Alimentarea pleaca de la adaptorul de 9V care intra in jack-ul DC al Arduino-ului. Regulatorul intern al placii furnizeaza 5V stabil, care alimenteaza toti senzorii si ESP32-ul (prin pinul Vin). ESP32-ul are propriul regulator intern care coboara tensiunea la 3.3V pentru functionarea interna. Toate componentele impart o masa comuna distribuita prin rail-urile negative ale celor doua breadboard-uri.

Schema electrica

Schema electrica (PDF)

Pinout

Arduino MEGA:

Componenta	Pini Arduino
PIR HC-SR501	OUT → D12, VCC → 5V, GND → GND

HC-SR04	Trig → D11, Echo → D10 (PCINT4)
Sound Sensor	A0 → A0, D0 → D9, + → 5V, G → GND
Water Sensor	S → A1, + → 5V, - → GND
Emergency Button	D18 cu INPUT_PULLUP, celalalt capat → GND
Stepper ULN2003	IN1→D8, IN2→D7, IN3→D6, IN4→D5
Buzzer	+ → D4, - → GND
UART catre ESP32	TX2 (D16) → divizor 1k/2k → RX2 ESP; RX2 (D17) ← TX2 ESP direct

ESP32:

Componenta	**Pini ESP32**
DHT11	GPIO32
LCD1602	RS→GPIO5, EN→GPIO18, D4→GPIO19, D5→GPIO21, D6→GPIO22, D7→GPIO23
UART catre Arduino	RX2 (GPIO16), TX2 (GPIO17)

Compatibilitatea electrica

Aspectul electric principal este interfatarea intre Arduino (logica 5V) si ESP32 (logica 3.3V). Conform datasheet-ului, pinii GPIO ai ESP32 sunt 5V tolerant pe input, ceea ce permite conectarea directa a liniei TX a Arduino-ului la RX-ul ESP32-ului fara level shifter. Pentru directia inversa (ESP32 TX 3.3V → Arduino RX 5V), tensiunea de 3.3V este interpretata ca HIGH, intrucat pragul de detectie al Arduino-ului este in jur de 3V.

Divizorul 1k/2k pe linia Arduino TX → ESP32 RX functioneaza fara erori la 115200 baud. La 250 000 baud parser-ul ESP32 pierde sincronizarea frecvent - cauza probabila este marginea redusa de integritate a semnalului prin divizor si fire de breadboard. La 115200 link-ul este stabil iar bandwidth-ul ramane peste necesar (cadrul de 27 bytes la 20 Hz ocupa sub 5% din capacitate).

LCD-ul este alimentat la 5V dar accepta semnale de control de 3.3V, intrucat pragul sau de HIGH este in jurul valorii de 2.5V.

Estimare consum energetic

Valorile de mai jos sunt estimate pe baza datasheet-urilor componentelor.

Componenta	**Curent (estimat)**	**Putere**
Arduino MEGA active	200 mA	1000 mW
ESP32 cu WiFi activ	240 mA	792 mW
PIR	50 mA	250 mW
HC-SR04	15 mA	75 mW
Sound sensor	4 mA	20 mW
Water sensor	5 mA	25 mW
DHT11	0.3 mA	1.5 mW
LCD1602	20 mA	100 mW
Stepper activ	300 mA	1500 mW

Buzzer activ	30 mA	150 mW

Consumul total maxim estimat este de aproximativ 840 mA (3.9 W), iar consumul tipic - cu stepper-ul in functiune intermitenta si buzzer-ul activat ocazional - se situeaza in jur de 550 mA (2.8 W). In practica ESP32-ul ruleaza cu `WiFi.setSleep(false)` (pentru latenta retelei), consuma constant ~240 mA, iar valoarea reala este apropiata de plafon.

Pentru o eventuala alimentare cu acumulatori NiMH de 1.2V / 2500 mAh, sunt necesari 8 acumulatori in serie pentru a obtine 9.6V (compatibil cu intrarea Vin a Arduino-ului). Autonomia estimata este de aproximativ 4 ore.

Software Design

Mediu de dezvoltare

- **Arduino IDE** pentru ambele microcontrollere, folosind board package-ul ESP32 oficial Espressif
- **Python 3 cu Flask + flask-sock** pentru server-ul web
- **SQLite** in modul WAL pentru persistenta datelor
- **HTML, CSS si JavaScript vanilla** pentru dashboard, cu Chart.js incarcat din CDN

Biblioteci si surse 3rd-party

Pe Arduino:

- `avr/wdt.h` si `avr/interrupt.h` pentru watchdog si ISR (built-in)
- Driver propriu pentru motorul stepper (full-step, two-phases-on, cu rampa de acceleratie)
- Parser propriu pentru cadrele UART (state machine, fara alocare dinamica)

Pe ESP32:

- `WiFi.h` - conectarea la retea
- `WebSocketsClient` (Markus Sattler / Links2004) - clientul WebSocket spre server
- `LiquidCrystal.h` - controlul LCD-ului (interfata paralela)
- `DHT.h` (Adafruit) - citirea senzorului DHT11
- `ArduinoJson.h` (Benoit Blanchon) - serializarea cadrelor in JSON

Pe server:

- `Flask` - framework web minimalist
- `flask-sock` + `simple-websocket` - endpoint-urile WebSocket
- `sqlite3` (built-in) - persistenta datelor
- `Chart.js` (CDN) - graficele de pe dashboard

Algoritmi si structuri

State machine pentru siguranta (Arduino)

Sistemul functioneaza pe baza unei masini cu 7 stari active si o stare rezervata. Prioritatile, de la cea mai mare la cea mai mica:

```
EMERGENCY      - buton apasat (latched, eliberat prin triple-press)
SOUND_SHUTDOWN - 5 secunde dupa un spike de zgomot
MAINTENANCE    - 10 secunde dupa detectie PIR
PROXIMITY      - obiect < 40 cm (live, histereza la 50 cm)
WATER_HIGH    - apa > 512 (live, histereza la 450)
TEMP_HIGH      - temperatura > 35 °C (live, histereza la 33 °C)
WARNING        - rezervata (anomalie minora care nu opreste motorul)
NORMAL         - functionare nominala
```

La fiecare iteratie a buclei principale (10 ms), conditiile sunt evaluate de la prioritatea cea mai mare catre cea mai mica. Prima conditie indeplinita determina starea curenta. Motorul functioneaza exclusiv in NORMAL.

Histerezele sunt necesare pentru starile “live” (PROXIMITY, WATER_HIGH, TEMP_HIGH); in absenta lor, sistemul ar oscila intre stop si start cand un senzor se afla in jurul pragului.

E-stop cu latching si reset prin triple-press

Prima apasare latcheaza EMERGENCY indefinit. Iesirea necesita inca 3 apasari in maxim 2 secunde - secventa impune o actiune deliberata si previne resetarile accidentale. Debouncing 50 ms in software. La reset se sterg si timerele PIR / sunet, asadar linia reporneste imediat.

Driver stepper non-blocking cu rampa de acceleratie

28BYJ-48 are frecventa maxima de pull-in in jur de 600 Hz - sub viteza tinta pentru demo (~20 RPM). Solutia adoptata este o rampa liniara: pornire la 4000 µs/pas (~7 RPM, sub pull-in) si reducerea delay-ului cu 2 µs/pas pana la 1500 µs/pas (~20 RPM). Rampa se completeaza in ~1250 pasi (~3.4 secunde). Este resetata la fiecare tranzitie OFF → ON, astfel incat motorul reporneste lin dupa fiecare iesire din MAINT / PROXIMITY (incercarea de a porni direct la viteza de croaziera produce vibratie fara rotatie).

Secventa de comutare este full-step “two-phases-on”. Pe placa fizica IN3 si IN4 sunt inversate fata de ordinea alfabetica - mapping-ul real este *A=IN1, B=IN2, C=IN4, D=IN3*, verificat cu un sketch de diagnostic care energizeaza fiecare coil pe rand. Secventa din software reflecta acest mapping; fara corectie, motorul vibreaza fara sa se roteasca.

Senzor ultrasonic non-blocking

`pulseIn` este blocking pana la ~25 ms - jitter inacceptabil pentru bucla. In locul sau, ecoul este masurat printr-o intrerupere pin-change pe D10 (PCINT4): ISR citeste micros() la fiecare tranzitie si calculeaza latimea pulsului. Bucla declanseaza trigger-ul la 60 ms si citeste rezultatul cand este disponibil. Jitter rezultat ~3-5 µs, sub precizia proprie a HC-SR04.

Protocolul UART

Doua tipuri de cadre, ambele cu acelasi pattern (header + LEN + payload + XOR):

```
Forward  (Arduino -> ESP32): 0xAA 0x55 LEN(23) payload XOR    la fiecare 50
ms
Reverse  (ESP32 -> Arduino): 0xBB 0x66 LEN(4)  payload XOR    la fiecare ~2
s
ACK / NACK (single byte): 0x06 / 0x15, ESP32 -> Arduino       per cadru
forward
```

Payload-ul forward contine starea, toti senzorii, uptime, temp/umiditate (retransmise de la ESP32), scorul de sanatate, unghiul motorului si timpul ramas pana la repornire. Reverse contine temp_x10 si hum_x10 - DHT11 nu se actualizeaza mai rapid de 1 Hz. Checksum-ul XOR detecteaza coruptiile cauzate de EMI-ul stepper-ului. NACK-urile sunt contorizate dar **nu** declanseaza retransmisie - retransmisia ar introduce variabilitate de timing, iar urmatorul cadru ajunge in 50 ms.

Detectia status-ului link-ului

Arduino monitorizeaza contorul ACK. La absenta incrementarii timp de 1.5 s, declara link pierdut si afiseaza »> `LINK LOST/UP` pe USB Serial la tranzitii. ESP32 aplica simetric aceeasi logica, cu timeout 3 s (mai permisiv intrucat propria bucla se poate bloca temporar in operatii WiFi/WS).

Calculul scorului de sanatate (Arduino)

Scorul (0-100) este calculat cu aritmetica intreaga la fiecare 50 ms, chiar inainte de trimiterea cadrului. Penalizari:

```
temperatura in afara 18-28 °C  : 3 puncte / grad
umiditate in afara 30-70 %     : 1 punct / procent
apa peste 500                  : -30 puncte
distanta sub 40 cm             : -10 puncte
stare WARNING                  : -20
stare MAINTENANCE              : -10
stare SOUND_SHUTDOWN           : -25
stare EMERGENCY                : -100 (forteaza 0)
link cu ESP32 pierdut          : -40
DHT nu a raportat inca         : -5
```

Watchdog timer

Pe Arduino este activat watchdog-ul cu timeout de 4 secunde. La blocarea buclei principale din orice motiv (deadlock, ciclu infinit), microcontroller-ul se reseteaza automat. Apelul `wdt_reset()` este executat la inceputul fiecarei iteratii a buclei.

Optimizari pentru determinism si latenta

Decizii aplicate in fiecare strat pentru o bucla Arduino predictibila si latenta end-to-end redusa:

- **Arduino**: ultrasonic prin PCINT (fara `pulseIn` blocking), link UART open-loop fara retransmisii la NACK, port I/O direct (PORTx/PINx) pe pinii frecvent accesati, prescaler ADC /32 (~27 µs/citire),

watchdog 4 s.
- **ESP32**: `WiFi.setSleep(false)` pentru a evita ~100 ms latenta per pachet introdusa de modem sleep, buffer UART RX 2048 bytes pentru ca blocajele temporare ale buclei sa nu desincronizeze parser-ul, WebSocket persistent catre server cu push la fiecare cadru primit (20 Hz).
- **Server**: SQLite WAL cu `synchronous=NORMAL` (commit-uri sub-ms fata de ~10 ms cu fsync), broadcast WS thread-safe peste un set de clienti.
- **Frontend**: WebSocket pentru update-uri live, bootstrap istoric prin REST la incarcare urmat de append incremental, Chart.js cu `animation: false`.

Functii principale

Pe Arduino:

- `readFastSensors()` - citeste senzorii rapizi prin acces direct la PINx
- `ultrasonicTick()` - state machine pentru declansarea pulsului si culegerea rezultatului
- `buttonTick()` - detectia edge a butonului E-stop, latching, triple-press reset
- `evaluateSafetyState()` - aplica logica state machine cu histereza
- `stepperTick()` - driver non-blocking cu rampa de acceleratie
- `buzzerTick()` - patternuri sonore diferite per stare
- `sendFrame()` - impacheteaza cadrul forward si il scrie in TX-ul UART
- `pollEspStream()` - parser pentru bytes-urile primite de la ESP32 (ACK/NACK si cadre reverse)
- `computeHealth()` - calculul scorului de sanatate (aritmetica intreaga)
- `updateLinkStatus()` - monitorizarea contorului de ACK-uri, mesaje »> pe Serial

Pe ESP32:

- `pollUart()` - parser pentru cadrele forward de la Arduino, ACK/NACK
- `pushReadingOverWs()` - serializare JSON + sendTXT pe WebSocket
- `sendReverseFrame()` - trimite temperatura/umiditate spre Arduino
- `renderLcd()` - actualizare LCD cu pagini rotite
- `ensureWifi()` - reconectare automata WiFi
- `wsEvent()` - callback pentru evenimente WebSocket (conectat / deconectat)

Pe server (Flask):

- `/ws/ingest` (WS) - primeste datele de la ESP32, le salveaza in SQLite si le retransmite tuturor clientilor de dashboard
- `/ws/stream` (WS) - clientii dashboard se aboneaza aici, primesc fiecare mesaj ingest in timp real
- `/api/data` (POST) - acelasi ingest ca WS-ul, expus si pe HTTP pentru testare cu `curl`
- `/api/latest`, `/api/history`, `/api/stats` (GET) - pentru bootstrap-ul dashboard-ului
- `/` (GET) - serveste dashboard-ul HTML

Interfata utilizator (dashboard)

Dashboard-ul are o ierarhie vizuala definita, de sus in jos:

1. **Banner de stare** - elementul cel mai proeminent al paginii, cu numele starii afisat mare, o descriere scurta si, cand este cazul, un countdown amplu pentru timpul ramas pana la repornire.

Culorile se schimba per stare, iar EMERGENCY pulseaza.
2. **Cardurile “feature”** - scorul de sanatate cu un gauge SVG si motorul cu un cadran care indica unghiul curent in timp real.
3. **Senzori grupati** in 3 sectiuni: *Ambient* (temp, umiditate), *Proximity & Personnel* (distanta, PIR, E-stop), *Anomaly detectors* (sunet, apa). PIR si E-stop devin rosii cand sunt active.
4. **Grafice istorice** Chart.js pentru temperatura/umiditate, distanta/apa, sunet si scor de sanatate.

Countdown-ul motorului foloseste interpolare locala: la fiecare cadru primit, dashboard-ul retine `hold_ms` si timestamp-ul local, apoi decrementeaza local la 100 ms intre cadre. Fiecare cadru nou resincronizeaza valoarea, rezultand o afisare fluida in loc de salturi in trepte de 50 ms.

Rezultate Obtinute

Sistemul functioneaza end-to-end conform specificatiilor.

Latenta end-to-end (schimbare de senzor → pixel pe dashboard): ~50-75 ms tipic, ~25 ms in cazul cel mai favorabil. Componentele:

Etapa	Latenta
Citire senzor + cadru Arduino	pana la 50 ms (cadenta UART)
Transmisie UART (27 bytes @ 115200)	~2.3 ms
Parsare + push WebSocket ESP32	~1 ms
WiFi → server	~2-5 ms
Insert SQLite WAL + broadcast	~1 ms
Server → browser	~2-5 ms
Randare browser	~16 ms (un frame la 60 fps)

Motorul functioneaza stabil la ~20 RPM dupa rampa de acceleratie, fara vibratii. Pornirile dupa MAINT / PROXIMITY sunt line, iar opririle pentru siguranta sunt instantanee.

State machine-ul raspunde corect la toate scenariile: trecere de mana = MAINT cu countdown, obiect sub 40 cm = PROXIMITY (live), bataie din palme = SOUND_SHUTDOWN, apa peste prag = WATER_HIGH, buton = EMERGENCY latched, triple-press = reset.

Link-ul UART este stabil la 115200 baud prin divizor - zero NACK-uri si zero “LINK LOST” in functionare normala. 250000 baud nu se transmite corect prin divizor combinat cu fire de breadboard.

Dashboard-ul primeste 20 Hz fara lag - 4 grafice, KPI-uri, gauge sanatate si cadran motor redesenate la fiecare cadru.

Probleme intalnite:

- **Coil mapping pe stepper**: pe placa fizica IN3 si IN4 sunt inversate fata de ordinea alfabetica. Verificat cu un sketch de diagnostic (`firmware/stepper_test`) care energizeaza fiecare coil pe rand; secventa din firmware reflecta mapping-ul real.
- **Frecventa de pull-in**: 28BYJ-48 nu poate porni direct la viteza de croaziera, rampa de acceleratie este obligatorie.
- **WiFi modem sleep**: activ implicit, introduce ~100 ms per pachet. `WiFi.setSleep(false)` produce o imbunatatire substantiala a latentei.

- **Buffer UART RX ESP32**: cei 256 bytes impliciti sunt insuficienti cand bucla se blocheaza ocazional (reconectare WS); un buffer de 2048 ofera headroom de ~4 secunde la cadenta de 20 Hz.
- **Heartbeat de debug**: zeci de apeluri `Serial.print` separate blocheaza bucla timp de 19 ms (rezultand desincronizarea motorului); un singur `snprintf` compact in TX-ul UART are impact neglijabil.

Concluzii

Cateva decizii arhitecturale au avut impact disproportionat de mare:

Logica de siguranta pe Arduino, nu pe ESP32. Calculul scorului de sanatate si state machine-ul ruleaza pe Arduino, ESP32 functionand ca punte. Avantajele: o singura sursa de adevar, eliminarea variabilitatii de timing introduse de retea si rezistenta la caderea WiFi (Arduino opreste motorul corect independent de conectivitate). Orice modul WiFi care implementeaza protocolul UART de 27 bytes poate inlocui ESP32-ul.

WebSocket cu push la 20 Hz. Latenta end-to-end este in jur de 75 ms, iar dashboard-ul are comportament live in loc de refresh periodic. Push-ul declansat de fiecare cadru parsat elimina necesitatea unui timer separat pentru transmisia catre server.

Determinismul are cost. Cost-per-iteratie bounded necesita numeroase alegeri individuale (PCINT, port I/O direct, absenta retransmisiilor, ADC prescaler, heartbeat compact). Cumulate, rezultatul este o bucla cu cost predictibil in zona de microsecunde.

Histereza este obligatorie. Toate starile “live” (PROXIMITY, WATER_HIGH, TEMP_HIGH) necesita praguri separate enter / exit. In caz contrar, un senzor care variaza in jurul pragului determina sistemul sa intre si sa iasa rapid din stare - comportament inutilizabil.

Triple-press pentru reset. Impune o actiune deliberata si previne resetarile accidentale ale E-stop-ului. Un singur buton indeplineste atat functia de latch cat si pe cea de reset - secventa de 3 apasari in 2 secunde are probabilitate redusa de declansare accidentala.

Imbunatatiri viitoare neimplementate din lipsa de timp sau de componente:

- Senzor de vibratii (MPU6050) pentru detectia uzurii rulmentilor
- Profil de acceleratie S-curve pentru o pornire mai lina
- Tranzitii de stare salvate ca eveniment separat, nu doar metricile
- Autentificare pe dashboard si HTTPS/WSS pentru un deployment in productie
- Trecere la un motor mai puternic - 28BYJ-48 la 20 RPM are cuplu limitat pentru aplicatii reale

Download

https://gitlab.cs.pub.ro/andrei.bleortu/pm

Jurnal

Etapele s-au aliniat pe milestone-urile cursului:

- **Saptamana 11 (4-8 Mai) - Milestone documentatie**: alegerea componentelor, motivarea arhitecturii dual-MCU, schema electrica in KiCad, calculul de consum si redactarea acestei pagini in forma initiala.
- **Saptamana 12 (11-15 Mai) - Milestone hardware**: montaj fizic pe cele doua breadboard-uri, cablare divizor 1k/2k pentru linia Arduino TX → ESP32 RX, testare individuala a fiecarui senzor. In aceasta etapa a fost identificata prima problema: motorul stepper vibra fara sa se roteasca. A fost dezvoltat un sketch dedicat (`firmware/stepper_test`) care energizeaza fiecare coil pe rand, iar testul a indicat ca pe placa concreta IN3 si IN4 sunt inversate fata de ordinea alfabetica.
- **Saptamana 13 (18-22 Mai) - Milestone software**: prima versiune end-to-end functionala (Arduino + ESP32 cu HTTP POST + Flask + dashboard cu polling), urmata in aceeasi saptamana de refactorul principal: inversarea arhitecturii (logica real-time pe Arduino, ESP32 doar ca punte), migrarea la WebSocket persistent in ambele directii, rampa de acceleratie pentru motor, starile noi (PROXIMITY, TEMP_HIGH, WATER_HIGH) cu histereza, E-stop cu latching si triple-press reset, optimizari pentru determinism (PCINT, port I/O direct, ADC prescaler, WiFi.setSleep(false), SQLite WAL).
- **Saptamana 14 (25-29 Mai) - PM Fair**: prezentarea proiectului.

Bibliografie/Resurse

Resurse Hardware:

- Datasheet ATmega2560 - https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf
- Datasheet ESP32 WROOM-32D - https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf
- Datasheet HC-SR04 - https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
- Datasheet DHT11 - https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
- Documentatie 28BYJ-48 cu ULN2003 - https://components101.com/motors/28byj-48-stepper-motor

Resurse Software:

- Documentatia Arduino - https://docs.arduino.cc/
- ESP32 Arduino Core - https://github.com/espressif/arduino-esp32
- WebSockets pentru Arduino/ESP32 (Markus Sattler) - https://github.com/Links2004/arduinoWebSockets
- Documentatie Flask - https://flask.palletsprojects.com/
- flask-sock - https://github.com/miguelgrinberg/flask-sock
- Chart.js - https://www.chartjs.org/docs/latest/
- ArduinoJson - https://arduinojson.org/

Export to PDF