Sesiunea de comunicări științifice studențești, ediția 2024
Robotică, electronică aplicată și automatizarea proceselor
Universitatea Politehnica din București

Motion tracking using wearable sensors

Gul Farhad Ali Irinel
farhadgul2001@gmail.com
Coordonator: Conf. Dr. Ing. Dan Ștefan Tudose
dan.tudose@upb.ro

Abstract –
Problema: Construirea unui set de date cu modele de mers poate fi problematică în momentul în care se încearcă reproducerea mișcărilor persoanelor din videoclipuri de calitate inferioară.
Soluția: Captarea modelelor de mers bazată pe folosirea unor senzori plasați strategic pe corpul fiecărei persoane în timpul colectării datelor. Acești senzori sunt proiectați și programați pentru a înregistra în mod precis mișcările fiecărui participant. În această lucrare voi prezenta detaliile tehnice ale sistemului de captare a datelor.

Cuvinte cheie – Setup, Client, Server, Peer, Senzor, Download Mode, Collect Mode, Hardware, Software

1. Introducere

În experimentele anterioare, pentru a construi un set de date destinat detecției bazate pe modele de mers, au fost folosite doar camere. În acest caz, precizia detectării este semnificativ influențată de performanța modelului de estimare a poziției [1], iar performanța acestui model este afectată de calitatea variabilă a videoclipurilor.

Astfel, s-a dorit introducerea unei alte metode de detecție, care să ajute la estimarea unei poziții mai corecte.

De aceea, pe lângă înregistrările obținute de la camere, am decis să integrăm și senzorii. Cu informațiile obținute de la aceștia, putem construi un schelet cât mai aproape de realitate.

În cadrul proiectului, voi descrie arhitectura sistemului, modul în care acesta poate fi utilizat urmată de o analiză a fiecărei componente în parte.

2. Descrierea arhitecturii

Infrastructura [2] este format dintr-o placă de dezvoltare utilizată pe post de server, pentru a expune un API [3] care să poată fi utilizat prin cereri HTTP și 11 plăci de dezvoltare identice folosite pe post de senzori (peers), care comunică individual cu serverul prin protocolul ESP-NOW [4] cunoscut pentru consumul redus de energie, eliminând astfel necesitatea fiecărui senzor de a expune propriul server web, așa cum am proiectat inițial setup-ul [5].

Pentru a minimiza numărul de încărcări într-o zi și pentru a extinde durata de viață a senzorilor, aceștia sunt proiectați pentru consum redus de energie.

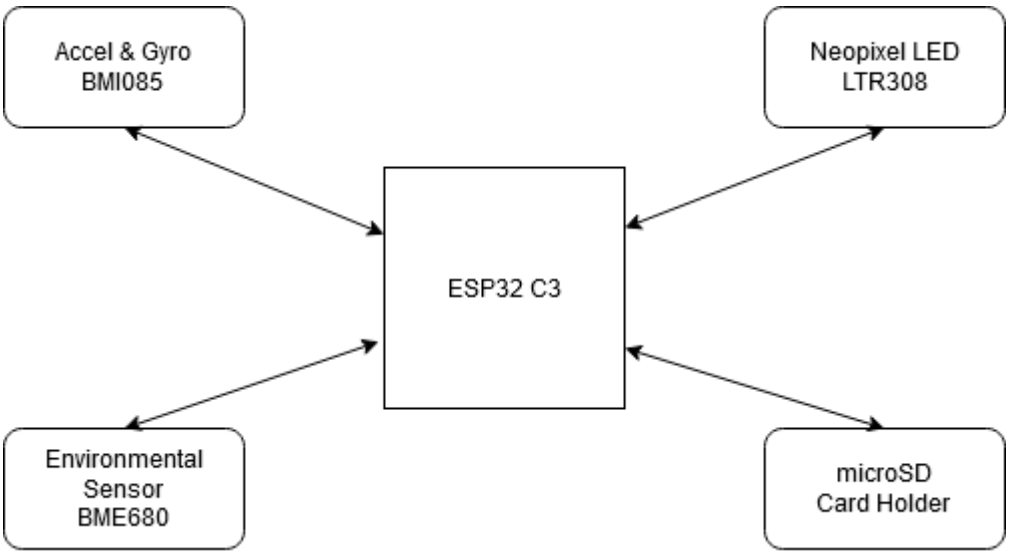

Fig 1. Diagramă cu componentele plăcii

Această placă de dezvoltare dispune de un cititor de carduri microSD pe care se salvează datele, de un accelerometru & giroscop BMI085 [6] care înregistrează datele, de un LED care indică statusul programului și de un senzor de temperatură, presiune și umiditate. Pentru acest proiect, doar primele 3 componente vor fi utilizate.

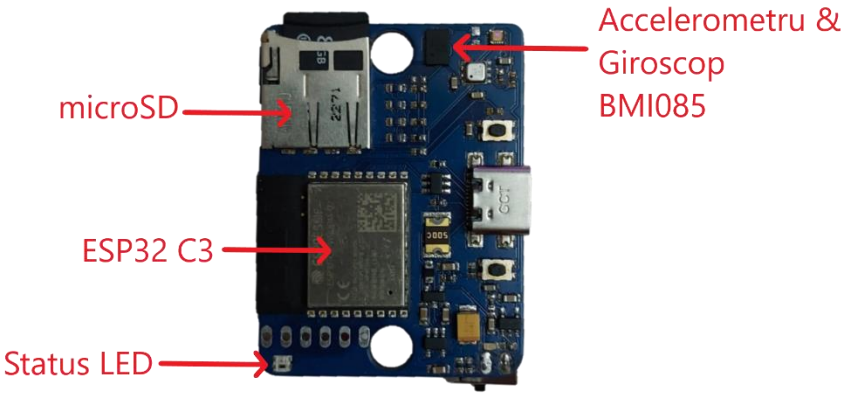

Fig. 2 Placa de dezvoltare

Informațiile pe care dorim să le extragem sunt cele de la accelerometru & giroscop. Datele referitoare la accelerație și rotație pe cele trei axe de coordontate furnizează o modalitate de a reconstitui modelul de mers.

Informațiile extrase au următorul format:

Timestamp	Index	AccelX	AccelY	AccelZ	gyroX	GyroY	GyroZ
1713772743	0	1.043909	0.507589	9.720806	0.000000	0.036220	-0.057526

Timestamp-ul (secundă) are rol de sincronizare, pentru a știi că toți senzorii pornesc simultan. Index reprezintă numărului cadrului din secunda curentă. Restul valorilor sunt cele de la accelerometru & giroscop.

Serverul acționează ca intermediar între client și senzori, facilitând comunicarea între aceste două componente.

Clientul trimite HTTP Request pentru diverse acțiuni precum începerea și oprirea colectării, aflarea nivelului de baterie pentru fiecare senzor, sincronizarea timestamp-ului descris mai sus pentru ca toți senzorii să aibă aceeași referință.

Senzorii primesc mesaje sub formă de pachete de la server prin protocolul ESP-NOW, le interpretează și le execută. Un mesaj conține comanda care trebuie executată și parametrii dacă este cazul.

Ulterior, senzorii transmit la server răspunsurile tot prin ESP-NOW iar serverul răspunde request-ului inițiat de client.

Pentru a putea permite descărcarea datelor de la senzori, aceștia conțin un mod de configurare pe care îl poate iniția clientul, pentru ca ei să expună un server web strict pentru descărcarea fișierelor de pe cardurile SD, fără a folosi serverul ca intermediar.

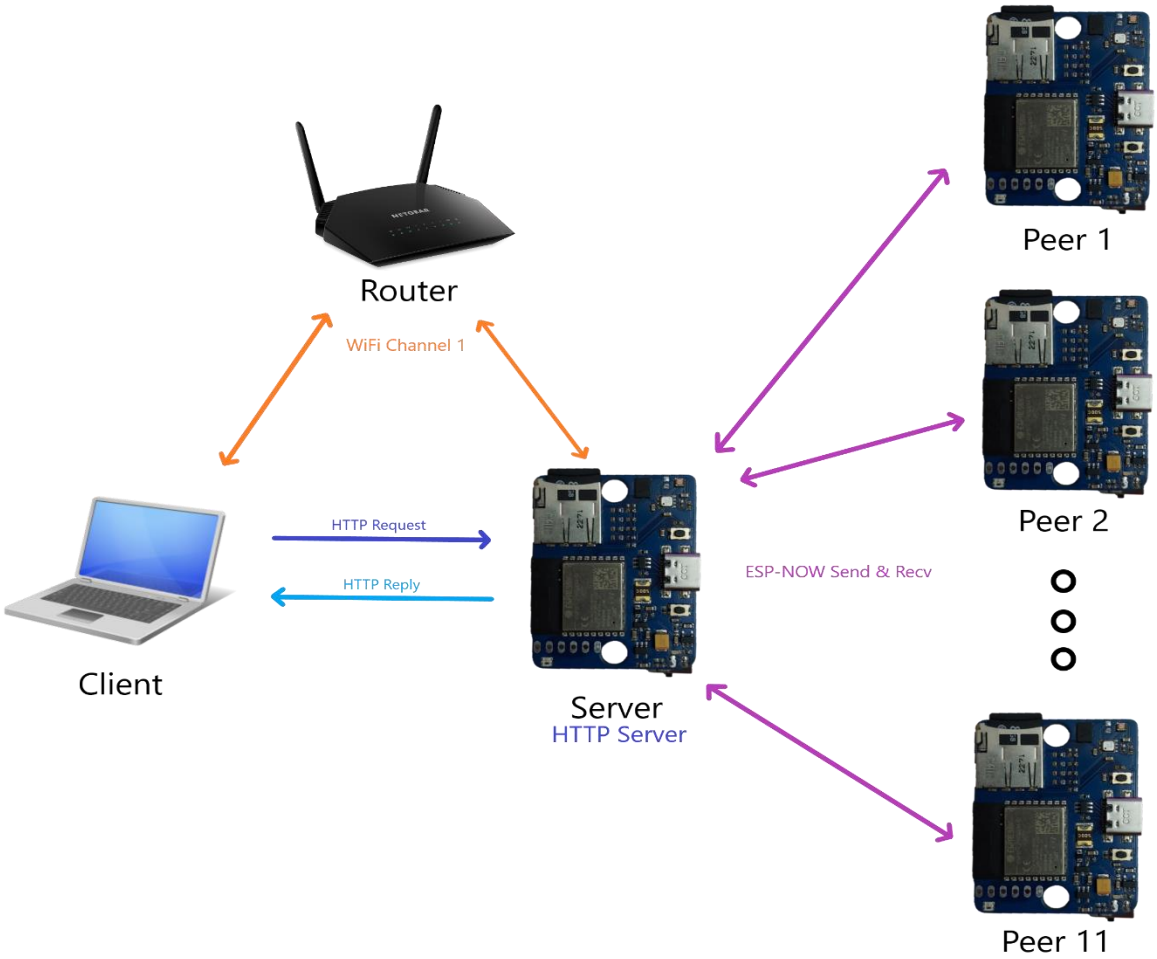

Fig. 3 Infrastructura (mod standard)

3. Utilizare API

3.1. Utilizare API Server

Serverul expune printr-un server web asincron o interfață pentru cereri HTTP. Există și un server web expus de senzori care este prezent pe fiecare senzor aflat în modul de descărcare. Acest API funcționează atunci când rularea este în mod de colectare (*COLLECT MODE*) precum în figura 3 de mai sus.
Cererile expuse de server permanent sunt:

Nr. Comandă	Request	Descriere
1	<SERVER_IP>/isalive	Verificare dacă serverul este activ
2	<SERVER_IP>/sync	Pentru a sincroniza ceasul fiecărui senzor cu ceasul curent al serverului
3	<SERVER_IP>/all_timestamp	Pentru a verifica timestamp-ul de pe fiecare senzor
4	<SERVER_IP>/all_bat	Pentru a verifica nivelul de baterie al fiecărui senzor
5	<SERVER_IP>/file_exists?file=<file_name>	Pentru a verifica daca fișierul cu numele <file_name> există pe fiecare card SD al fiecărui senzor
6	<SERVER_IP>/init?file=<file_name>	Pentru a genera fișierul de colectare pentru runda curentă cu numele <file_name> (**Observație**: Doar se creează fișierul și se pornește timerul, preluarea datelor de la accelerometru, giroscop și scrierea acestora nu sunt pornite încă)
7	<SERVER_IP>/start	Pentru a începe preluarea datelor de la accelerometru, giroscop și scrierea lor pe cardul SD
8	<SERVER_IP>/stop	Pentru a opri runda de colectare
9	<SERVER_IP>/enable_download	Pentru a muta toți senzorii în modul DOWNLOAD_MODE (**Observație**: Această cerere va pune toți senzorii în DOWNLOAD_MODE, ceea ce înseamnă că aceștia nu mai pot comunica cu serverul prin ESP-NOW, pentru mai multe informații, există secțiunea *Moduri de funcționare senzori* din *Comunicarea dintre client, server și senzori)*

Tabel 1 – HTTP Request pentru server de la client

După cum se poate observa din cereri, am decis să separ *init* de *start* deși ambele au ca scop începerea colectării. Motivul separării este cel de timp, fiecare senzor având un timp

aleator între momentul deschiderii fișierului de scriere pe cardul SD + pornirea timerului și momentul în care se începe citirea de pe accelerometru & giroscop.

Soluția separării oferă o întârziere necesară explicată la *Înlănțuirea comenzilor* pentru a asigura pornirea directă a preluării datelor de la accelerometru & giroscop prin comanda *start*, fără să mai fie necesară deschiderea fișierului și pornirea timerului pentru că acestea au fost deja realizate la *init*.

3.2. Utilizare API Senzor în DOWNLOAD MODE

Cererile realizate de către client trebuie făcute pentru fiecare senzor în parte. Acest API funcționează atunci când rularea este în mod de descărcare (*DOWNLOAD MODE*) pentru toți senzorii precum în figura 4.
Cererile expuse de senzori în *DOWNLOAD MODE* sunt:

Nr. Comandă	Request	Descriere
1	<PEER_IP>/download?file=<file_name>	Pentru a porni descarcarea unui fișier cu numele <file_name> de la senzorul cu ip-ul <PEER_IP>
2	<PEER_IP>/disable_download	Pentru a muta senzorul în *COLLECT MODE* (se recomandă verificarea statusului fiecărui senzor la trecerea înapoi în *COLLECT MODE* prin verificarea comunicării cu serverul, de testat nivelul bateriei și resincronizat timestamp-ul)

Tabel 2 – HTTP Request pentru senzori în modul de descărcare

4. Descrierea funcționării și sincronizarea arhitecturii

Pentru realizarea setup-ului, am folosit extensia PlatformIO [7] disponibilă pentru Visual Studio Code. Am împărțit setup-ul în 2 proiecte:

1) esp32_motion_tracking_server – proiectul corespunzător serverului, conține:
    a. main.cpp – programul principal
    b. server.h – variabile reținute precum adrese MAC etc.
    c. secrets.h – date private precum SSID și parola WiFi
2) esp32_motion_tracking_peer – proiectul corespunzător senzorilor, conține:
    a. main.cpp – programul principal
    b. peer.h – variabile reținute pentru programul de colectare sau descărcare
    c. secrets.h – date private precum SSID și parola WiFi

4.1. Implementarea serverului

La inițializare, serverul se conectează la WiFi în modul *WIFI_AP_STA* [8]. Se preia timestamp-ul Unix utilizând un server NTP. Se inițializează protocolul ESP-NOW și se adaugă toți senzorii din lista de adrese MAC. După acești pași, inițializarea este încheiată și este pornit un server web utilizând AsyncWebServer [9] cu request-urile disponibile prezentate la *Utilizare API Server*.

Printre variabilele reținute de către server se enumeră și bufferele unde sunt reținute informațiile provenite de la senzori precum nivelul de baterie al fiecărui senzor (*battery_level_str*), timestampul curent al fiecărui senzor (*timestamp_str*) sau existență fișier pe card SD (*file_exists_str*).

4.2. Implementarea senzorului – tip peer

Inițializarea senzorului depinde de modul de pornire (*COLLECT MODE* / *DOWNOAD MODE*) stabilit anterior. La prima pornire, modul prestabilit este cel de colectare.

În modul *COLLECT MODE* se inițializează:
- cardul SD
- accelerometru & giroscop
- timer-ul (folosit pentru incrementarea frame-ului curent din secundă la colectare)
- alocările de memorie (de exemplu payload-ul reținut în memoria flash înainte de a fi scris pe SD)
- protocolul ESP-NOW împreună cu adăugarea adresei MAC a serverului ca peer.

În modul *DOWNLOAD MODE* se inițializează:
- cardul SD
- conectare la WiFi
- server web prin AsyncWebServer cu request-urile disponibile prezentate la *Utilizare API Senzor în DOWNLOAD MODE*.

Pentru a reține modul în care trebuie să pornească senzorul, am folosit variabila de tip enum state *program_state* care poate să ia valori între *COLLECT_STATE (0)* și *DOWNLOAD_STATE (1)* declarată cu atributul **RTC_NOINIT_ATTR** care permite salvarea între reporniri.

Singura situație de abordat este momentul primei inițializării pentru că de fapt acest atribut blochează reinițializarea variabilei la pornire [10]. Pentru a repara această problemă, am forțat o inițializare în COLLECT_STATE la oricare pornire care nu are ca și cauză de wake-up ESP_RST_SW [11].

4.3. Proiectarea comunicării prin ESP-NOW

Pentru a realiza o comunicare între server și senzori este important ca toate părțile să folosească aceeași structură a pachetelor transmise.

Structura aleasă este următoarea:

```
// For ESP-NOW
typedef struct struct_message {
    uint8_t id;      // id for each esp32
    uint8_t command; // command to be executed
    char text[112];  // text to be send

} struct_message;
```

Am ales să limitez mărimea structurii la 128 bytes.

Folosind **câmpul id**, se pot trimite mesaje către un anumit peer, ajutând și la depanarea problemelor pentru că fiecare senzor trimite mesaj cu id-ul propriu (pentru a diferenția id-ul propriu am utilizat adresa MAC ca element unic, fiind diferit între senzori) și dacă apar probleme, se știu exact de la care senzor provin.

Câmpul command conține indicatorul comenzii care trebuie executate. Întrucât și serverul trimite date la senzori, dar și senzorii la server (comunicare bidirecțională), am ales să folosesc aceleași numere drept comenzi pentru ambele direcții.

În cazul în care comanda necesită informații adiționale (precum timestamp, nivelul de baterie), am introdus **câmpul text**, acolo unde se pot pune aceste informații.

Comunicarea este centrată pe 2 funcții de tip „handler" care se ocupă cu anumite acțiuni în anumite circumstanțe. Circumstanțele apărute în cod sunt cele de primire mesaj, prin funcția „*on_data_recv*", funcție înregistrată prin intermediul *esp_now_register_recv_cb* și cele de trimitere mesaj, prin funcția „*on_data_sent*", funcție înregistrată prin intermediul *esp_now_register_send_cb*.

Scopul funcției *on_data_sent* este acela de a afișa pe serială dacă trimiterea pachetului s-a încheiat cu succes sau nu.

Scopul funcției *on_data_recv* este acela de a interpreta pachetul primit prin despachetarea acestuia și trecerea în tipul mesajului, adică în *struct_message*.

Se realizează o analiză a datelor primite precum numărul de bytes primiți, care este sursa (adresa MAC) precum și tratarea pachetului prin analiza celor 3 câmpuri de *id*, *command* și *text*.

În funcție de comanda indicată, se execută ce trebuie executat și se răspunde înapoi la server dacă este cazul.

Mai jos este un tabel cu corespondența între câmpurile *command*, *text* și comanda executată din perspectiva senzorului:

Command	**text**	**Request-ul de pe server**	**Descriere**	**Răspuns către server**
0	<file_name>	<ESP_IP>/init?file= <file_name>	Se deschide fișierul de pe cardul SD și se pornește timer-ul	-
1	-	<ESP_IP>/start	Se activează flag-ul loop_blocker, răspunzător cu pornirea efectivă a preluării datelor și scrierea acestora	-
2	-	<ESP_IP>/stop	Se oprește colectarea	-
3	-	<ESP_IP>/all_bat	Se transmite nivelul de baterie al senzorului	command: *3* text:

				voltage
4	*timestamp*	<ESP_IP>/sync	Se transmite de la server la senzor timestampul curent	-
5	-	<ESP_IP>/all_timestamp	Se transmite de la senzor la server timestampul curent	command: *5* text: *timestamp*
6	<file_name>	<ESP_IP>/file_exists?file=<file_name>	Se interoghează senzorul dacă fișierul cu numele <file_name> există la el pe card SD	command: *6* text: *<file_name> exists / does not exist*
7	-	<ESP_IP>/enable_download	Se trece senzorul în *DOWNLOAD STATE*	-

Tabel 3 – Descrierea pachetelor trimise de la server la senzori

Perspectiva serverului este asemănătoare, păstrând convenția command – request. Serverul își actualizează memoria internă și transmite clientului informațiile cerute.

5. Comunicarea dintre client, server și senzori

5.1. Preluarea informatiilor de către client

Limitările protocolului ESP-NOW nu permit ca datele sa fie preluate de la senzori și expuse serverului și clientului conectat la server în aceeași conexiune HTTP.

În consecință, comenzile de preluare a timestamp-ului, nivelului de baterie sau verificare existență fișier nu pot fi satisfacute în cadrul aceluiași request.

Rezolvarea acestei limitări ar fi înlănțuirea a două cereri identice de către client.

Dacă, de exemplu, se dorește a se afla nivelul bateriei pentru toți senzorii, clientul va trebui să execute următoarea secvență de cereri HTTP la server:

Prima cerere:

<SERVER_IP>/all_bat

În cadrul acestei cereri, clientul trasmite serverului că se dorește nivelul de baterie al tuturor senzorilor (prin server http, via WiFi), serverul transmite mai departe tuturor senzorilor cererea (prin ESP-NOW). Aici nu se poate aștepta pentru răspunsul de la senzori, întrucât este asincron și nu se știe exact la ce diferență de timp o să vină, de aceea serverul reține informația intermediară în buffer și transmite ceea ce are deja la client prin răspuns HTTP fără să poată actualiza valorile de la senzori. Ulterior trimiterii răspunsului, într-o perioadă de timp relativ scurtă, primește și noile informații cu nivelul bateriei de la fiecare senzor și actualizează informația în buffer-ul local.

Dacă aceasta este prima rulare de *all_bat* (adică nu s-au mai făcut cereri de când a fost serverul pornit), rezultatul ar trebui să fie:

„
ESP32_1:
ESP32_2:
ESP32_3:
ESP32_4:
ESP32_5:
ESP32_6:
ESP32_7:
ESP32_8:
ESP32_9:
ESP32_10:
ESP32_11:
”

Dacă nu este prima rulare, rezultatele afișate sunt de fapt valorile care erau de actualitate la cererea anterioară celei făcute acum.

Se reintroduce aceeași cerere HTTP care ar trebui să raspundă cu valorile pe care clientul le-a vrut când a dat comanda 1:

<SERVER_IP>/all_bat

Acum, în spate, serverul a primit răspunsurile de la toți senzorii și au fost puse noile valori în buffer-ul intern al serverului. Raspunsul cererii:

„
ESP32_1: BATTERY_LEVEL_1
ESP32_2: BATTERY_LEVEL_2
ESP32_3: BATTERY_LEVEL_3
ESP32_4: BATTERY_LEVEL_4
ESP32_5: BATTERY_LEVEL_5
ESP32_6: BATTERY_LEVEL_6
ESP32_7: BATTERY_LEVEL_7
ESP32_8: BATTERY_LEVEL_8
ESP32_9: BATTERY_LEVEL_9
ESP32_10: BATTERY_LEVEL_10
ESP32_11: BATTERY_LEVEL_11
”

unde BATTERY_LEVEL_x este valoarea în volți a nivelului de baterie pentru senzorul x. Aceste valori sunt de fapt valorile de baterie pe care le aveau senzorii când s-a executat comanda 1. În spate, similar comenzii 1, serverul își actualizează bufferul cu noile date primite de la senzori în momentul cererii 2.

5.2. Înlănțuirea comenzilor

Din cauza numărului ridicat de transferuri efectuate la un moment prin protocolul ESP-NOW între server și toți senzorii pentru diverse comenzi, pentru a asigura trimiterea cu succes a tuturor pachetelor, este necesar și recomandat să se aștepte un minim de **2-3 secunde** între

transferuri de comenzi de tipul aflării nivelului de baterie, timestamp și existența fișier și **6-7 secunde** intre init si start.

5.3. Moduri de funcționare senzori

O altă limitare a protocolul ESP-NOW ar fi lungimea maximă a unui pachet care este de 250 bytes [12]. Acest aspect nu afectează colectarea în sine ci doar preluarea datelor de pe cardurile SD de către client. Pentru a evita necesitatea scoaterii cardurilor SD, este necesar un mod de a descărca fișierele de pe cardurile SD de la distanță.

O variantă ar fi fost segmentarea unui fișier în mai multe pachete de 250 bytes și trimiterea la server, variantă respinsă din riscul de pierdere al unui pachet din cauza înlănțuirii de mai multe pachete fără o durată de așteptare, iar incluzând această perioadă de așteptare, descărcarea ar fi durat mult prea mult.

Pentru a evita orice risc impus de ESP-NOW, am decis să separ funcționalitatea senzorilor în 2 mari categorii: *COLLECT MODE* (mod de colectare) și *DOWNLOAD MODE* (mod de descărcare).

Ideea programului ar fi să ruleze un server web doar când este în mod descărcare, astfel se asigură și un transfer mult mai rapid și mult mai sigur (comunicare directă între senzori și client fără server ca intermediar).

Pentru a evita conflictele între ESP-NOW și WiFi la nivel de senzor (peer), fiecare senzor pornește la boot cu modul prestabilit în rularea anterioară. La început, toți senzorii pornesc în COLLECT MODE.

5.4. Senzor în mod de colectare

Acesta este modul standard de funcționare, exact ca în figura 3.

Pentru trecerea în modul de colectare, este suficientă repornirea fiecărui senzor din switch-ul atașat sau requestul special pentru reintrarea în *COLLECT MODE* din *DOWNLOAD MODE* descris în secțiunea *Utilizare API*.

Clientul comunică cu serverul și serverul comunică cu fiecare senzor prin ESP-NOW, așadar nu există comunicare directă între client și senzor.

În acest mod, se pot colecta date, prin API-ul prezentat la secțiunea *Utilizare API Server*.

Senzorii colectează date la o frecvență de 100 Hz, adică preiau 100 de cadre de informație pe secundă.

5.5. Senzor în mod de descărcare

Setup-ul este realizat conform schemei următoare:

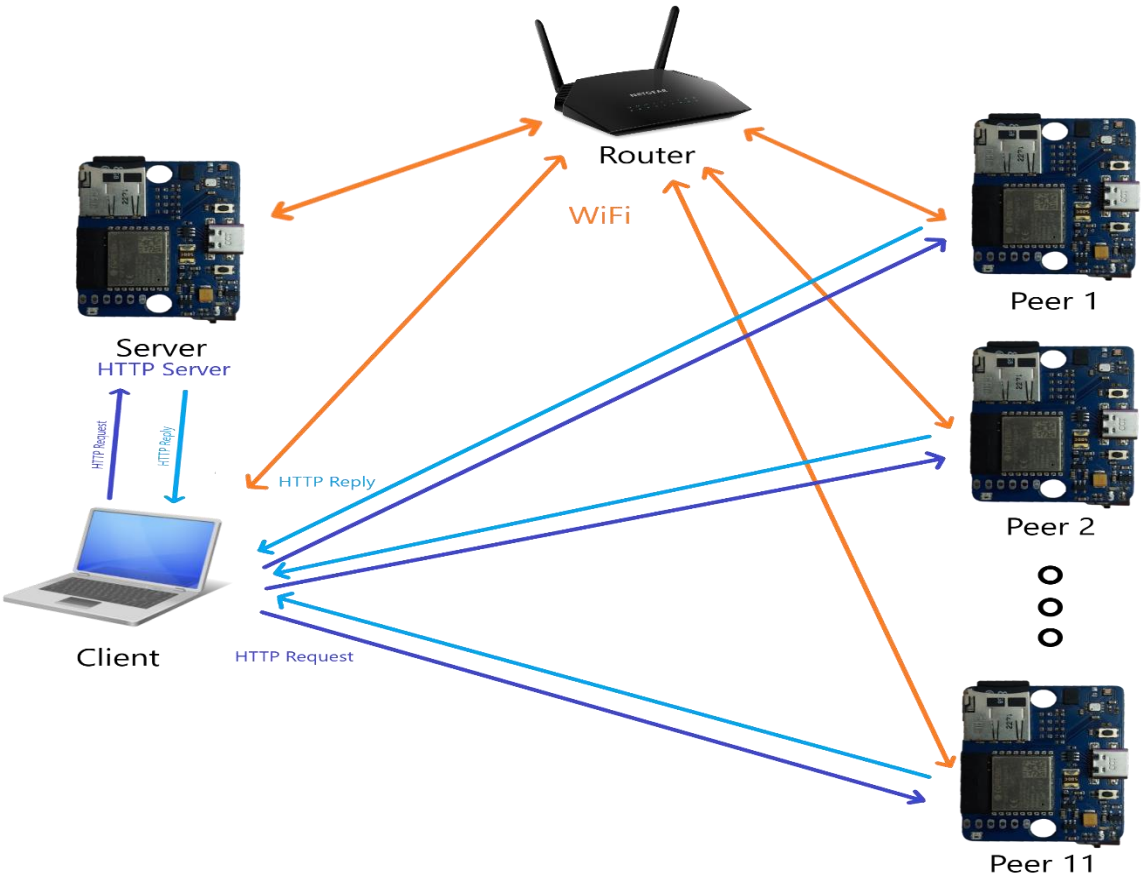

Fig. 4 Infrastructura în DOWNLOAD MODE

Fiecare senzor va expune un server web pe ip-ul pe care îl afișează pe serială în momentul inițializării (în setup-ul curent el este fix și se cunoaște la fiecare conectare, fiind facută corespondența din router prin adresa MAC unică a fiecărui senzor).

În acest mod, senzorii nu vor comunica cu serverul, comunicarea prin protocolul ESP-NOW fiind oprită la nivelul fiecărui senzor. Senzorii vor comunica fiecare direct cu clientul și clientul va comunica cu fiecare senzor individual prin server HTTP ținut pe fiecare senzor.

API-ul folosit pentru acest mod este cel prezentat mai jos la secțiunea *Utilizare API Senzor în DOWNLOAD MODE*.

Se poate observa că încă există comunicare între server și client, însă utlizarea ei nu este recomandată în acest mod din cauza lipsei de comunicare dintre server și senzori.

6. Observații privind utilizarea programului

Comenzile la care se fac referiri mai jos sunt corelate cu cele de la *Utilizare API Server* pentru cele standard iar pentru cele legate de modul de descărcare, Utilizare API Senzor în DOWNLOAD MODE.

6.1. Statusul LED-ului indicator

Fiecare senzor are atașat un LED conform schemei din figura 2.

Acest LED poate avea următoarele afișări:

1) ROȘU -> problemă la inițializare, cauze pozibile: erori de conectare la WiFi, cititorul de carduri SD sau la accelerometru & giroscop;
2) VERDE -> inițializarea s-a finalizat cu succes, acesta stă aprins o fracțiune de secundă;
3) STINS -> senzorul este pornit și funcționează sau este complet stins (din switch), aici se pot consulta comenzi precum aflarea nivelului de baterie (*comanda 4*) sau timestamp (*comanda 3*) pentru a verifica prin server statusul senzorilor.

6.2. Recomandări privind utilizarea infrastructurii

În cadrul utilizării acestui setup se recomandă:

1) **sincronizarea** (prin sync, *comanda 2*) tuturor senzorilor înainte de a începe o secvență de colectare.
2) verificarea existentei fișierelor de la colectarea curentă pentru toți senzorii (*comanda 5*).
3) Verificarea log-urilor din seriala serverului din client și monitorizarea trimiterii pachetelor, se vrea apariția mesajului *„Delivery Success”*, mare atenție la mesajul *„Delivery Fail”* care semnifică faptul că un pachet trimis de server nu a putut fi primit de un senzor.
   Recomandare: reîncercare secvență anterioară (reluarea colectarii rundei curente / comenzii date care a generat problema) iar dacă problema persistă, verificarea nivelului bateriei (*comanda 4*). Dacă acesta este sub 3.6 - 3.7V, aceasta ar putea fi sursa problemei (o reîncărcare a senzorilor ar trebui să rezolve problema).
4) Descărcarea fișierelor la fiecare 3-4 colectări complete prin trecerea senzorilor în *DOWNLOAD MODE* (*comanda 9*) și rularea descărcării de pe fiecare senzor (*comanda 1* din modul de descărcare)

6.3. Aspecte necesare privind utilizarea infrastructurii

În cadrul utilizării acestui setup sunt necesare următoarele:

1) **sincronizarea** prin *comanda 2* după momentul primei porniri a senzorilor.
2) verificarea fiecărui led de pe senzori, să nu fie aprins în momentul colectării, statusul led-ului este explicat la secțiunea *Statusul LED-ului indicator*.
3) verificarea serverului la începutul colectărilor (*comanda 1*) și așteptarea răspunsului *„Yes”*.
4) descărcarea tuturor fișierelor înregistrare în ziua respectivă.
5) verificarea constantă a nivelului de baterie (*comanda 4*) a senzorilor la fiecare 3-4 colectări.
6) încărcarea senzorilor în momentul în care minimul de nivel de baterie atinge 3.6V.

Cunoașterea modului curent de funcționare al senzorilor (*DOWNLOAD MODE* sau *COLLECT MODE*) pentru a ști cum se interacționează cu API-ul.

7) așteptarea timpilor precizați la *Înlănțuirea comenzilor* pentru comenzi dar și așteptarea a 7-8 secunde la momentul trecerii între *DOWNLOAD MODE* și *COLLECT MODE*.
8) resincronizarea senzorilor (*comanda 2*) după trecerea din *DOWNLOAD MODE* în *COLLECT MODE*.

9) toți senzorii trebuie să fie în același mod, dacă apar variațiuni (o parte din senzori sunt în *DOWNLOAD MODE* și cealaltă în *COLLECT MODE*) se impune repornirea prin switch a tuturor senzorilor.

7. Perspectivă de viitor

În momentul de față, senzorii cu care se execută colectarea dețin ca microcontroller un ESP32 C3, single core, care este low power [13].

Limitarea de memorie internă a microcontroller-ului impune o scriere periodică a datelor pe cardurile SD care în momentul de față este setată la 10 secunde. Astfel, colectarea are loc normal, salvându-se în memoria internă pentru 9 secunde, urmând ca în următoarea secundă să se execute și colectarea la frecvența de 100 Hz dar și scrierea pe SD. Din cauza faptului că scrierea durează mai mult, afectează numărul de cadre extrase în secunda respectivă.

O soluție ar fi paralelizarea acțiunilor, pe un nucleu să se execute colectarea și pe celălalt scrierea pe cardul SD, utilizând double-buffering [14]. Având doi vectori asemănători A și B, în momentul în care se face scrierea vectorului A pe cardul SD, colectarea de date se poate face pe bufferul B și invers.

Această soluție ar implica pe viitor realizarea unor senzori cu ESP32 S3 [15], care este dual core și ar permite astfel de operațiuni.

8. Referințe

[1] https://arxiv.org/abs/2308.10623
[2] https://github.com/FarhadGUL06/esp32_motion_tracking.git
[3] https://www.ibm.com/topics/api
[4] https://www.espressif.com/en/solutions/low-power-solutions/esp-now
[5] https://github.com/FarhadGUL06/esp32_motion_tracking_webserver.git
[6] https://ro.mouser.com/datasheet/2/783/BST_BMI085_DS001-1509577.pdf
[7] https://platformio.org/
[8] https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/generic-class.html
[9] https://randomnerdtutorials.com/esp32-async-web-server-espasyncwebserver-library/
[10] https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/memory-types.html
[11] https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/misc_system_api.html
[12] https://docs.espressif.com/projects/esp-faq/en/latest/application-solution/esp-now.html
[13] https://www.espressif.com/sites/default/files/documentation/esp32-c3-wroom-02_datasheet_en.pdf
[14] https://www.geeksforgeeks.org/double-buffering/
[15] https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf